

New Challenges in Learning Classifier Systems: Mining Rarities and Evolving Fuzzy Models

Albert Orriols i Puig

Grup de Recerca en Sistemes Intel·ligents
Enginyeria i Arquitectura La Salle
Universitat Ramon Llull

November 2008

Supervisor: Dr. Ester Bernadó i Mansilla



Universitat Ramon Llull

Aquesta Tesi Doctoral ha estat defensada el dia 12 de NOVEMBRE de 2006
al Centre ENGINYERIA I ARQUITECTURA LA SANTE
de la Universitat Ramon Llull

davant el Tribunal format pels Doctors sotasignants, havent obtingut la qualificació:

EXCEL·LENT AM LAUDE

President/a

DAVID E. GOLDBERG

Vocal

FRANCESC XAVIER LLOPÀ I FABREGA

Vocal

MARTIN V. BUTZ

Vocal

FRANCISCO VERNERA

Secretari/ària

XAVIER VILASIS GARDIA

Doctorand/a

Resum

Durant l'última dècada, els sistemes classificadors (LCS) d'estil Michigan—sistemes d'aprenentatge automàtic que combinen *tècniques de repartiment de crèdit* i *algorismes genètics* (AG) per evolucionar una població de classificadors *online*—han renaixut. Juntament amb la formulació dels sistemes de primera generació, s'han produït avenços importants en (1) el disseny sistemàtic de nous LCS competents, (2) la seva aplicació en dominis rellevants i (3) el desenvolupament d'anàlisis teòriques. Malgrat aquests dissenys i aplicacions importants, encara hi ha reptes complexos que cal abordar per comprendre millor el funcionament dels LCS i per solucionar problemes del món real eficientment i escalable.

Aquesta tesi tracta dos reptes importants—compartits amb la comunitat d'aprenentatge automàtic—amb LCS d'estil Michigan: (1) aprenentatge en dominis que contenen classes estranyes i (2) evolució de models comprensibles on s'utilitzin mètodes de raonament similars als humans. L'aprenentatge de models precisos de classes estranyes és crític, doncs el coneixement clau sol quedar amagat en exemples d'aquestes, i la majoria de tècniques d'aprenentatge no són capaces de modelar la raresa amb precisió. La detecció de rareses sol ser complicat en aprenentatge *online* ja que el sistema d'aprenentatge rep un flux d'exemples i ha de detectar les rareses al vol. D'altra banda, l'evolució de models comprensibles és crucial en certs dominis com el mèdic, on l'expert acostuma a estar més interessat en obtenir una explicació intel·ligible de la predicció que en la predicció en si mateixa.

El treball present considera dos LCS d'estil Michigan com a punt de partida: l'XCS i l'UCS. Es pren l'XCS com a primera referència ja que és l'LCS que ha tingut més influència fins al moment. L'UCS hereta els components principals de l'XCS i els especialitza per aprenentatge supervisat. Tenint en compte que aquesta tesi especialment se centra en problemes de classificació, l'UCS també es considera en aquest estudi. La inclusió de l'UCS marca el primer objectiu de la tesi, sota el qual es revisen un conjunt de punts que van restar oberts en el disseny inicial del sistema. A més, per il·lustrar les diferències claus entre l'XCS i l'UCS, es comparen ambdós sistemes sobre una bateria de problemes artificials de complexitat acotada.

L'estudi de com els LCS aprenen en dominis amb classes estranyes comença amb una anàlisi que *descompon* el problema en cinc elements crítics i deriva *models per facetes* per cadascun d'ells. Aquesta anàlisi s'usa com a eina per dissenyar guies de configuració que permeten que l'XCS i l'UCS solucionin problemes que prèviament no eren resolubles. A continuació, es comparen els dos LCS amb alguns dels sistemes d'aprenentatge amb més influència en la comunitat d'aprenentatge automàtic sobre una col·lecció de problemes del món real que contenen classes estranyes. Els resultats indiquen que els dos LCS són els mètodes més robustos de la comparativa. Així mateix, es demostra experimentalment que *remostrejar* els conjunts d'entrenament amb l'objectiu d'eliminar la presència de classes estranyes beneficia, en mitjana, el rendiment de les tècniques d'aprenentatge.

El repte de crear models més comprensibles i d'usar mecanismes de raonament que siguin similars als humans s'aborda mitjançant el disseny d'un nou LCS per aprenentatge supervisat que combina les capacitats d'avaluació de regles online, la robustesa mostrada pels AG en problemes complexos i la representació comprensible i mètodes de raonament fonamentats proporcionats per la lògica difusa. El nou LCS, anomenat Fuzzy-UCS, s'estudia en detall i es compara amb una bateria de mètodes d'aprenentatge. Els resultats de la comparativa demostren la competitivitat del Fuzzy-UCS en termes de precisió i intel·ligibilitat dels models evolucionats. Addicionalment, s'usa Fuzzy-UCS per extreure models de classificació acurats de grans volums de dades, exemplificant els avantatges de l'arquitectura d'aprenentatge *online* del Fuzzy-UCS.

En general, les observacions i avenços assolits en aquesta tesi contribueixen a augmentar la comprensió del funcionament dels LCS i en preparar aquests sistemes per afrontar problemes del món real de gran complexitat. Finalment, els resultats experimentals ressalten la robustesa i competitivitat dels LCS respecte a altres mètodes d'aprenentatge, encoratjant el seu ús per tractar nous problemes del món real.

Resumen

Durante la última década, los sistemas clasificadores (LCS) de estilo Michigan—sistemas de aprendizaje automático que combinan *técnicas de repartición de crédito* y *algoritmos genéticos* (AG) para evolucionar una población de clasificadores *online*—han renacido. Juntamente con la formulación de los sistemas de primera generación, se han producido avances importantes en (1) el diseño sistemático de nuevos LCS competentes, (2) su aplicación en dominios relevantes y (3) el desarrollo de análisis teóricos. Pese a eso, aún existen retos complejos que deben ser abordados para comprender mejor el funcionamiento de los LCS y para solucionar problemas del mundo real escalable y eficientemente.

Esta tesis trata dos retos importantes—compartidos por la comunidad de aprendizaje automático—con LCS de estilo Michigan: (1) aprendizaje en dominios con clases raras y (2) evolución de modelos comprensibles donde se utilicen métodos de razonamiento similares a los humanos. El aprendizaje de modelos precisos de clases raras es crítico pues el conocimiento clave suele estar escondido en ejemplos de estas clases, y la mayoría de técnicas de aprendizaje no son capaces de modelar la rareza con precisión. El modelado de las rarezas acostumbra a ser más complejo en entornos de aprendizaje *online*, pues el sistema de aprendizaje recibe un flujo de ejemplos y debe detectar las rarezas al vuelo. La evolución de modelos comprensibles es crucial en ciertos dominios como el médico, donde el experto está más interesado en obtener una explicación inteligible de la predicción que en la predicción en sí misma.

El trabajo presente considera dos LCS de estilo Michigan como punto de partida: el XCS y el UCS. Se toma XCS como primera referencia debido a que es el LCS que ha tenido más influencia hasta el momento. UCS es un diseño reciente de LCS que hereda los componentes principales de XCS y los especializa para aprendizaje supervisado. Dado que esta tesis está especialmente centrada en problemas de clasificación automática, también se considera UCS en el estudio. La inclusión de UCS marca el primer objetivo de la tesis, bajo el cual se revisan un conjunto de aspectos que quedaron abiertos durante el diseño del sistema. Además, para ilustrar las diferencias claves entre XCS y UCS, se comparan ambos sistemas sobre una batería de problemas artificiales de complejidad acotada.

El estudio de cómo los LCS aprenden en dominios con clases raras empieza con un estudio analítico que *descompone* el problema en cinco elementos críticos y deriva *modelos por facetas* para cada uno de ellos. Este análisis se usa como herramienta para diseñar guías de configuración que permiten que XCS y UCS solucionen problemas que previamente no eran resolubles. A continuación, se comparan los dos LCS con algunos de los sistemas de aprendizaje de mayor influencia sobre una colección de problemas del mundo real que contienen clases raras. Los resultados indican que los dos LCS son los métodos más robustos de la comparativa. Además, se demuestra experimentalmente que *remuestrear* los conjuntos de entrenamiento con el objetivo de eliminar la presencia de clases raras beneficia, en promedio, el rendimiento de los métodos de aprendizaje automático incluidos en la comparativa.

El reto de crear modelos más comprensibles y usar mecanismos de razonamiento que sean similares a los humanos se aborda mediante el diseño de un nuevo LCS para aprendizaje supervisado que combina las capacidades de evaluación de reglas *online*, la robustez mostrada por los AG en problemas complejos y la representación comprensible y métodos de razonamiento proporcionados por la lógica difusa. El sistema que resulta de la combinación de estas ideas, llamado Fuzzy-UCS, se estudia en detalle y se compara con una batería de métodos de aprendizaje altamente reconocidos en el campo de aprendizaje automático. Los resultados de la comparativa demuestran la competitividad de Fuzzy-UCS en referencia a la precisión e inteligibilidad de los modelos evolucionados. Adicionalmente, se usa Fuzzy-UCS para extraer modelos de clasificación precisos de grandes volúmenes de datos, ejemplificando las ventajas de la arquitectura de aprendizaje *online* de Fuzzy-UCS.

En general, los avances y observaciones proporcionados en la tesis presente contribuyen a aumentar la comprensión del funcionamiento de los LCS y a preparar estos tipos de sistemas para afrontar problemas del mundo real de gran complejidad. Además, los resultados experimentales resaltan la robustez y competitividad de los LCS respecto a otros métodos de aprendizaje, alentando su uso para tratar nuevos problemas del mundo real.

Abstract

During the last decade, Michigan-style learning classifier systems (LCSs)—genetic-based machine learning (GBML) methods that combine *apportionment of credit techniques* and *genetic algorithms* (GAs) to evolve a population of *classifiers* online—have been enjoying a renaissance. Together with the formulation of first generation systems, there have been crucial advances in (1) systematic design of new competent LCSs, (2) applications in important domains, and (3) theoretical analyses for design. Despite these successful designs and applications, there still remain difficult challenges that need to be addressed to increase our comprehension of how LCSs behave and to scalably and efficiently solve real-world problems.

The purpose of this thesis is to address two important challenges—shared by the machine learning community—with Michigan-style LCSs: (1) learning from domains that contain rare classes and (2) evolving highly legible models in which human-like reasoning mechanisms are employed. Extracting accurate models from rare classes is critical since the key, unperceptive knowledge usually resides in the rarities, and many traditional learning techniques are not able to model rarity accurately. Besides, these difficulties are increased in online learning, where the learner receives a stream of examples and has to detect rare classes on the fly. Evolving highly legible models is crucial in some domains such as medical diagnosis, in which human experts may be more interested in the explanation of the prediction than in the prediction itself.

The contributions of this thesis take two Michigan-style LCSs as starting point: the *extended classifier system* (XCS) and the *supervised classifier system* (UCS). XCS is taken as the first reference of this work since it is the most influential LCS. UCS is a recent LCS design that has inherited the main components of XCS and has specialized them for supervised learning. As this thesis is especially concerned with classification problems, UCS is also considered in this study. Since UCS is still a young system, for which there are several open issues that need further investigation, its learning architecture is first revised and updated. Moreover, to illustrate the key differences between XCS and UCS, the behavior of both systems is compared on a collection of boundedly difficult problems.

The study of learning from rare classes with LCSs starts with an analytical approach in which the problem is *decomposed* in five critical elements, and *facetwise models* are derived for each element. The analysis is used as a tool for designing configuration guidelines that enable XCS and UCS to solve problems that previously eluded solution. Thereafter, the two LCSs are compared with several highly-influential learners on a collection of real-world problems with rare classes, appearing as the two best techniques of the comparison. Moreover, *re-sampling* the training data set to eliminate the presence of rare classes is demonstrated to benefit, on average, the performance of LCSs.

The challenge of building more legible models and using human-like reasoning mechanisms is addressed with the design of a new LCS for supervised learning that combines the online evaluation capabilities of LCSs, the search robustness over complex spaces of GAs, and the legible knowledge representation and principled reasoning mechanisms of fuzzy logic. The system resulting from this crossbreeding of ideas, referred to as Fuzzy-UCS, is studied in detail and compared with several highly competent learning systems, demonstrating the competitiveness of the new architecture in terms of the accuracy and the interpretability of the evolved models. In addition, the benefits provided by the online architecture are exemplified by extracting accurate classification models from large data sets.

Overall, the advances and key insights provided in this thesis help advance our understanding of how LCSs work and prepare these types of systems to face increasingly difficult problems, which abound in current industrial and scientific applications. Furthermore, experimental results highlight the robustness and competitiveness of LCSs with respect to other machine learning techniques, which encourages their use to face new challenging real-world applications.

Acknowledgments

If belonging to a competitive research group and collaborating, sharing, and enjoying with your group mates is one of the best experiences of a PhD lifetime, I consider myself one of the luckiest PhD students, since I have been working in three highly-competitive research groups. For this reason, I am tremendously grateful to all and each one of the people in there. If I tried to write sounding words to express all my gratitude, these would only give a vague idea of what I mean. Thence, following my engineering vocation, I would start simplifying and saying “thank you”.

First, I would like to thank Ester Bernadó-Mansilla for accepting me as her first PhD student, for her advice, and for permitting my successive visits to the *Illinois genetic algorithms laboratory* (IlliGAL) and the *soft computing and intelligent information systems* (SCI2S) group. I would like to extend this acknowledgment to all the people in the *grup de recerca en sistemes intel·ligents* (GRSI) and the computer engineering department of *enginyeria i arquitectura la Salle* in general, with which I had interesting discussions, combined with lots of fun, during the last four years.

I am in terrible debt with all the people of the two research groups which I consider as my second homes: the IlliGAL and the SCI2S. My visits to Champaign and Granada were extremely fruitful and defined the largest part of the present document.

I am really glad that Prof. Goldberg accepted my visits to the IlliGAL and considered me one of his students. I still remember when I reached the lab for the first time with a particular problem to solve and left it with a methodology to face new challenging engineering problems. I think that Prof. Goldberg’s great ability to see the big picture, to make key points, to formulate difficult, really interesting questions, to express with strength the ideas in a piece of paper, and, to, in five minutes, go far beyond my reach will never stop surprising me. I will never know what I have missed from Prof. Goldberg explanations, but I am extremely glad of all the things I know I got from him.

I also thank all the great people I met in the IlliGAL. I would especially single out Kumara Sastry. I am very lucky to have been able to cope with, as defined by him, his *not-so-nice* explanation skills. I am really grateful to Kumara for all his explanations, for his help, and for making the lab a funny place to work. I also enjoyed working with and learned a lot from Pier Luca Lanzi, who showed me new faces of learning classifier systems that I had never seen before. I am also grateful for the support of Xavier Llorà and Tian Li Yu in my visits.

I am very grateful to Jorge Casillas for insisting on my first visit to the SCI2S group with the original idea of mixing learning classifier systems and fuzzy logic and for all the great support and guidance provided not only during my successive visits, but also during the entire last year of my PhD. I learned a lot from all our long, daily talks and from Jorge’s eagerness and passion to face new challenging real-world problems. I would like to extend this acknowledgement to Francisco Herrera for all his valuable advices and for always being ready to help me and to answer my questions. My visits to Granada were not only fruitful research-wise but also life-wise. I really enjoyed our Thursday’s home parties and hanging around Granada with the SCI2S members. Especially, I would like to thank my flatmates, Pietro Ducange and Manolo Cobo, for making my stay in Granada so joyful.

The present work is the result of the collaboration with a number of researchers. I would like to thank my coauthors Ester Bernadó-Mansilla, Jorge Casillas, David E. Goldberg, Pier Luca Lanzi, Núria Macià, Francisco J. Martínez-López, Sergio Morales-Ortigosa, Joaquim Rios-

Boutin, Kumara Sastry, and Francesc Teixidó-Navarro. I would also thank Xavier Llorà for really interesting research talks.

Last, but not least, I would like to thank the unconditional support of all my family. I would like to thank my grandparents Josep and Antònia, my parents Albert and Maria Cinta, and my sister Gemma for their motivation to go on with my PhD. Also, I would like to especially thank the person who suffered my busy weekends, my stress when deadlines were approaching, and the distance while I was abroad the most; M. Carme, thanks for being there despite all this.

This research has been financially supported by the *departament d'universitats, recerca i societat de la informació* (DURSI) under a scholarship in the FI research program with reference *2005FI-00252*. I also acknowledge the support provided by DURSI in my visits to the IlliGAL, with two travel grants with references *2006BE-00299* and *2007BE2-00124*. Finally, I would like to acknowledge the *ministerio de educación y ciencia* for its support under the KEEL and the KEEL II projects (with references TIC2002-04036-C05-03 and TIN2005-08386-C05-04), and *Generalitat de Catalunya* for its support under the grant 2005SGR-00302.

Contents

List of Figures	vii
List of Tables	xi
List of Algorithms	xvii
1 Introduction	1
1.1 Framework: From Holland’s Definition to Current LCSs	2
1.2 Two Critical Challenges in LCSs and Machine Learning	5
1.3 Thesis Objectives	7
1.4 Road Map	8
2 Machine Learning with Learning Classifier Systems	11
2.1 Machine Learning	11
2.1.1 Supervised learning	14
2.1.2 Unsupervised learning	14
2.1.3 Reinforcement Learning	14
2.2 Evolutionary Computation and Genetic Algorithms	15
2.2.1 Biological Principles that Inspire Evolutionary Computation	16
2.2.2 Evolutionary Computation: A Taxonomy	17
2.2.3 Genetic Algorithms	18
2.2.4 Basic Theory of GA	21
2.2.5 Genetic Algorithms in Real-World Applications	24
2.3 Genetic-based Machine Learning and Learning Classifier Systems	25
2.3.1 Michigan-style LCSs	26
2.3.2 Pittsburgh-style LCSs	28
2.3.3 Iterative Rule Genetic-based Machine Learning	29
2.3.4 Genetic Cooperative-Competitive Learning	30

2.3.5	The Organizational Classifier System	30
2.4	Summary	31
3	Description of XCS and UCS	33
3.1	The XCS Classifier System	34
3.1.1	Knowledge Representation	34
3.1.2	Learning Interaction	35
3.1.3	Classifier Evaluation	36
3.1.4	Classifier Discovery	38
3.1.5	Class Inference in Test Mode	39
3.1.6	Why Does XCS Work?	39
3.2	The UCS Classifier System	41
3.2.1	Knowledge Representation	41
3.2.2	Learning Interaction	42
3.2.3	Classifier Evaluation	43
3.2.4	Classifier Discovery	43
3.2.5	Class Inference in Test Mode	44
3.2.6	Why does UCS work?	45
3.3	Rule Representations for LCSs	45
3.3.1	From the Ternary to the Interval-based Rule Representation in LCSs	46
3.3.2	The Unordered Bound Representation	47
3.4	Summary and Conclusions	50
4	Revisiting UCS: Fitness Sharing and Comparison with XCS	51
4.1	Fitness Sharing in GAs and LCSs	52
4.2	A New Fitness Sharing Scheme for UCS	53
4.3	Methodology	54
4.4	Analyzing the Fitness Sharing Scheme in UCS	55
4.5	Comparing UCSs with XCS	60
4.6	Lessons Learned from the Analysis	64
4.6.1	Fitness Sharing	64
4.6.2	Explore Regime	65
4.6.3	Accuracy Guidance	65
4.6.4	Population Size	66
4.7	Summary and Conclusions	66

5	Facetwise Analysis of XCS for Domains with Class Imbalances	67
5.1	The Challenges of Learning from Imbalanced Domains in Machine Learning	68
5.2	The XCS Classifier System in Imbalanced Domains	70
5.2.1	Hypotheses of XCS Difficulties in Learning from Imbalanced Domains	70
5.2.2	Empirical Observations of XCS Behavior on Class Imbalances	71
5.3	Facetwise Analysis of XCS in Imbalanced Domains	73
5.3.1	Design Decomposition in GAs	73
5.3.2	Carrying the Design Decomposition from GAs to XCS	73
5.3.3	A Boundedly Difficult Problem for LCSs: The Imbalanced Parity Problem	74
5.3.4	Decomposition of the Class Imbalance Problem in XCS	75
5.4	Estimation of Classifier Parameters	77
5.4.1	Imbalance Bound	77
5.4.2	Does the Widrow-Hoff Rule Provide Accurate Estimates?	78
5.4.3	Obtaining Better Estimates with the Widrow-Hoff Rule	79
5.4.4	Obtaining Better Estimates with Gradient Descent Methods	80
5.5	Supply of Schemas of Starved Niches in Population Initialization	81
5.6	Generation of Classifiers in Starved Niches	82
5.6.1	Assumptions for the Model	83
5.6.2	Genetic Creation of Representatives of Starved Niches	83
5.6.3	Deletion of Representatives of Starved Niches	85
5.6.4	Bounding the Population Size	85
5.6.5	Experimental Validation of the Models	86
5.7	Occurrence-based Reproduction: The Role of θ_{GA}	89
5.7.1	Including θ_{GA} in the Generation Models	90
5.7.2	Experimental Validation	91
5.8	Takeover Time of Accurate Classifiers in Starved Niches	92
5.8.1	Model Assumptions	93
5.8.2	Takeover Time for Proportionate Selection	93
5.8.3	Takeover Time for Tournament Selection	97
5.8.4	Experimental Validation of the Takeover Time Models	101
5.9	Lessons Learned from the Models	104
5.9.1	Patchquilt Integration of the Facetwise Models	104
5.9.2	Solving Problems with Large Imbalance Ratios	105
5.10	Summary and Conclusions	106

6	Carrying over the Facetwise Analysis to UCS	109
6.1	Design Decomposition for UCS	110
6.2	Estimation of Classifier Parameters	111
6.3	Supply of Schemas of Starved Niches in Population Initialization	112
6.4	Generation of Classifiers in Starved Niches	113
6.4.1	Assumptions for the Model	114
6.4.2	Creation and Deletion of Representatives of Starved Niches	114
6.4.3	Bounding the Population Size	115
6.4.4	Experimental Validation of the Models	116
6.5	Occurrence-based Reproduction	119
6.6	Takeover Time of Accurate Classifiers in Starved Niches	120
6.6.1	Conditions for Starved Niches Extinction under Proportionate Selection	121
6.6.2	Conditions for Starved Niches Extinction under Tournament Selection	122
6.7	Reassembling the Theoretical Framework: UCS in Imbalanced Domains	122
6.7.1	Patchquilt Integration: from XCS to UCS	122
6.7.2	Solving Highly Imbalanced Domains with UCS	123
6.8	Summary and Conclusions	125
7	XCS and UCS for Mining Imbalanced Real-World Problems	127
7.1	LCSs in Imbalanced Real-World Problems: What Makes the Difference?	128
7.1.1	XCS and UCS Enhancements to Deal with Continuous Data	128
7.1.2	What Do we Need to Apply the Theory?	130
7.2	Self-Adaptation to Particular Unknown Domains	132
7.2.1	Online Adaptation Algorithms	132
7.2.2	Experiments	134
7.3	LCSs in Imbalanced Real-World Domains	136
7.3.1	Comparison Methodology	136
7.3.2	Results	138
7.4	Re-sampling Techniques	141
7.4.1	Random Over-sampling	142
7.4.2	Under-sampling based on Tomek Links	143
7.4.3	SMOTE	143
7.4.4	cSMOTE	144
7.4.5	What Do Re-sampling Techniques Do? A Case Study	146
7.5	Results on Re-sampled Domains	150

7.5.1	Experimental Methodology	150
7.5.2	Statistical Analysis of the Results	151
7.5.3	Summary	154
7.6	Discussion	155
7.7	Summary and Conclusions	158
8	Fuzzy-UCS: Evolving Fuzzy Rule Sets for Supervised Learning	159
8.1	Why Using Fuzzy Logic in LCSs?	160
8.2	Fuzzy Logics in GBML	162
8.2.1	Fuzzy Logic and Fuzzy Systems	162
8.2.2	Genetic Algorithms in Fuzzy Systems	163
8.2.3	Related Work on Learning Fuzzy-Classifer Systems	165
8.3	Description of Fuzzy-UCS	166
8.3.1	Knowledge Representation	167
8.3.2	Learning Interaction	169
8.3.3	Classifiers Update	170
8.3.4	Classifiers Discovery	171
8.3.5	Fuzzy-UCS in Test Mode	172
8.4	Sensitivity of Fuzzy-UCS to Configuration Parameters	174
8.5	Knowledge Representation and Decision Boundaries	177
8.5.1	Approximate Fuzzy-UCS	179
8.5.2	Decision Boundaries: Study on an Artificial Domain	181
8.5.3	Comparison Between Linguistic and Approximate Representations	185
8.6	Comparison of Fuzzy-UCS to Several Machine Learning Techniques	193
8.6.1	Experimental Methodology	193
8.6.2	Comparison to Fuzzy Rule-Based Classification Systems	194
8.6.3	Comparison with Non-Fuzzy Learners	200
8.7	Fuzzy-UCS for Mining Large Data Sets	208
8.7.1	Data Set Description	209
8.7.2	Results	209
8.8	Summary, Conclusions, and Further Work	211
8.8.1	Summary	211
8.8.2	SWOT Analysis	212
9	Summary, Conclusions, and Further Work	215
9.1	Summary and Conclusions	215

9.2	Lessons from LCSs Design and Application	220
9.3	Further Work	222
A	Description of the Artificial Problems	225
A.1	Parity	225
A.2	Decoder	226
A.3	Position	226
A.4	Multiplexer	227
A.4.1	Imbalanced Multiplexer	228
A.4.2	Multiplexer with Alternating Noise	228
B	Statistical Tests	229
B.1	Statistical Tests for Contrasting Hypotheses	229
B.2	Comparisons of Two Learning Systems	230
B.2.1	The Wilcoxon Signed-Ranks Test	230
B.3	Comparisons of Multiple Classifiers	232
B.3.1	Friedman's Test	233
B.3.2	Post-hoc Nemenyi Test	233
B.3.3	Post-hoc Bonferroni-Dunn Test	235
B.4	Summary	236
C	Full Results of the Comparison of the Re-sampling Techniques	237
D	Empirical Analysis of the Sensitivity of Fuzzy-UCS to Configuration Parameters	243
D.1	Configuration Parameters of Fuzzy-UCS	244
D.2	Experimental Methodology	244
D.3	Fuzzy-UCS's Sensitivity to Configuration Parameters	245
D.3.1	Sensitivity to Rule Initialization	245
D.3.2	Sensitivity to Fitness Pressure	247
D.3.3	Sensitivity to the GA	249
D.3.4	Sensitivity to Deletion	250
D.4	Summary and Conclusions	253
	References	255
	Index	279

List of Figures

2.1	Examples of (a) supervised, (b) unsupervised, and (c) reinforcement learning. . .	13
2.2	Evolution of a GA population.	19
2.3	Simplified schematic of Michigan-style LCSs which the typical process organization.	26
2.4	Simplified schematic of Pittsburgh-style LCSs.	28
3.1	Schematic of the process organization of XCS.	35
3.2	Schematic of the process organization of UCS.	42
3.3	Example of covering in the hyper rectangular representation.	48
3.4	A crossover example. (a) plots the two parents and (b) shows the offspring resulting from two cut points occurring in the middle of each interval.	49
3.5	Example mutation in the hyper rectangle representation.	49
4.1	Proportion of the best action map achieved by UCSns and UCSs in the parity, the position, and the decoder problems.	56
4.2	Proportion of the best action map achieved by (a) UCSns and (b) UCSs in the noisy 20-bit multiplexer with $P_x = \{0.05, 0.10, 0.15\}$	57
4.3	Proportion of the best action map achieved by (a) UCSns and (b) UCSs in the noisy 20-bit multiplexer with $P_x = \{0.05, 0.10, 0.15\}$ and using $\beta = 0.01$ and $\theta_{GA} = 100$	59
4.4	Proportion of the best action map achieved by UCSs and XCS in the parity, the position, and the decoder problems.	61
4.5	Proportion of the best action map achieved by (a,c) UCSs and (b,d) XCS in the noisy 20-bit multiplexer with $P_x = \{0.05, 0.10, 0.15\}$ with (a,b) the original configuration and with (c,d) the original configuration but setting $\beta = 0.01$ and $\theta_{GA} = 100$	62
4.6	Error of XCS's classifiers along the over-general/optimal classifier dimension. The curve depicts how the error of the most over-general classifier #####:0 evolves as the bits of the classifier are specified, until obtaining the maximally accurate rule 000000000:0.	63
4.7	Error of XCS's classifiers along the over-general/optimal classifier dimension. . .	63

5.1	Evolution of (a) the proportion of the optimal population and (b) the product of TP rate and TN rate in the 11-bit multiplexer with imbalance ratios ranging from $ir=1$ to $ir=1024$	72
5.2	Histogram of the error of the most over-general classifier with Widrow-Hoff delta rule at $\beta = 0.2$ and different imbalance ratios.	79
5.3	Histogram of the error of the most over-general classifier with Widrow-Hoff delta rule at $\beta = 0.01$ and different imbalance ratios.	80
5.4	Histogram of the error of the most over-general classifier with gradient descent at $\beta = 0.2$ and different imbalance ratios.	81
5.5	Probability of activating covering on a minority class instance given a certain specificity $\sigma[P]$ and the imbalance ratio ir . The curves have been drawn from equation 5.13 with $\ell = 20$ and different specificities.	83
5.6	Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and the default configuration with (a) Widrow Hoff rule update with adjusted β according to ir and (b) gradient descent parameter update with $\beta = 0.2$. The dots show the empirical results and lines plot linear increases with ir (according to the theory).	87
5.7	Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and different XCS's configurations. The dots show the empirical results and lines plot linear increases with ir (according to the theory).	89
5.8	Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and different XCS's configurations with $\theta_{GA} = n \cdot m \cdot ir$. The points indicate the empirical values of the minimum population size required by XCS. The lines depict the theoretical increase calculated with the previous models, which assumed $\theta_{GA} = 0$	91
5.9	Takeover time in proportionate selection for (a) $m=1$, (b) $m=2$, and (c) $m=3$ and $\rho=\{0.01,0.10,0.20,0.30,0.40,0.50\}$	102
5.10	Takeover time in tournament selection for (a) $m=1$, (b) $m=2$, and (c) $m=3$	103
5.11	Evolution of (a) the proportion of the optimal population and (b) the product of TP rate and TN rate in the 11-bit multiplexer with imbalance ratios ranging from $ir=1$ to $ir=1024$	106
6.1	Histogram of the error of the most over-general classifier in UCS for $ir = \{1, 10, 100\}$	112
6.2	Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and the default configuration with (a) tournament selection and (b) roulette wheel selection The dots shows the empirical results and lines plot linear increases with ir (according to the theory).	117
6.3	Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and different UCS's configurations that do not satisfy the initial model assumptions: (a) using 2-point crossover and (b) using the correct set size deletion scheme. The dots shows the empirical results and lines plot linear increases with ir (according to the theory).	118

6.4	Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and different UCS's configurations with $\theta_{GA} = n \cdot m \cdot ir$. The points indicate the empirical values of the minimum population size required by UCS. The lines depict the theoretical increase calculated with the previous models, which assumed $\theta_{GA} = 0$	119
6.5	Evolution of (a) the proportion of the optimal population and (b) the geometric mean of TP rate and TN rate in the 11-bit multiplexer with $ir=\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$	124
7.1	Example of a domain with two niches (a) and examples of possible representatives of the two niches and over-general classifiers (b) in a two-dimensional problem with continuous attributes	129
7.2	Example of two domains with the same imbalance ratio in the training data set but different niche imbalance ratio.	130
7.3	Example of a domain with oblique boundaries. Several interval-based rules are required to define the class boundary precisely.	132
7.4	Evolution of (a,c) the proportion of the optimal population and (b,d) the geometric mean of TP rate and TN rate of XCS and UCS, respectively, in the 11-bit multiplexer with $ir=\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$	135
7.5	Original domain (a) and domains after applying random over-sampling (b), under-sampling with Tomek links (c), SMOTE (d), cSMOTE (e).	147
7.6	Models created by C4.5, SMO, IBk, XCS and l'UCS with the original and the re-sampled data sets.	149
7.7	Comparison of the performance obtained by (a) C4.5, (b) SMO, (c) IBk, (d) XCS, and (e) UCS with the different re-sampling techniques. Groups of classifiers that are not significantly different at $\alpha = 0.10$ are connected.	152
8.1	Schematic of GFRBS architecture.	163
8.2	Schematic illustration of Fuzzy-UCS. The run cycle depends on whether the system is under exploration (training) or exploitation (test).	167
8.3	Representation of a fuzzy partition for a variable with (a) three and (b) five triangular-shaped membership functions.	168
8.4	Graphical comparison between (a) linguistic and (b) approximate fuzzy rule sets.	178
8.5	(a) Domain of the tao problem and (b) decision boundaries obtained by UCS.	182
8.6	Decision boundaries obtained by linguistic Fuzzy-UCS with weighted average inference and 5 (a), 10 (b), 15 (c) and 20 (d) linguistic terms per variable.	182
8.7	Decision boundaries obtained by linguistic Fuzzy-UCS with action winner inference and (a) 5, (b) 10, (c) 15, (d) and 20 linguistic terms per variable.	183
8.8	Decision boundaries obtained by linguistic Fuzzy-UCS with fittest rules inference and (a) 5, (b) 10, (c) 15, (d) and 20 linguistic terms per variable.	184
8.9	Decision boundaries obtained by approximate Fuzzy-UCS.	184

8.10	Comparison of the training performance of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.	188
8.11	Comparison of the test performance of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.	190
8.12	Evolution of the training and test accuracies with approximate Fuzzy-UCS on the <i>bal</i> problem.	191
8.13	Comparison of the number of rules evolved by all learners against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.	191
8.14	Comparisons of one learner against the others with the Bonferroni-Dunn test at a significance level of 0.1. All the learners are compared to three different control groups: (1) Fuzzy-UCS with weighted average inference, (2) Fuzzy-UCS with action winner inference, and (3) Fuzzy-UCS with fittest rules inference. The learners connected are those that perform equivalently to the control learner. . .	196
8.15	Illustration of the significant differences (at $\alpha = 0.05$) of the test accuracy among the fuzzy-methods and Fuzzy-UCS. An edge $L_1 \xrightarrow{p_{value}} L_2$ indicates that the learner L_1 outperforms the learner L_2 with the corresponding p_{value} . To facilitate the visualization, Fuzzy-AdaBoost and Fuzzy MaxLogitBoost, the two most outperformed algorithms, were not included in the graph.	198
8.16	Examples of part of the models evolved by (a) the GP-based methods, i.e., Fuzzy GP, Fuzzy GAP, and Fuzzy SAP; (b) the boosting learners, i.e., Fuzzy AdaBoost, Fuzzy LogitBoost, and Fuzzy MaxLogitBoost; and (c) Fuzzy-UCS for the two-dimensional tao problem. In the fuzzy learners, we used the following five linguistic terms per variable: {XS, S, M, L, XL}. All fuzzy learners use triangular-shaped membership functions. Moreover, GP-based learners also use trapezoid-shaped membership functions.	199
8.17	Illustration of the significant differences (at $\alpha = 0.05$) of the test accuracy among non-fuzzy methods and Fuzzy-UCS. An edge $L_1 \xrightarrow{p_{value}} L_2$ indicates that the learner L_1 outperforms the learner L_2 with the corresponding p_{value} . To facilitate the visualization, ZeroR and SMO with Gaussian kernels, the two most outperformed algorithms, were not included in the graph.	205
8.18	Examples of part of the models evolved by (a) SMO, (b) C4.5, (c) Part, (d) GAssist, (e) UCS, and (f) Fuzzy-UCS for the two-dimensional tao problem. . . .	206
8.19	Evolution of test accuracies and the population size of Fuzzy-UCS with action winner inference in first 50 000 learning iterations of the 1999 KDD Cup data set.	210
B.1	Comparison of the performance of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.	234

List of Tables

3.1	Example of two-point crossover, in which the two cut points are in the middle of each interval.	48
4.1	Accuracy and fitness of UCSns’s classifiers along the generality-specificity dimension, depicted for the parity problem with $\ell = 4$	58
7.1	Description of the data sets properties. The columns describe the data set identifier (Id.), the original name of the data set (Data set), the number of problem instances (#Ins.), the number of attributes (#At.), the proportion of minority class instances (%Min.), the proportion of majority class instances (%Maj.), and the imbalance ratio (ir).	137
7.2	Comparison of C4.5, SMO, IBk, XCS, and UCS on the 25 real-world problems. Each cells depicts the average value of the product of TP rate and TN rate and the standard deviation. <i>Avg</i> gives the performance average of each method over the 25 data sets. The two last rows show the average rank of each learning algorithm (<i>Rank</i>) and its position in the ranking (<i>Pos</i>).	139
7.3	Comparison of C4.5, SMO, IBk, XCS, and UCS on the 25 real-world problems. For a given problem, the \bullet and \circ symbols indicate that the learning algorithm of the column performed significantly worse/better than another algorithm at 0.95 confidence level (pairwise Wilcoxon signed-ranks test). <i>Score</i> counts the number of times that a method performed worse-better, and <i>Score_{ir>5}</i> does the same but only for the highest imbalanced problems (<i>ir</i> > 5).	140
7.4	TP rate (TPR) i TN rate (TNR) obtained by C4.5, SMO, IBk, XCS and UCS with the original domain and the re-sampled data sets.	150
7.5	Intra-method ranking for original and re-sampled data sets for C4.5, SMO, IBk, XCS, and UCS. Rows <i>1st</i> to <i>5th</i> indicate the number of times that each re-sampling technique was ranked in the correspondent position. The last column shows the average rank and its standard deviation.	154

8.1	Properties of the data sets. The columns describe: the identifier of the data set (Id.), the name of the data set (dataset), the number of instances (#Ins), the total number of features (#Fea), the number of continuous features (#Cnt), the number of nominal features (#No), the number of classes (#C), the proportion of instances of the minority class (%Min), the proportion of instances of the majority class (%Maj), the proportion of instances with missing values (%MI), and the proportion of features with missing values (%MA).	175
8.2	Configurations used to test the sensitivity of Fuzzy-UCS to configuration parameters.	175
8.3	Comparison of the sensitivity of Fuzzy-UCS to configuration parameters. Each cell shows the average rank of each configuration for a given inference scheme. The best ranked method is in bold. The \ominus symbol indicates that the corresponding method significantly degrades the results obtained with the best ranked method.	176
8.4	Summary of Fuzzy UCS results with interval-based, approximate and linguistic representation with 5, 10, 15, and 20 linguistic terms per variable in the tao problem. Columns show the training accuracy and the number of rules for action winner and weighted average inference schemes.	185
8.5	Comparison of the training accuracy of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fittest rules inference (nfit), and approximate Fuzzy-UCS on a set of twenty real-world problems.	187
8.6	Pairwise comparisons of the training accuracy achieved by linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS.	188
8.7	Comparison of the test accuracy of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fittest rules inference (nfit), and approximate Fuzzy-UCS on a set of twenty real-world problems.	189
8.8	Pairwise comparisons of the test accuracy achieved by linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS.	190
8.9	Comparison of the population sizes of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fittest rules inference (nfit), and approximate Fuzzy-UCS on a set of twenty real-world problems.	192
8.10	Pairwise comparisons of the sizes of the rule sets evolved by linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS.	192
8.11	Comparison of the test accuracy of Fuzzy-UCS with weighted average (wavg), action winner (awin), and fittest rules (nfit), to Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost, Fuzzy LogitBoost, and Fuzzy MaxLogitBoost.	195
8.12	Pairwise comparison of the test accuracy of fuzzy learners Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost (ABoost), Fuzzy LogitBoost (LBoost), Fuzzy MaxLogitBoost (MLBoost), and Fuzzy UCS with weighted average inference (wavg), action winner inference (awin), and fittest rules inference (nfit) by means of a Wilcoxon signed-ranks test.	197

8.13	Size of the models evolved by Fuzzy GP, Fuzzy GAP Fuzzy SAP, Fuzzy AdaBoost (ABoost), Fuzzy LogitBoost (LBoost), Fuzzy MaxLogitBoost (MLBoost), and Fuzzy UCS with weighted average inference (wavg), action winner inference (awin), and fittest rules inference (nfit).	201
8.14	Pairwise comparisons of the sizes of the models of Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost (ABoost), Fuzzy LogitBoost (LBoost), Fuzzy MaxLogitBoost (MLBoost), and Fuzzy UCS with weighted average inference (wavg), action winner inference (awin), and fittest rules inference (nfit) by means of a Wilcoxon signed-ranks test.	201
8.15	Comparison of the test accuracy of Fuzzy-UCS with weighted average (wavg), action winner (Awin), and fittest rules inference (nfit) to ZeroR (0R), C4.5, IB5, Part, Naïve Bayes (NB), SMO with polynomial kernels of order 3 (SMO_{p3}), SMO with Gaussian kernels (SMO_{rbf}), and GAssist.	203
8.16	Pairwise comparison of the test accuracy of Fuzzy-UCS with weighted average (wavg), action winner (Awin), and fittest rules inference (nfit) to ZeroR (0R), C4.5, IB5, Part, Naïve Bayes (NB), SMO with polynomial kernels of order 3 (SMO_{p3}), SMO with Gaussian kernels (SMO_{rbf}), and GAssist by means of a Wilcoxon signed-ranks test.	204
8.17	Average sizes of the models build by C4.5, Part, GAssist, UCS and Fuzzy-UCS with weighted average (wavg), action winner (awin), and fittest rules inference (nfit).	207
8.18	Properties of the 1999 KDD Cup intrusion detection data set. The columns describe: the identifier of the data set (Id.), the number of instances (#Inst), the total number of features (#Fea), the number of real features (#Re), the number of nominal features (#No), the number of classes (#Cl), the proportion of instances with missing values (%MisInst), and the dispersion of the data set (Disp) computed as #Fea/#Inst.	209
8.19	Test performance and number of rules evolved by Fuzzy-UCS with weighted average (wavg), action winner (awin), and fittest rules (nfit) in the 1999 Kdd Cup intrusion detection data set at different number of learning iterations.	210
8.20	SWOT analysis of Fuzzy-UCS.	212
A.1	Best action map (first and second columns) and complete action map (all columns) of the parity problem with $\ell = k = 4$	225
A.2	Best action map (first column) and complete action map (all columns) of the decoder problem with $\ell = k = 4$	226
A.3	Best action map (first column) and complete action map (all columns) of position with $\ell=6$	227
A.4	Best action map (first column) and complete action map (all columns) of the multiplexer problem with $\ell = 6$	227

B.1	Comparison of the performance of methods M1 and M2 (second and third column). The fourth column provides the performance difference, and the fifth column supplies the rank of the differences.	231
B.2	Comparison of the performance of methods M1, M2, and M3. For each method and data set, the average rank is supplied in parentheses. The last column provides the rank of each learning algorithm for each data set.	232
B.3	Critical values for the two-tailed Nemenyi test.	234
B.4	Critical values for the two-tailed Bonferroni-Dunn test.	235
C.1	Comparison of the performance, measured as the product of TP rate and TN rate, achieved by C4.5 with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. <i>Avg</i> provides the performance average of each method over the 25 data sets. Rows <i>Rank</i> and <i>Pos</i> show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides <i>Inf/Sup</i> , where <i>Inf</i> is the number of times that the learner has been surpassed by another one, and <i>Sup</i> is the number of times that the method has outperformed another one.	238
C.2	Comparison of the performance, measured as the product of TP rate and TN rate, achieved by SMO with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. <i>Avg</i> provides the performance average of each method over the 25 data sets. Rows <i>Rank</i> and <i>Pos</i> show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides <i>Inf/Sup</i> , where <i>Inf</i> is the number of times that the learner has been surpassed by another one, and <i>Sup</i> is the number of times that the method has outperformed another one.	239
C.3	Comparison of the performance, measured as the product of TP rate and TN rate, achieved by IBk with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. <i>Avg</i> provides the performance average of each method over the 25 datasets. Rows <i>Rank</i> and <i>Pos</i> show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides <i>Inf/Sup</i> , where <i>Inf</i> is the number of times that the learner has been surpassed by another one, and <i>Sup</i> is the number of times that the method has outperformed another one.	240

C.4 Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **XCS** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 data sets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one. 241

C.5 Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **UCS** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 data sets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one. 242

D.1 Comparison of the test accuracy obtained with the three types of inference and the three configurations which vary $P_{\#}$. *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology. 246

D.2 Comparison of the model sizes obtained with the three types of inference and the three configurations which vary $P_{\#}$. *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology. 246

D.3 Comparison of the test accuracy obtained with the three types of inference and the three configurations which vary the fitness pressure ν . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology. 248

D.4 Comparison of the model sizes obtained with the three types of inference and the three configurations which vary the fitness pressure ν . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology. 248

D.5	Comparison of the test accuracy obtained with the three types of inference and the five configurations varying θ_{GA} , θ_{del} , and θ_{sub} . <i>Rank</i> gives the average rank of each configuration for each one of the three inference schemes. <i>Pos</i> shows the absolute position in the ranking. <i>Frd</i> reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.	250
D.6	Comparison of the model sizes obtained with the three types of inference and the five configurations varying θ_{GA} , θ_{del} , and θ_{sub} . <i>Rank</i> gives the average rank of each configuration for each one of the three inference schemes. <i>Pos</i> shows the absolute position in the ranking. <i>Frd</i> reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.	251
D.7	Pairwise comparison of the test accuracy of Fuzzy-UCS obtained with the three types of inference and the five configurations varying θ_{GA} , θ_{del} , and θ_{sub} by means of a Wilcoxon signed-ranks test.	251
D.8	Comparison of the test accuracy obtained with the three types of inference and the two configurations which vary the deletion pressure δ . <i>Rank</i> gives the average rank of each configuration for each one of the three inference schemes. <i>Pos</i> shows the absolute position in the ranking. <i>PW</i> reports the p-value obtained with the pairwise Wilcoxon signed-ranks test performed for each inference methodology.	252
D.9	Comparison of the model sizes obtained with the three types of inference and the three configurations which vary the deletion pressure δ . <i>Rank</i> gives the average rank of each configuration for each one of the three inference schemes. <i>Pos</i> shows the absolute position in the ranking. <i>PW</i> reports the p-value obtained with the pairwise Wilcoxon signed-ranks test performed for each inference methodology.	252

List of Algorithms

2.2.1 Pseudo code of a simple GA.	21
7.2.1 Pseudo code for the <i>online adaptation algorithm</i> in XCS.	133
7.2.2 Pseudo code for the on-line adaptation of β	133
7.2.3 Pseudo code for the <i>online adaptation algorithm</i> in UCS.	134
7.4.1 Pseudo code for the Tomek Links algorithm.	142
7.4.2 Pseudo code for the SMOTE algorithm.	144
7.4.3 Pseudo code for the cSMOTE algorithm.	145

Chapter 1

Introduction

In the last few decades, there has been increasing interest in *machine learning* (Mitchell, 1997, 2006; Nilson, 2005; Bishop, 2007), a field in artificial intelligence (Feigenbaum and Feldman, 1995; Brooks, 1990; Russell and Norvig, 2002; McCharthy, 2007) that is concerned with the development of machines that can learn from the experience. The appeal of machine learning is based on the idea of having computers that teach themselves to solve new challenging problems instead of programming them with a deterministic behavior. This approach appears to be especially attractive in applications (1) that are too complex for human experts to manually design and implement the solver, or (2) that require the software to improve or refine itself continuously. Therefore, the ultimate aim of machine learning is to solve problems that are too complex for human beings to solve or to give instructions on how to solve them.

Machine learning does not only aim at solving engineering problems, but it is also closely related to different fields such as logic and philosophy, theoretical computer science, statistics, biology, experimental psychology, cognitive science, and communication theory among others (Buchanan, 2005). For example, several machine learning techniques derive from works of psychologists that try to understand animal and human behavior through computational modeling. Similarly, machine learning research and biological learning are related, and research in each area benefits from the other one resulting in a fruitful crossbreeding among the techniques studied in the two areas. Thence, machine learning holds promise not only in empowering computers so that they can solve new challenging problems, but also in providing a framework to study artificial intelligence.

One of the most appealing machine learning techniques is *learning classifier systems* (LCSs), whose theoretical foundation was early outlined by Holland (1962). With the purpose of creating true artificial intelligence itself, Holland (1971, 1976) envisioned LCSs as cognitive systems that received perceptions from their environment and, in response to these perceptions, performed actions in the real world to achieve certain goals; besides, the policy of these programs *evolved* with their interaction with the changing environment, refining the policies with the aim of achieving a maximum reward—which was aligned to the goals of the system—while adapting to the changes found in the environment. Since the first definition by Holland, there has been an augmenting research on LCSs, which has led to design new systems and to solve increasingly complex and challenging problems. Currently, LCSs are competent machine learning techniques that are able to solve hard problems that range in different disciplines. Nevertheless, some chal-

lenges, most of the times shared by the machine learning community, still need to be addressed to scalably and efficiently solve new complex real-world problems.

This thesis is concerned about advancing in the research on LCSs to gain a better understanding of their behavior and to improve them to deal with current problems in science, engineering, and industry. Further, we elaborate the framework of this thesis in more detail, providing some historical remarks on the LCSs's research. Taking the original definition of Holland, we follow the contributions that have led to the current LCSs' architectures. Then, we identify two important challenges in LCSs and machine learning systems alike, which are later articulated in the objectives of this work. Finally, we provide the road map of the entire document.

1.1 Framework: From Holland's Definition to Current LCSs

Holland (1971, 1976) proposed the original idea of LCSs as cognitive systems that infer environmental patterns from experience and associate appropriate response sequences with them. The initial schemas of learning classifier systems already pointed out three key aspects, which have been preserved up to the most recent implementations: (1) a knowledge representation based on *classifiers*—usually implemented as production rules—that enables the system to map sensorial states with actions, (2) an *apportionment of credit algorithm* which shares the credit obtained by the machine among classifiers, (3) an algorithm to *evolve* the knowledge base—typically, a *genetic algorithm* (GA) (Holland, 1975). Since the first definition of LCS, more than 30 years ago, several systems have been designed following Holland's initial definition; also, slightly different angles of the same problem resulted in different types of LCSs, shaping the LCS branches that currently exist. As proceeds, we review some of the most important aspects of these 30 years of history.

The first successful implementation of LCSs was the cognitive system one (CS-1) by Holland and Reitman (1978), which was designed to imitate animal behavior. The goal of the system was to satisfy its needs by means of obtaining a finite number of *resources* that were maintained in different *reservoirs*. CS-1 acted in a stimulus-response way; given each sensorial input, the machine performed an action to the environment, which in turn responded with a reward. The system implemented the three aforementioned key aspects in the following way. CS-1 used simple string rules to code the internal policy. An *epochal apportionment of credit system* was employed to share the resources among rules. This algorithm tracked the utility of rules during an epoch—that is, a certain time in which payoff events were received—and, at the end of the epoch, distributed the payoff according to the value of each rule. Learning was accomplished through a GA. CS-1 was faced to two maze-running tasks in which different units of food and water were placed around the maze. The experimental results showed that CS-1 could evolve an accurate policy to reach the system goals, highlighting that LCSs held promise for machine learning.

Subsequently to the development of CS-1, several authors continued on the design and implementation of new LCSs based on Holland's original ideas. Booker (1982) adopted an LCS with an architecture inherited from CS-1 to study the connections between LCSs and cognitive science. The system was tested on environments where the classifier wandered in a feature space with the aim of avoiding aversive stimuli (poison) and reaching attractive stimuli (food). Contemporaneous with this work, Wilson (1981, 1985a) proposed an LCS for the sensory-motor

coordination on movable video camera, addressed as the EYE-EYE system. Later, Wilson (1985b) simplified the LCSs architecture and applied the system to the modeling of an artificial animal—i.e., the *animat* problem. Thereafter, to investigate further on the system, Wilson (1987) simplified the learning task by designing a non-sequential problem that provided immediate reward at each learning iteration. Approximately together with the development of these works, Goldberg applied LCSs to the learning control of a simulated gas pipeline (Goldberg, 1983, 1985a,b, 1987a,b).

In parallel to Holland's work on CS-1 and its derivations, Smith (1980) took a different approach and developed a new type of LCSs. Regarding learning as an *adaptive search procedure*, Smith raised the focus of operation one notch and developed the learning system 1 (LS-1) (Smith, 1980, 1983, 1984), a system that used a GA to evolve rule sets instead of evolving individual classifiers or rules as in Holland's approach. Therefore, genetic manipulation worked at rule set level instead of at rule level. This permitted sidestepping the apportionment of credit algorithm. That is, as the rules belonged to a set, the need to compute the individual contribution of each rule to the whole model disappeared. This LCS model was furthered in new implementations (Jong et al., 1993; Janikow, 1993). Therefore, CS-1 and LS-1 defined two different ways to perform learning by means of GAs. After some years of development, the algorithms resulting from both approaches were distinguished and addressed with different names: Holland's LCSs were referred to as Michigan-style LCSs, whilst Smith's LCSs were addressed as Pittsburgh-style LCSs. In this thesis, we focus our research on Michigan-style LCS.

Despite these promising designs and first applications on attractive problems, LCSs did not reach a general acceptance in the machine learning community, probably due to their lack of mathematical foundation and their restricted applications. Consequently, after a period of strong research in the late 1970s and early 1980s, the late 1980s were known as the LCSs winter, in which the problems detected in the first LCSs seemed to cloud the whole field. At the end of the 1980s, Wilson and Goldberg (1989) published a critical review of LCSs, identifying the critical factors and problems that hindered the success of the LCSs of that time. These problems were associated with (1) the difficulties of distributing credit among the rules, (2) the inadequacy of the decision-making process and the tendency to produce over-general rules, and (3) the limits of the classifier syntax. Besides, the authors also pointed out the need for theory, such as population size models, to gain a better understanding of how these systems worked.

Some years after the critical review, Wilson (1994) first presented the *zeroth-level classifier system* (ZCS) and, one year after, Wilson (1995) proposed the *extended classifier system* (XCS), heralding the second spring and summer of the LCSs field. XCS came to give answer to most of the key problems that were identified in previous LCSs. XCS introduced a new credit apportionment algorithm—which was adapted from a well-known reinforcement learning technique (Sutton and Barto, 1998), Q-learning (Watkins, 1989)—to solve the difficulties in distributing credit. The tendency of producing a large number of over-general classifiers was corrected by (1) basing the classifier fitness on the accuracy of the prediction instead of the prediction itself and (2) using appropriate *niching techniques* and *fitness-sharing schemes*. Besides, the system architecture was simplified with respect to the initial LCSs' architecture. Already in the original paper, XCS was shown to be able to evolve accurate models for single step tasks—in particular, the multiplexer problem (Wilson, 1987)—and to learn optimal policies in maze-running environments, problems that previously eluded solution.

The first publication of XCS promoted an increasing amount of research in the LCSs area, resulting in the so-called LCSs renaissance. Ad hoc with the implementation of XCS, there have been crucial advances in (1) enhancements of the learning architecture and design of new operators, (2) theoretical analyses for design, and (3) applications in important domains. With respect to the learning architecture, the original scheme of XCS was first refined by Wilson (1998) and later by Kovacs (1999), resulting in the standard XCS scheme currently used. Also, there have been notorious works on inclusion of new knowledge representations (Wilson, 2000, 2001, 2008; Lanzi, 1999a; Lanzi and Perrucci, 1999; Bull and O’Hara, 2002; Butz et al., 2008), analyses and improvements of the credit apportionment algorithm (Butz et al., 2005a; Drugowitsch and Barry, 2008), and enhancements of some genetic operators (Butz et al., 2005b,c). Second, there have been several theoretical analyses that enabled a better understanding of the system (Butz and Pelikan, 2001; Butz et al., 2004b, 2005a, 2007; Drugowitsch and Barry, 2008; Drugowitsch, 2008). Finally, XCS and similar systems have been applied to important applications such as data mining (Bull, 2004; Bull et al., 2008), function approximation (Wilson, 2002b; Butz et al., 2008), reinforcement learning (Lanzi, 1999b, 2002; Lanzi et al., 2005; Butz et al., 2005a), and clustering (Tamee et al., 2006, 2007), demonstrating the competitiveness of LCSs, and XCS in particular, with respect to other machine learning techniques from other paradigms such as decision trees (Quinlan, 1995) or neural networks (Widrow and Lehr, 1990). Besides, Michigan-style LCSs provide a competitive advantage with respect to other machine learning techniques: they evolve the knowledge online from a stream of examples. Therefore, the data can be made available in streams, which is very common in current industrial applications where large volumes of data are generated online (Aggarwal, 2007; Gama and Gaber, 2007).

Along with the application of XCS to important domains, there have been some proposals in which the learning architecture of XCS has been modified for specific types of tasks (Wilson, 2002b; Bernadó-Mansilla and Garrell, 2003; Bull, 2005; Tamee et al., 2006). In the particular case of data classification tasks, Butz et al. (2003) detected that XCS produced a deceptive pressure toward the optimal solution in some specific problems. In order to overcome this problem, Bernadó-Mansilla and Garrell (2003) proposed the *supervised classifier system* (UCS), a system that inherits the main components of XCS, but specializes them for supervised learning—specifically, for classification tasks. The advantages of the new architecture with respect to UCS were analyzed on a set of boundedly difficult problems—problems in which the complexity along different dimensions can be controlled—, illustrating that the system could overcome the detected problems and solve complex applications more efficiently than XCS. Nonetheless, UCS is still young and, as pointed out by Bernadó-Mansilla and Garrell (2003), there are some open issues that still need to be addressed to enhance the system and gain a better comprehension of the implications of the changes introduced to the original XCS.

In summary, starting from Holland’s idea of creating true artificial intelligence, during the last decade, research on LCSs has been enjoying a renaissance, which has been mainly promoted by the creation of XCS. Currently, LCSs have reached a mature state and are ready to face new challenging problems in machine learning. Furthermore, Michigan-style LCSs have two main assets that distinguish them from machine learning techniques alike:

1. They have a flexible knowledge representation that can be easily adapted to deal with new types of data.
2. They build the model online, which is crucial to succeed in problems with large volumes

of data or where the data is made available in data streams.

For this reason, the present work focuses on LCSs as promising alternatives for machine learning. Despite the recent improvements and applications proposed in the field of LCSs, there are still important challenges—which are not particular to LCSs, but shared by the machine learning community in general—that need to be addressed to scalably and efficiently solve real-world problems. In the following section, we identify the two key challenges in the machine learning and LCSs community from which we define the objectives of this thesis.

1.2 Two Critical Challenges in LCSs and Machine Learning

Research on machine learning has resulted in the design of several learning techniques, such as LCSs, that can extract accurate models from the experience. Due to the maturity of the area, the machine learning community has started to address new important challenges that appear when applying learning techniques to real-world problems. Among the different research lines, the following two key challenges have received especial attention:

1. Learning from domains that contain rare classes.
2. Building more understandable models and bringing reasoning mechanisms closer to human ones.

A more detailed discussion of why these two items represent a critical challenge not only for LCSs but for machine learning in general is provided as follows.

Learning from domains that contain rare classes. The advances in machine learning have led to the application of learning algorithms to new complex real-world problems—for which humans cannot provide an accurate solution—with the aim of extracting novel, interesting, and useful knowledge. It has been identified that, in these types of problems, the key knowledge usually is hidden in examples that are rare in nature (Chan and Stolfo, 1998; den Bosch et al., 1997; Grzymala-Busse et al., 2000; Kubat et al., 1998). In fact, for this reason it is too complex for human beings to identify this key, hidden knowledge. Empirical studies have shown that traditional machine learning techniques may not be able to extract critical information from these rarities. Therefore, a new field has emerged with the aim of creating new approaches to enhance the extraction of the key knowledge from rare classes. The problem of modeling rare classes has taken several names such as the problem of *mining rarities* (Weiss, 2004), the problem with the *small disjuncts*, (Holte et al., 1989) or the *class-imbalance* problem (Japkowicz and Stephen, 2002). In the three of them, the goal is the same: model patterns or examples that occur infrequently accurately.

While this critical problem has been widely studied in the context of traditional machine learning techniques—which learn from collections of static data—little research has been conducted on online learners and, specifically, on LCSs. Two main reasons explain this lack of analyses in LCS. First, as mentioned in the previous section, the first successful LCS’s architecture was designed in 1995, and most of the research conducted during the last decade has been centered on the analysis and improvement of this architecture. This has resulted in mature LCSs

that are now ready to face new challenges. On the other hand, the first studies of the *small disjuncts* problem started in the late 1980s (for example, see (Holte et al., 1989)). Therefore, the problem of rare classes has received much more attention in traditional learning techniques than in LCSs. Second, learning from rare classes poses more complex challenges to LCSs since they have an online architecture that learns from a stream of examples. That is, the online system receives a stream of examples, and it has to learn from the upcoming rarities on the fly. Besides, due to this online architecture, techniques developed for traditional machine learning techniques cannot be applied to LCSs. Note that the study and improvement of LCSs to extract accurate models from rarities that come infrequently in a stream of examples appears to be a crucial task to address, accurately and efficiently, the new problems that are more often presented to machine learning techniques.

Building more understandable models and bringing reasoning mechanisms closer to human ones. Besides extracting accurate models from rare classes, a second important challenge in machine learning is to build learning techniques that represent the knowledge and apply reasoning mechanisms that are similar to the human ones. This point is especially important in classification tasks where human experts may need explanations about the decisions taken by the systems. For example, in medical domains, human experts are sometimes more interested in the explanation that yields a prediction than in the prediction itself (Robnik-Sikonja et al., 2003). As proceeds, we discuss why LCSs may evolve poorly interpretable models and present fuzzy logic as a competent approach to create highly legible models.

Michigan-style LCSs evolve models that consist of classifiers—typically rules—which can be individually interpreted by human experts. Nevertheless, it has been detected that Michigan-style LCSs evolve a large number of rules when dealing with problems that have continuous-valued attributes (Bernadó-Mansilla and Ho, 2005; Bacardit and Butz, 2004; Wilson, 2002a; Dixon et al., 2004; Fu et al., 2001), which can be found usually in real-world problems. Besides, the reasoning mechanisms of LCSs may not be natural for human experts. Until recently, few alternatives of new reasoning mechanisms, as well as the first pieces of theory that explain how they work, have been developed for some particular LCSs (Brown et al., 2007). Despite these first promising results, more research needs to be conducted to approach reasoning mechanisms to human reasoning.

Contemporaneous with the recent advances on LCSs, there has been a strong research on fuzzy systems, that is, systems that use fuzzy logic (Zadeh, 1965, 1973) to create highly legible models from environments with uncertainty and imprecision. Essentially, the fuzzy set theory provides a robust reasoning mechanism that approaches human reasoning. Therefore, the combination of fuzzy systems with LCSs appears as an appealing alternative to improve their explicative capabilities. As a consequence, the first attempts to mix both disciplines have been taken (Valenzuela-Rendón, 1991; Nomura et al., 1998; Parodi and Bonelli, 1993; Furuhashi et al., 1994; Velasco, 1998; Ishibuchi et al., 1999b; Casillas et al., 2007); but, so far, no competitive Michigan-style LCSs that creates fuzzy classification models online from streams of examples and uses fuzzy reasoning mechanisms have been designed.

Therefore, the scope of this thesis is to address these two challenges in the context of LCSs. Specifically, we consider XCS and UCS as starting point. We select XCS since it is, by far, the most influential Michigan-style LCS, representing the state of the art in the LCS field. Besides, we incorporate UCS since it was specifically designed for supervised learning, and this work

is especially concerned with classification problems. With these two challenges in mind, the following section explicitly articulates the objectives of this thesis.

1.3 Thesis Objectives

The general goal of the present work is to address the two aforementioned key challenges with LCSs, particularly focusing on XCS and UCS. As stated in the previous sections, UCS is a young promising system derived from XCS that, although having shown to be competitive with respect to other machine learning techniques, still has some open issues that have to be addressed before applying it to new complex problems. Thus, before approaching the two particular challenges defined in the previous section, we first study the UCS classifier system in detail and update its architecture. Furthermore, to understand its differences with XCS, we empirically compare both systems on a set of boundedly difficult problems. Thereafter, we take XCS and the revised version of UCS as a departure point to analyze and improve LCSs to model rare classes accurately. For this reason, we propose to follow an analytical approach to study the LCSs' behavior on problems with rare classes, improve the systems, and apply them to real-world problems with class imbalances. Furthermore, we also propose to include fuzzy logic in an LCS architecture to bring the reasoning mechanisms closer to the human's ones. Specifically, this leads to the definition of the following four objectives:

1. Revise and update UCS and compare it with XCS.
2. Analyze and improve LCSs for mining rarities.
3. Apply LCSs for extracting models from real-world classification problems with rarities.
4. Design and implement an LCS with fuzzy logic reasoning for supervised learning.

As follows, each one of the four objectives is elaborated in detail.

Revise and update UCS and compare it with XCS. Whereas XCS has received an increasing amount of attention during the last decade, resulting in many improvements in the architecture, UCS is still a young system which has received no further modifications since its initial design. Nevertheless, [Bernadó-Mansilla and Garrell \(2003\)](#) detected some critical aspects that needed to be investigated in more detail. The most important one was the lack of *fitness sharing* in the credit apportionment algorithm. That is, differently from almost all the current Michigan-style LCSs, the rules in UCS are evaluated independent of the remaining rules in the population. [Bernadó-Mansilla and Garrell \(2003\)](#) took this approach since the benefits of a credit apportionment algorithm that shared the fitness among individuals were not clearly identified, and further analysis on sharing algorithms was pointed out as an important future work line. Therefore, the first objective of the thesis is to design a fitness-sharing scheme similar to those proposed by GAs and XCS, introduce it to UCS, and analyze the advantages that the new credit apportionment algorithm provides to UCS. We also aim at analyzing the differences between UCS and XCS on supervised learning problems.

Analyze and improve LCSs for mining rarities. In this second objective, we address the challenge of extracting accurate models from domains that contain rare classes with LCSs—in

particular, with XCS and UCS. Specifically, the goal is to study the intrinsic capacities of both LCSs to learn from rare classes, identifying critical factors for the success of the systems. For this purpose, we propose to use *design decomposition* (Goldberg, 2002) to separate the problem of learning from domains with rarities in several critical elements and to derive *facetwise models* for each element. The integration of these models would permit us to draw the domain of applicability of both LCSs and to extract critical bounds on their behavior on imbalanced domains. Moreover, we aim at extracting lessons from the analysis that help improve the systems and enable them to extract accurate models from problems with rare classes that currently elude solution.

Apply LCSs for extracting models from real-world classification problems with rarities. After studying and improving LCSs for mining rarities, we aim at applying both LCSs to a set of real-world classification problems with rare classes. To analyze their performance, we propose to compare the accuracy of the models evolved by the two LCSs with the accuracy of the models created by several highly influential machine learning techniques. As we seek to extract highly accurate models, we also propose to include and analyze the impact of some of the most known *re-sampling techniques* (Japkowicz and Stephen, 2002; Chawla et al., 2002; Batista et al., 2004), that is, pre-processing methods that try to remove rare classes from the original data sets.

Design and implement an LCS with fuzzy logic reasoning for supervised learning. In the last objective of this thesis, we address the second aforementioned challenge and take an inventiveness approach to mix the ideas of the fuzzy systems and the LCSs fields. That is, we purpose to create a hybrid system that mixes the best characteristics of LCSs—as accurate online classifier’s evaluators—, GAs—as robust search mechanisms—, and fuzzy logic—as a human-like approach to represent the knowledge and to reason for decision making.

Each one of these objectives gets, at least, a chapter of the present thesis. The overall structure of the document is provided in the following section.

1.4 Road Map

This thesis is organized, in addition to the present chapter, in eight chapters whose content is introduced in what follows.

Chapter 2 starts with a concise introduction to machine learning and to the types of problems that we can find in this discipline, which is followed by an introduction to evolutionary computation. This gives way to the presentation of the current LCS families and to what we currently understand as GBML. That is, while in the present chapter we have provided a brief history of LCSs, in chapter 2 we review the current branches, draw a big picture of different learning methodologies that use evolutionary algorithms, and place LCSs in this picture.

Chapter 3 provides a detailed explanation of both XCS and UCS, which can be used as implementation guidelines. Thence, chapters 2 and 3 give all the background material that is necessary to start with the contributions of this work. Thus, each of the subsequent chapters focuses on one of the objectives of the thesis.

Chapter 4 reviews UCS and updates the system with a new fitness-sharing scheme. Then, UCS with fitness sharing is empirically compared with the original UCS on a set of four artificial problems that have different complexities that are usually present in real-world problems. Therefore, the comparison enables us to highlight the benefits of having a credit apportionment algorithm that shares fitness in UCS. Later, we also introduce XCS in the comparison with the aim of emphasizing the advantages that the modifications introduced by UCS supply in problems with certain characteristics.

Chapter 5 starts with the study of how LCSs can learn from domains that contain rare classes. Although the chapter is focused on XCS, we first take a general approach and intuitively analyze the problems that may arise in a general LCS architecture when learning from rare classes. With this intuition in mind, we decompose the problem and identify five elements that need to be satisfied by any LCS and systems alike to learn, efficiently and scalably, from domains with rare classes. Thence, we create a general framework from learning from class-imbalanced problems without getting tied to any particular LCS architecture. Subsequently, we look at the particular architecture of XCS and derive facetwise models that explain each one of the different elements. During the analysis of each facet, we assume that the remaining facets behave in an ideal manner. The integration of all these models enables us to draw the domain of applicability of XCS, detecting the sweet spot where XCS can efficiently extract accurate models from instances that come infrequently. At the end of the chapter, we show that the lessons extracted from the analysis enable us to solve problems with infrequent classes that previously eluded solution.

Chapter 6 carries over the facetwise analysis to UCS. We start reviewing the general framework proposed in the previous chapter and analyze the components that UCS changes with respect to XCS. We derive new models for these components and plug them into the initial framework, thus, adapting the domain of applicability to UCS. Lastly, we show that UCS has similar learning capabilities to XCS in imbalanced domains.

Chapter 7 moves the theory developed in the previous two chapters to real-world problems, in which the characteristics of the domains are not known. We design two heuristic procedures that gather information from the population evolution and self-adapt XCS and UCS according to the lessons learned from the theory. Then, we test both LCSs on a collection of real-world problems that contain rare classes. To evaluate the performance of both systems, we compare them to three of the most influential machine learning techniques. Later, we introduce pre-processing techniques that try to remove the rare classes by changing the distribution of the training examples and analyze the impact of applying these techniques in combination with each one of the five learners.

Chapter 8 presents Fuzzy-UCS, a hybrid between LCSs, GAs, and fuzzy systems. Fuzzy-UCS is inspired by UCS, but includes a fuzzy representation and usual reasoning mechanisms in fuzzy systems, which approach human reasoning. This chapter performs a large experimentation to show the excellence of Fuzzy-UCS with respect to other machine learning techniques in data classification tasks. We compare Fuzzy-UCS with several top-notch fuzzy learners and show that Fuzzy-UCS outperforms them all. Moreover, we also compare the system to some of the most influential non-fuzzy learners. Fuzzy-UCS appears to be, at least, as accurate as the best performer among the tested learners. Besides, we show that the models evolved by Fuzzy-UCS

are clearly more interpretable than those created by UCS. Finally, we finish the chapter by demonstrating the value of Fuzzy-UCS to mine large volumes of data. We use Fuzzy-UCS to evolve classification models from the data provided in the KDD'99 cup intrusion detection data set (Hettich and Bay, 1999). The data set consists of 494 022 instances, 21 classes, and 41 attributes. It is worth noting that most of the learners used in the comparison are not able to learn from such a large data set. The online architecture of LCSs enables Fuzzy-UCS to deal with this large amount of data.

Chapter 9 finishes with the contributions of this thesis by summarizing, providing key conclusions, reviewing the main lessons extracted from this work, and gathering future work lines.

The material presented in the nine chapters is complemented with four appendices. **Appendix A** describes all the boundedly difficult problems used along the experiments of the thesis. **Appendix B** gives details about the statistic tests employed in different chapters to compare results. **Appendix C** supplies the detailed tables of results of the comparison of several machine learning techniques with LCSs on a collection of real-world problems with rare classes performed in chapter 7. Finally, **Appendix D** provides an analysis of the sensitivity of Fuzzy-UCS to its configuration parameters.

Chapter 2

Machine Learning with Learning Classifier Systems

This chapter provides a brief introduction to *learning classifier systems* (LCSs) as one of the most appealing alternatives for *machine learning* (ML). The chapter starts with a concise definition of machine learning, and then, presents a task-oriented taxonomy of ML techniques which divides ML methods—depending on the type of problems that they can solve—in supervised learning, unsupervised learning, and reinforcement learning techniques. Next, *evolutionary computation*—a field of study devoted to the design and implementation of problem solvers inspired by principles of natural evolution and genetics—is briefly introduced. This introduction is followed by a more detailed explanation of *genetic algorithms*, one of the most promising techniques in evolutionary computation, since they guide the discovery process in LCSs. Finally, the current branches or families of algorithms that use GAs for machine learning—usually addressed as *genetic-based machine learning* systems (GBML)—are presented, identifying both Pittsburgh- and Michigan-style LCSs in this taxonomy. For each one of these families, a picture of their process organization is provided, and the main differences among them are discussed.

2.1 Machine Learning

Machine learning is concerned with the design of computer programs that are able to learn from the experience and with the definition of the fundamental laws that govern all learning processes (Mitchell, 1997, 2006; Nilson, 2005; Bishop, 2007). This definition covers a large variety of learning tasks such as (1) the design of goal-oriented agents that learn behavioral policies from their interaction with the real world, (2) the extraction of frequent, interesting patterns from large volumes of plain data generated, for example, from industrial processes, and (3) the modeling of specific domains from examples. As a unified vision of all these learning tasks, Mitchell (2006) considers that a machine *learns* with respect to a certain task T , a performance metric P , and a type of experience E , if the system reliably improves its performance P at task T by following the experience E . Therefore, any application that falls under this definition can be considered as a machine learning method. With this broad definition in mind, this section discusses the relationship of ML with other scientific fields as well as the necessity of having computers that use their own experience to program themselves.

ML is not an isolated discipline, but it is closely related to other fields such as *computer science*, *statistics*, and *psychology* and *neuroscience*. On the one hand, computer science and ML share a common objective; that is, they are concerned about building machines that can solve problems. However, the main difference is that computer science is mainly focused on building deterministic programs, while ML aims at creating programs that learn by themselves. On the other hand, statistics and ML share the common goal of inferring patterns, behaviors, or conclusions from data. Nevertheless, the key difference between both resides in the fact that ML involves additional questions, such as algorithmic scalability, that aim at the creation of machines that can efficiently, accurately, and scalably deal with complex real-world problems. In addition, ML is closely related to the study of human and animal learning in fields such as psychology and neuroscience. For example, several ML techniques derive from works of psychologists that try to understand animal and human behavior through computational modeling. Similarly, ML and biological learning are related, and research in each area benefits the other one, resulting in fruitful crossbreeding of ideas and techniques.

Of course, the idea of having computers teaching themselves is not easy to implement in practice. Therefore, the question that arises is why we should spend efforts in building machines that learn to solve new complex problems instead of relying on humans to code hard-wired solutions for these problems. The need for further research on ML can be explained with two main reasons. From the pure learning point of view, ML can help understand animal and human learning processes, thus providing key insights to psychologist and neuroscientists. From a pure engineering point of view, ML has already provided, and is expected to supply, efficient algorithms to solve new challenging, complex engineering problems whose solution is not known. More specifically, the most important reasons that may lead to the application of ML to solve engineering problems are enumerated as follows.

1. Difficulty of human experts to describe the problem and manually design an algorithm to solve it.
2. Necessity of programs that continuously adapt to changing environments.
3. Necessity of processing overwhelming volumes of data with hidden concepts.

Below, each item is elaborated in more detail.

The application of ML is mandatory when the problem is too complex to manually design and code an algorithm to address it properly. These types of complex problems abound in engineering, having some specific examples in speech recognition (Karat et al., 2003), or computer vision (Jahne et al., 1999). For instance, recognizing faces is a simple task for humans, but manually programming a system to perform this task is too complex. Nevertheless, collecting some examples and training a computer vision program that recognizes these objects—with a certain accuracy—is a more straightforward manner of facing the problem.

Another reason that makes the use of ML necessary is in changing environments. Independent of whether we can provide an initial solution to the problem, the system may need to adapt to changing situations. For example, in speech recognition systems, the program can be provided with an initial speaker-independent voice-recognition system, plus a learning system that adapts to the characteristics of each particular person. Other examples where adaptation is necessary can be found in robot control.

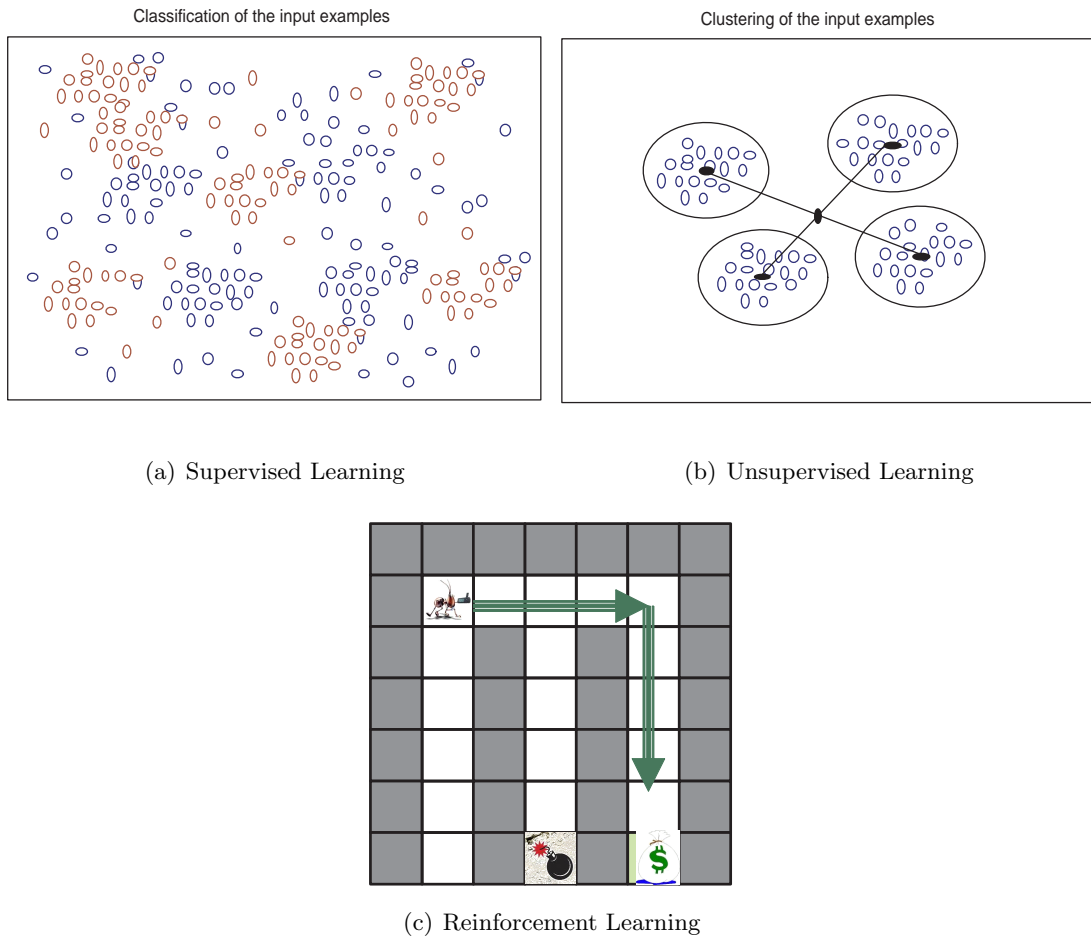


Figure 2.1: Examples of (a) supervised, (b) unsupervised, and (c) reinforcement learning.

The last reason that may lead us to the application of ML is when overwhelming volumes of data need to be processed to extract novel, interesting, and useful knowledge from patterns hidden in these data. Actually, this is a definition of *data mining* (Frawley et al., 1992). In this case, ML can be applied to build programs that use heuristics to extract potentially interesting and novel patterns from the data.

Therefore, ML gathers a large variety of techniques, and several classification criteria can be used to group them in different families. As follows, we present a classic taxonomy of ML techniques that is based on the task to perform. New trends in ML may incorporate new groups or subgroups to this taxonomy; nonetheless, the provided taxonomy gives the three fundamental types of learning: *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

2.1.1 Supervised learning

Supervised learning is the process of extracting a *function* or *model* that maps the relation between a set of descriptive input attributes and one or several output attributes. Depending on the type of output attributes, supervised learning can be further classified as *data classification* or *data regression*. That is, for categorical output attributes (which represent the classes of the examples), the task is addressed as *data classification*; in this case, the goal is to find a model that predicts the class of new instances. Otherwise, for continuous output attributes, the problem is referred to as *data regression*; thence, the goal is to find a function that predicts the output value of new instances. Thus, in general, a supervised learner has to build a function or model that predicts the output value for any valid input object by means of generalizing from the known data.

As follows, we present an application example of supervised learning. Let us imagine that we aim at designing a machine capable of distinguishing poisonous mushrooms from edible mushrooms. Suppose we have been provided with 1 000 examples of poisonous mushrooms and 1 000 examples of edible mushrooms, which form our training data set. Moreover, let us assume that the mushrooms are represented by two characteristics or attributes: the length of the stem and the diameter of the mushroom. These characteristics define the inputs of the classification problem. The problem has a single output attribute that can take two values, which represent whether the mushroom is poisonous or edible. Figure 2.1(a) shows how the different examples are distributed in the feature space (each class is depicted with a different color). By only considering the 2 000 known examples, the given ML technique has to be able to generalize and extract a *classification model*, which will be used to predict the class of new unlabeled examples.

2.1.2 Unsupervised learning

In *unsupervised learning*, the machine receives a set of examples that consist of input attributes, but that have no associated output attributes. Then, the goal of unsupervised machine learning is to build representations from the input that identify novel, interesting knowledge. The resulting representations can be used for decision making, predicting future inputs, grouping similar inputs, or creating prototypes that are fed to other machine learning techniques among others. Two cornerstones of unsupervised learning are *clustering* and *dimensionality reduction*.

Figure 2.1(b) presents an example of clustering. Note that, differently from figure 2.1(a), the training examples have no associated output (all points are depicted with the same color). Without any further information about the data rather than the input attributes, unsupervised learners would group the examples in different clusters according to some proximity criterion (in the example of the figure, the center of each cluster is depicted with a black dot). This type of learning is very common when hidden patterns are searched on large volumes of data. A typical example can be found in the characterization of customer habits from information about their purchases.

2.1.3 Reinforcement Learning

Reinforcement learning lies between supervised and unsupervised learning. In this type of task, an agent interacts with an environment in the following manner: the machine receives

perceptions from the environment—which provide total or partial information about its state—and performs actions to this environment with the aim of achieving a particular, or several, goals. The machine eventually receives positive or negative rewards as consequence of its actions. Therefore, reinforcement learning aims at learning a behavioral policy to maximize a notion of long-term reward—that is, to maximize not the immediate but the total reward received from the environment.

Figure 2.1(c) shows an example of reinforcement learning problem in which an ant or agent aims at reaching its goal—that is, to find the food—as fast as possible without falling in any trap. The agent may receive negative rewards from the interaction with the environment—for example, if it finds a trap—and positive rewards if it reaches a goal. Thence, the aim of the agent is to learn a behavioral policy that maps the best action for each possible sensorial input. Provided that there may be a large number of possible sensorial states and actions for each state, generalization over these sensorial states has become a key aspect in reinforcement learning to scalably solve real-world problems.

In this section we presented a classic taxonomy that identifies three types of learning. Different machine learning techniques have been developed to perform some or several of the aforementioned tasks. One of the most promising approaches to face the problems that range in the three aforementioned families is *learning classifier systems* (Holland, 1971, 1976; Holland and Reitman, 1978). Originally implemented by Holland and Reitman (1978) with the aim of simulating the animal behavior—therefore, falling under the category of reinforcement learning—, current LCSs have been extended to deal with the other two types of learning, that is, supervised learning (Bacardit and Butz, 2004; Bernadó-Mansilla and Garrell, 2003; Fu et al., 2001; Wilson, 2000) and unsupervised learning (Tamee et al., 2006, 2007; Orriols-Puig et al., 2008g). Thence, LCSs represent a general learning architecture that can be used for different tasks ranging from extracting classification models from streams of labeled data to building clusters online, also including reinforcement learning problems. The flexibility of their architecture is one of the most valuable assets of LCSs with respect to other machine learning techniques, which tend to be designed specifically for one of the three types of machine learning.

The remainder of this chapter is focused on LCSs. We first provide a brief introduction to *evolutionary computation*, which is followed by a more detailed explanation of GAs, since they are the core of the discovery component of LCSs. Then, we present the different types of GBML systems, which represent different ways of using GAs for machine learning, and place LCSs in this big picture.

2.2 Evolutionary Computation and Genetic Algorithms

Evolutionary computation (EC) is a field of study devoted to the design, implementation, and analysis of computation techniques that are inspired by the evolution of biological life in the natural world (Jong, 2006). Actually, evolutionary computation does not refer to a single type of algorithm, but to a series of parallel efforts that shared the idea of using an evolutionary process for computer problem solving. In the following sections, we provide a brief introduction to the biological principles that inspire evolutionary computation methods and propose a taxonomy of the different methods that fall under the definition of evolutionary computation; then, we focus our explanation on *genetic algorithms* (Holland, 1971, 1975), one of the most prominent

techniques in the field of evolutionary computation.

2.2.1 Biological Principles that Inspire Evolutionary Computation

At the beginning of the nineteenth century, the first evolutionary theories, which promoted the hypothesis that the species are a result of the natural evolution, started to emerge. Jean Batista Lamarck was one of the first researchers that rejected the essentialist thought, which was the theory mainly considered at the time. Essentialism relied on the idea that living forms were unchanging. Lamarck proposed some revolutionary theories based on the concept of evolution, which were overlooked by the scientific community at that time. Some decades later, several researchers were inspired by these theories. Among them, there were Wallace and Darwin who independently developed the idea of the mechanism of *natural selection*; this research culminated in the publication of the book *The Origin of Species* by Darwin (1859). From then on, many researchers have adhered to this hypothesis and, currently, the most accepted collection of evolutionary theories is the new-Darwinian paradigm. As proceeds, we provide a brief introduction to the basic concepts of these theories, since evolutionary computation methods are inspired by the evolutionary model proposed by them.

In brief, the evolutionary theory argues that the individuals of a population have a genetic program—i.e., *genotype*—, which defines the genetic constitution of the individual. This genotype, together with the interaction with the environment, forms the *phenotype* of the individual, that is, the observable constitution of the organism. Then, the theory states that life can be accounted for by four physical processes operating on and within populations of species: reproduction, mutation, competition, and selection. That is, individuals of a population:

1. *are reproduced*, transferring the genotype of parents to offspring;
2. *are mutated*; that is, errors in the process of information transfer inevitably occur;
3. *compete*, as a consequence of creating new individuals—over-reproducing the species—in an environment with finite resources; and
4. *are selected*, as an inevitable result of competition due to the existence of finite resources.

Therefore, this results in a cycle where species evolve by means of individual competition for a limited amount of resources. Individuals whose phenotypes are better adapted to the environment are stronger and have higher probability to survive in competition with poorly adapted individuals.

Note that stochastic processes play a key role in the theory of evolution. That is, genetic variation by means of mutation is a chance phenomenon, since errors in information transfer are unpredictable. Also, selection is probabilistic; although the quality of the individual is one of the most important aspects for its survival, there are many external factors that may influence the selection process.

The ideas briefly presented in this section were taken as inspiration by different researchers who identified the evolutionary process as an appealing approach to solve optimization problems. Consequently, several authors started their ways on designing optimization methods that

simulate different aspects of evolution, which, nowadays, have been grouped under the evolutionary computation term. A taxonomy of these different methods is provided in the following section.

2.2.2 Evolutionary Computation: A Taxonomy

In the 1950s, some researchers started to develop the idea of using biological principles to design evolutionary problem solvers. At that time, there were the first attempts to apply these types of computer problems solvers to *automatic programming*—that is, to find a program that calculates input-output functions—(Friedberg, 1958; Friedberg et al., 1959), to numerical optimization problems (Bremermann, 1962), and to the design and analysis of industrial experiments (Box, 1957; Box and Draper, 1969). These early efforts were followed by the establishment, in the middle 1960s, of three main forms of evolutionary computation: *genetic algorithms* (Holland, 1967, 1971, 1975), *evolution strategies* (Rechenberg, 1965, 1973; Schwefel, 1981), and *evolutionary programming* (Fogel, 1962, 1964). Over the next 25 years, these three branches developed quite independently; not until the early 1990s, was the term *evolutionary computation* created to embrace these different technologies, which were considered different “dialects” of biology-inspired problem solvers.

Since then, the strong research on evolutionary computation has resulted in new branches of evolutionary solvers. Two of the most significant of these new approaches are *genetic programming* (Koza, 1989, 1992)—introduced as an extension of genetic algorithms to evolve computer programs—and estimation of distribution algorithms (EDAs) (Pelikan et al., 2000b; Larrañaga and Lozano, 2002; Pelikan et al., 2006)—a new approach that creates probabilistic models to solve optimization problems. In what follows, each one of these families is shortly introduced.

Genetic algorithms (GAs) were originally created by Holland (1967, 1971, 1975) with the initial aim of understanding the underlying principles of adaptive systems, and further propelled by Goldberg, who presented GAs to a broad audience by simply and precisely presenting theory and applications of GAs (Goldberg, 1989a); later, Goldberg (2002) proposed a methodology to design competent GAs. The key idea of Holland’s work was to use a combination of competition and innovation to build machines that could adapt to changing environments and could respond to unanticipated events; Holland simulated this process with a simple model of evolution that considered the notions of *survival of the fittest* and *continuous production of offspring*. The first implementations of this model used a binary representation and were based on the interaction of population size, crossover, and mutation. These ideas are still valid in current GAs.

Evolution strategies (ESs) were originally proposed by Bienert, Rechenberg, and Schwefel in 1964. The earliest idea of Bienert et al. did not aim at devising a new optimization method, but at building a robot that performed a series of experiments in a slender three-dimensional body so as to minimize its drag; the minimization method relied on changing one variable at each iteration and testing whether this change produced any improvement. ESs were born from this initial idea plus a random process to decide the variable changes (Rechenberg, 1965). The first versions of ESs used a single solution with continuous attributes that was mutated by means of a binomial distribution. The current ESs incorporate a population of solutions and perform a cycle that is similar to GAs, involving crossover, a normally distributed mutation,

and selection. In addition, ESs incorporate mechanisms for self-adapting the mutation operator to each particular individual.

Evolutionary programming (EP) was originally introduced by Fogel (1962, 1964) with the aim of creating a machine with adaptive behavior that could achieve its goals in a range of environments. For this purpose, Fogel identified that the machine should be able (1) to predict its environment and (2) to take appropriate actions in light of the predicted next state. *Finite-state machines* were found as useful to represent the behavior of the machine. Therefore, the evolutionary programming approach proposed to evolve a set of finite-state machines by using mutation as the primary reproductive operator. This general approach was applied to problems in prediction, identification, and automatic control (Fogel, 1964; Fogel et al., 1966).

Genetic programming (GP), initially proposed by Koza (1989, 1992), is an extension of GAs to evolve *computer programs*. To achieve this, GP usually employs a tree-based representation, whose internal nodes are represented with a set of primitive functions and the leaf nodes consist of terminals—usually variables of the problem. GP is based on the same GA cycle, thus involving crossover, mutation, and selection—which are redefined to let them cope with the new representation—on a finite population. GP have been applied to a large variety of problems, ranging from circuit design to quantum computing, which have result in the discovering of several inventions, which were already patented, and new patentable inventions (Koza et al., 2003).

Estimation of distribution algorithms (EDAs) are optimization methods that were recently derived from the field of GAs with the aim of building probabilistic models instead of coding the solution in populations of individuals (Pelikan et al., 2000b; Larrañaga and Lozano, 2002; Pelikan et al., 2006). Also addressed as *probabilistic model-building GAs*, EDAs replace both crossover and mutation with a probabilistic model—built from a data base that contains individuals from the previous generation—from which the new population is sampled. Although removing these two primary operators in evolutionary computation, EDAs can be considered as evolutionary computation methods since they use selection to choose good subsets of samples.

GAs appear as one of the most appealing alternatives among the five branches of evolutionary computation since they were initially designed with the general purpose of understanding natural adaptive systems and designing robust adaptive artifacts instead of specifically focusing on optimization techniques (Rechenberg, 1965) or intelligent agents (Fogel et al., 1966). This is one of the reasons that explain why GAs, as opposed to ESs or EP, have been selected as the primary discovery approach in GBML systems. Due to their importance, the next section explains how GAs work in more detail.

2.2.3 Genetic Algorithms

Genetic Algorithms (Holland, 1971, 1975; Goldberg, 1989a, 2002) are methods for search, optimization, and machine learning that are inspired by natural principles and biology. The key characteristics that differentiate GAs from other optimization techniques are:

- GAs learn from the objective function without assuming any structure or underlying distribution.

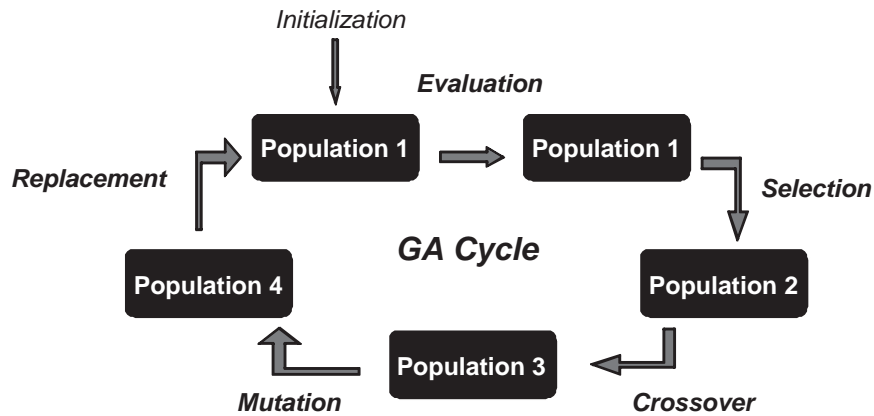


Figure 2.2: Evolution of a GA population.

- GAs search from a population of points that represent candidate solutions, not from a single point.
- GAs code potential solutions instead of directly tuning the decision variables of the problem.
- GAs use random, local operators instead of deterministic, global rules.

As follows, we describe the basic work flow of genetic algorithms, briefly review some existing theory that explains how and why GAs work, and present some of the real-life applications to which GAs have been applied in the fields of engineering, science, and industry.

Description of Genetic Algorithms

GAs evolve a *population* of rules, where each *individual* in the population represents a potential solution to the problem. Analogous to genetics, individuals are represented by *chromosomes*, which encode the decision variables of the optimization problem with a finite-length string. Each of the atomic parts of the chromosome is referred to as *genes*, and the values that the gene can take are addressed as *alleles*. For example, in the traveling salesman problem (Applegate et al., 2006), a chromosome represents a whole route—a sequence of cities—, and a gene represents a city.

To implement natural selection and competition among candidate solutions, GAs incorporate an *evaluation function* that is responsible for assessing the quality of each solution; the quality of each individual is made explicit with a *fitness* value that is given to the individual. Several evaluation functions have been used in GAs, such as mathematical functions provided by human experts or subjective functions where users choose the best solutions from a set of candidates. The design of a fitness function that correctly distinguishes between good solutions and poor solutions is a key point in the success of GAs, since the evolutionary process would push toward the fittest solutions in the population.

The population of individuals is evolved by a continuous process of *selection*, *crossover*, *mutation*, and *replacement* of individuals. Figure 2.2 schematically illustrates the cycle of a GA. Algorithm 2.2.1 complements the explanation with the pseudo code of a simple GA. In the beginning of the run, the population is initialized typically with random individuals—if available, domain-specific knowledge can be incorporated to the initialization process. Then, each individual is evaluated; therefore, each individual has a *fitness* that indicates the quality of the solution. Next, the GA performs a loop where the following for operators are iteratively applied:

- **Selection:** The selection operator chooses the fittest individuals in the population, simulating the survival-of-the-fittest mechanism. So far, several selection schemes have been presented with the common idea of biasing the selection toward the fittest individuals. For example, roulette-wheel selection (Holland, 1975; Goldberg, 1989a) and stochastic universal selection (Baker, 1985; Grefenstette and Baker, 1989) give each individual a selection probability that is proportional to its fitness. Other selection schemes such as tournament selection (Goldberg et al., 1989; Sastry and Goldberg, 2001) and truncation selection (Mühlenbein and Schlierkamp-Voosen, 1993) rank a set of individuals according to their fitness and select the fittest ones.
- **Crossover:** The crossover operator combines the genetic information of two or more parental solutions to create new, possibly better offspring. Recombination plays a key role in GAs, since it should detect important traits of parental solutions and exchange them with the aim of generating better individuals that are not identical to their parents. Several selection operators designed under this goal can be found in (Goldberg, 1989a, 2002; Pelikan et al., 2000a; Pelikan, 2005; Pelikan et al., 2006; Sastry and Goldberg, 2003a).
- **Mutation:** Mutation introduces random errors on the transference of the genetic information from parents to offspring. Thence, this operator acts on single individuals. Although different mutation operators have been designed (Bäck, 1996; Beyer, 1996; Goldberg, 1989a), the commonality among them is that they introduce one or more random changes applied to individual genes. Competent genetic operators that identify important traits of parental solutions and search in the structural neighborhoods of these solutions have been developed Lima et al. (2006); Sastry and Goldberg (2004).
- **Replacement:** After the selected individuals have gone through crossover and mutation, the offspring population replaces the original one. Several replacement schemes could be followed. For example, in a generational GA, all the offspring population may replace the parent population. Other schemes are elitist replacement—the elite individuals of the parent population are copied to the new population—or steady state replacement—the best individuals of the offspring population are copied to the original one, removing classifiers with poor fitness.

The synergy of all these operators pressures toward the evolution and selection of the best solutions, which are recombined yielding new promising offspring. Goldberg (2002) emphasized the idea that, while selection, crossover, and mutation can be shown to be ineffective when applied individually, they might produce a useful result when working together. This was explained with the *fundamental intuition of GAs*, which argues that the combination of the

Algorithm 2.2.1: Pseudo code of a simple GA.

Data: t is the time stamp and $P(t)$ is the population at time t

```
1 Algorithm: GA
2  $t := 0$ 
3  $P(t) :=$  Initialize randomly  $P(t)$ 
4  $P(t) :=$  Evaluate  $P(t)$ 
5 while not finish do
6    $t := t+1$ 
7    $P'(t) :=$  Select individuals from  $P(t-1)$ 
8    $P'(t) :=$  Apply crossover to  $P'(t)$ 
9    $P'(t) :=$  Apply mutation to  $P'(t)$ 
10   $P(t) := P'(t)$ 
11   $P(t) :=$  Evaluate  $P'(t)$ 
12 end
```

selection and *crossover* operators introduces a process of *innovation* or *cross-fertilizing*, whereas the combination of *selection* and *mutation* represents the *continuous improvement* or *local search* process.

After outlining a GA procedure and discussing the role of the most important operators, the next section briefly reviews some theory that provides key insights that help explain why GAs work.

2.2.4 Basic Theory of GA

Since the initial definition of GAs, several authors have developed formal theory to explain their behavior. In the following, we first go back to Holland (1975) and introduce the *schema theorem*, which uses the concept of *building block* (BB) to give some insights on how GAs work. Then, we present the work by Goldberg (2002), who adheres to the ideas proposed by the schema theorem and proposes a methodology for designing competent selecto-recombinative GAs.

Intuitive Idea of Why GAs Work: the Schema Theorem

We have just seen that the operation of GAs is based on the exchange of information from parents to offspring. Along the description of GAs, we already pointed out that key operators such as crossover should detect important traits from parents and exchange them properly to create new children. In this section, we further this idea and present the schema theorem, which is concerned about accounting for how the key solutions evolve in a population. We start with the definition of *schema* and then reproduce the *schema theorem* proposed by Holland (1975).

The *schema theorem* is based on the idea of *schema* or *building blocks* (BBs), that is, a template that identifies a subset of individuals. A schema is represented with a string $s = (s_1, s_2, \dots, s_\ell)$ where each bit s_i can take a different value of the ternary alphabet $\{0, 1, *\}$ (ℓ is the total number of bits of the schema). Thence, a *schema* represents a subspace $B^n = \{0, 1\}^n$

so that a binary string x belongs to this schema ($x \in B^n$) if

$$x_i \neq s_i \Leftrightarrow s_i = * \quad \forall i = 1, 2, \dots, n. \quad (2.1)$$

Thence, for example, provided the schema $1^{*}001$, instances 101001 and 111001 , among others, belong to this schema.

Before proceeding to the formalization, the following two concepts need to be defined:

- The *order* of the schema h , $o(h)$, is the number of fixed positions in the schema, that is, the number of bits that are 0- or 1-valued. For example, $o(* * 10*) = 2$.
- The *length* of the schema h , $\delta(h)$ is the distance between the first and the last specific positions. For instance, $\delta(* * 10 * *) = 1$.

Provided the definitions above, the schema theorem models how the different schemas evolve along a GA run. For this purpose, it considers the effects of the selection, the crossover, and the mutation operators. Moreover, it assumes fitness-proportionate selection, one point crossover, and gene-wise mutation. Then, the schema theorem demonstrates that the expected number of offspring that belong to schema s at iteration $t + 1$, i.e., $E[N_S(P(t + 1))|P(t)]$, satisfies that

$$E[N_h(P(t + 1))|P(t)] \geq N_h(P(t)) \frac{\bar{f}(h, t)}{\bar{f}(t)} \left(1 - \frac{\delta(h)}{\ell - 1} p_c \right) (1 - p_m)^{o(h)}, \quad (2.2)$$

where $N_S(P(t))$ is the number of individuals in the population $P(t)$ that belong to schema h at time t ; $\bar{f}(h, t)$ is the average fitness of the individuals that belong to h at time t ; and $\bar{f}(t)$ is the average fitness of the population. The effect of fitness-proportionate selection is given by the term $\frac{\bar{f}(h, t)}{\bar{f}(t)}$, which increments the expectation of the number of individuals in the next generation if the average fitness of the individual that belong to schema h is greater than the average fitness of the population. The effect of crossover is reflected in the term $1 - \frac{\delta(h)}{\ell - 1} p_c$, which indicates that the probability that a schema survives depends on the length of the schema and the crossover probability. Finally, the effect of mutation is modeled by the term $(1 - p_m)^{o(h)}$, which denotes that the probability that the schema is preserved to the next generation is inversely proportional to the mutation probability and exponentially proportional to the number of fixed bits of the schema ($o(h)$).

Thence, the schema theorem demonstrates that the expected number of individuals that belong to schema h at time $t + 1$ grows exponentially if the average fitness of the individuals that belong to schema h at time t is greater than the average fitness of the population at time t . Therefore, the effect of reproduction becomes quantitatively clear; that is, reproduction allocates exponentially increasing number of trials to schemas whose fitness is above the average.

Design Decomposition: Goldberg's Approach to Competent GA Design

Although some researchers have strongly criticized or even rejected the schema theorem, Goldberg proposed a framework to design selecto-recombinative GAs based on the initial Holland's notion of building block. Goldberg (2002) suggested thinking of building blocks as a kind of

matter and to ensure (1) that we have an initial stock of them, (2) that good ones grow in the market share, (3) that good decisions are made among them, and (4) that they are exchanged well to solve a large class of difficult problems.

In order to satisfy the four aforementioned points, Goldberg (2002) decomposes the problem of designing competent selecto-recombinative GAs in the following seven aspects:

1. Know that GAs process BBs.
2. Know the BB challengers.
3. Ensure adequate supply of raw BB.
4. Ensure increased market share for superior BBs.
5. Know BB takeover and convergence times.
6. Make decision well among competing BBs.
7. Mix BBs well.

Goldberg (2002) proposed to examine these items by means of facetwise analysis, which suggests analyzing separately each one of these elements, assuming that the other ones behave in an ideal manner. As proceeds, we elaborate on each one of the elements and mention some of the approaches by which GA researchers have studied each element.

The primary idea of this theory is that selecto-recombinative GAs work through a mechanism of *decomposition* and *reassembling*. That is, GAs implicitly decompose the problem and identify sets of well-adapted features, which form a building block. Then, these building blocks have to be correctly processed.

The second key idea in this theory is that complex problems are those problems whose building blocks are difficult to acquire. This could be a result of having large, complex building blocks, having building blocks that are hard to separate, or having a deceptive guidance toward high-order building blocks (Goldberg, 2002).

After identifying the first two key concepts, the next four items of the theory analyze how these building blocks evolve in a *market economy of ideas*. First, we need to ensure that the market is provided with enough stock of BBs. As GA populations are usually initialized randomly, one way to obtain more variability is to use larger populations (Goldberg, 1989b; Goldberg et al., 2001; Holland, 1975).

Having provided the population with an initial stock of BBs, the next two important aspects are (1) that the best BBs should grow and take over a dominant market share of the population, and (2) that this growth should be neither too slow—so, delaying the convergence time—, nor too quick—, thus increasing the risk of falling in a local optimum. Different approaches have been taken to understand time and convergence, which cover the fourth and fifth elements of the design decomposition. Three of the most important approaches are (1) takeover time models, which model the dynamics of the best individual (Bäck, 1994; Cantú-Paz, 1999b; Goldberg and Deb, 2003), (2) selection-intensity models, where the dynamics of the average fitness of the population are modeled (Bäck, 1995; Miller and Goldberg, 1995, 1996; Mühlenbein and Schlierkamp-Voosen, 1993; Thierens and Goldberg, 1994a,b), and (3) high-order cumulant models, where models of

the dynamics of average and high-order cumulants are developed (Blickle and Thiele, 1995, 1996; Cantú-Paz, 1999a).

Yet, just ensuring an initial adequate supply of raw BBs is not enough; in addition, good decisions among competing BBs need to be taken to ensure that the best BBs will grow in the market. It has been acknowledged that as we increase the population size, we increase the likelihood of making the best possible decisions (Jong, 1975; Goldberg et al., 1992; Goldberg and Rudnick, 1991; Harik et al., 1999). Therefore, decision making has been studied from the perspective of population sizing.

The last item of the design decomposition relies on the idea that the correct identification and exchange of BBs is the critical path to innovative success. That is, when designing a competent GA, one of the key challenges that needs to be addressed is how to identify BBs and exchange them effectively. In this regard, facetwise models have been developed to show that fixed-recombination operators, such as uniform crossover, may fail to effectively identify and exchange BBs, resulting in an exponential scaling up of the population size in boundedly difficult problems—that is, problems that, for example, have large sub-solutions that cannot be decomposed in simpler sub-solutions, have several optima, or are affected by noise—(Goldberg et al., 1993; Sastry and Goldberg, 2002, 2003b). In contrast, recombination operators that can automatically identify and exchange BBs efficiently have shown to scale up polynomially with the population size in these boundedly difficult problems (Goldberg, 2002; Pelikan, 2005; Pelikan et al., 2006).

The design decomposition and facetwise analysis has resulted in a better understanding of the underlying processes of GAs, creating a formal framework formed by different pieces of theory. Furthermore, these analyses have been used as a tool for designing *competent GAs*, genetic algorithms that can solve boundedly difficult problems quickly, reliably, and accurately (Goldberg, 2002). The first designs of competent GAs can be found in *messy GA* (Goldberg et al., 1989). Currently, there are several implementations of competent GA such as the *linkage learning genetic algorithm* (Harik, 1997), the *extended compact genetic algorithm* (Harik, 1999; Sastry and Orriols-Puig, 2007), or the *Bayesian optimization algorithm* (Pelikan et al., 1999). The maturity in the GA field has promoted the use of GAs in real-world problems. The next section reviews some of these important applications.

2.2.5 Genetic Algorithms in Real-World Applications

All the success and better understanding of genetic algorithms has led to their application to a large variety of problems in science, engineering, and industry. Therefore, GAs have not been stuck in “toyish” problems but have been applied to complex, previously unsolved, real-world problems. We review some of the most important applications in what follows.

In the scientific field, GAs have been employed in different applications such as the detection of coronary problems (Grefenstette and Fitzpatrick, 1992), the design and interaction in computer games (Jo and Ahn, 2002), and the generation of music (Goksu et al., 2005). But the application of genetic algorithms, differently from other optimization techniques, is not merely limited to a scientific field. GAs have been successfully applied to complex problems in industry, providing novel solutions. For instance, GAs were used to partially design the Japanese bullet train N700; specifically, the shape of the front of the train was optimized by a GA. Another

significant example is the EvoFIT tool¹, a system based on GAs that makes robot pictures, which was used by the Northamptonshire police.

There are also some companies that use GAs as the heart of their applications such as *Optimatics* and *Schema*. *Optimatics*² is a world leader provider of innovative and customized optimization solutions to water industry. This company uses GAs as an essential tool for optimization. The results provided by the company highlight that the GA-based optimization has resulted in savings of about 20%-30%, on average, in their projects. *Schema*³ is a global provider of end-to-end network optimization solutions for transport and mobile networks that uses GAs in their applications. Some of the most significant projects of this company are missile balancing, synthetic aperture radar, optimal container stowage, and frequency allocation for cellular networks.

Therefore, GAs have been used as competent optimization tools in some complex scientific and industrial applications, assisting the creation of commercial products. In addition to these applications in the optimization realm, GAs have also been used as the heart of several machine learning techniques, yielding to a discipline which has been referred to as *genetic-based machine learning*. The next section reviews the main branches of the algorithms that fall under these definitions, which includes both Michigan- and Pittsburgh-style LCSs.

2.3 Genetic-based Machine Learning and Learning Classifier Systems

The application of GAs has not been restricted to optimization problems, but they have also been used as the primary discovery heuristic in machine learning procedures. Since Holland (1962) outlined his theory for adaptive systems, GAs have been used as the main discovery component in Michigan-style LCSs (Holland, 1976; Holland and Reitman, 1978) and Pittsburgh-style LCSs (Smith, 1980, 1983, 1984), which conform the two original branches of GBML. Furthermore, the population-based search, robustness, and knowledge-representation flexibility of GAs, coupled with the recent advances in efficiency and competent GAs (Goldberg, 2002; Pelikan, 2005; Pelikan et al., 2006; Goldberg et al., 2007), has promoted the use of genetic search as the primary discovery heuristic in several machine learning techniques that belong to different learning paradigms that range from neural networks (Kitano, 1990; McInerney and Dhawan, 1993; Liu et al., 2004; Wierstra et al., 2005; Mierswa, 2007) to probabilistic classifiers (del Jesus et al., 2004; Otero and Sánchez, 2006; Yalabik and Fatos, 2007). This has resulted in several new approaches to use GAs in machine learning, which have shown to be highly competitive with respect to traditional non-evolutionary systems (Orriols-Puig et al., 2008e,d).

The purpose of this section is to describe the five main branches of GBML. We start with the description of Michigan- and Pittsburgh-style LCSs. As several particular implementations have been designed for both types of systems, we provide a general schema for each LCS. Then, we present three other forms of GBML that have received a special amount of attention during the last decade: *Iterative rule learning* (IRL) (Venturini, 1993), *genetic cooperative-competitive learning* (GCCL) (Giordana and Neri, 1995; Greene and Smith, 1993), and the

¹<http://www.evofit.co.uk>

²<http://www.optimatics.com>

³<http://www.schema.com>

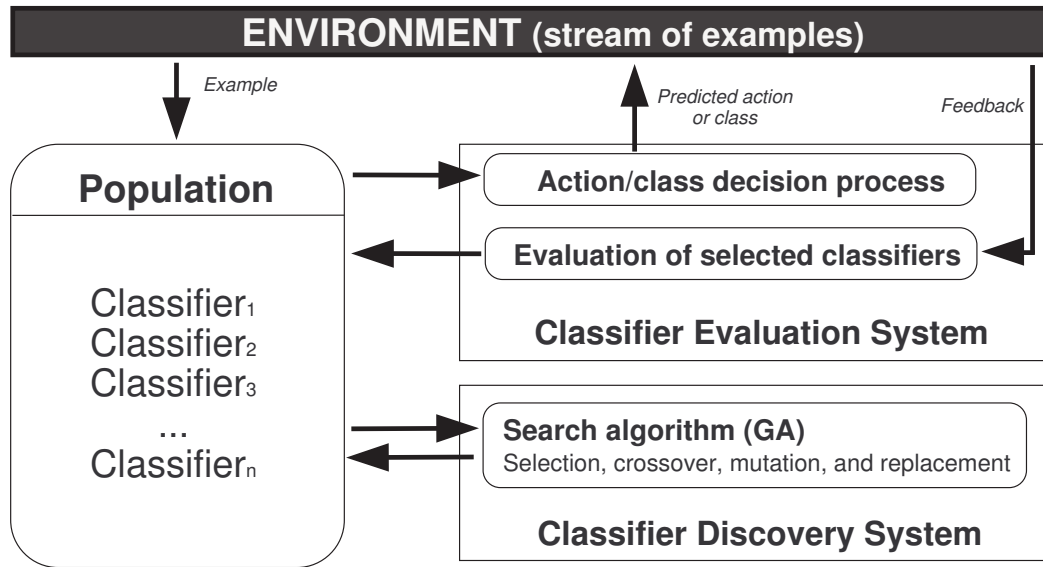


Figure 2.3: Simplified schematic of Michigan-style LCSs which the typical process organization.

organizational classifier system (OCS) (Wilcox, 1995). All these three approaches are combines of Michigan- and Pittsburgh-style LCSs. The IRL approach uses a Michigan-like representation in a Pittsburgh-style LCSs to learn a set of rules incrementally. GCCL systems define a framework where both competition in system niches and cooperation among all rules are performed. OCS distributes classifiers in organizations and takes ideas from the economic study of transaction costs to control the sizes of these organizations. A general schema of the process organization of each branch of GBML is presented as follows.

2.3.1 Michigan-style LCSs

Since the first successful implementation of a Michigan-style LCSs (Holland and Reitman, 1978), research on Michigan-style LCSs has resulted in new systems that have been applied to different types of learning tasks. Therefore, although initially designed to simulate animal behavior—later inspiring the whole field of reinforcement learning (Sutton and Barto, 1998)—, current LCSs can be applied to a large variety of learning tasks such as supervised learning and data mining (Bernadó-Mansilla and Garrell, 2003; Bull, 2004; Bull et al., 2008), function approximation (Wilson, 2002b; Butz et al., 2008), reinforcement learning (Lanzi, 1999b, 2002; Lanzi et al., 2005; Butz et al., 2005a), and clustering (Tamee et al., 2006, 2007). As proceeds, we present a general architecture that highlights the common points among the different implementations.

Figure 2.3 illustrates the common process organization of current Michigan-style LCSs. That is, all Michigan-style LCSs share three key components that distinguish them from other GBML and machine learning techniques:

1. a knowledge representation based on *classifiers*, which maps the inputs with classes or actions,

2. a *classifier evaluation system* which evaluates the population of classifiers online, and
3. a *classifier discovery system* that is triggered with a certain frequency and is responsible for discovering new promising classifiers and adapting the knowledge base to eventual changes in the environment.

It is worth highlighting that the system learns online from an environment, which can represent either the environment in which an agent lives or a set of examples that are made available in a data stream.

The core of the system maintains a population of *classifiers*. Each classifier consists of (a) a structure that maintains an input/output mapping, identifying to which inputs the classifier is applicable and which action should be performed in case of matching, and (b) several parameters that maintain different statistics of each classifier, such as its fitness. The structure that maintains the input/output mapping has usually been implemented with *production rules* (Holland and Reitman, 1978; Wilson, 1994, 1995, 2001; Bernadó-Mansilla and Garrell, 2003); other implementations such as neural networks (Bull and O'Hara, 2002), first-order logic expressions (Mellor, 2005), messy representations (Lanzi, 1999a), LISP s-expressions (Lanzi and Perrucci, 1999), and gene expression programs (Wilson, 2008) have also been used. In any case, note that each classifier covers a restricted set of sensorial inputs; therefore, the solution of a given problem is the whole population.

Michigan-style LCSs update the parameters of these classifiers online by means of interacting with the environment. That is, at each learning iteration, the environment provides a new input example. Then, the system uses a sub-population of classifiers to decide the action or class that should be taken according to the current input. This action is given to the environment, which, in turn, returns a feedback that indicates the quality of the prediction. Then, the evaluation component uses this information to adjust the quality of the classifiers that have participated in the action decision process. Moreover, with a certain frequency, the rule discovery system is triggered, generating new promising classifiers. Usually, a niche-based steady-state GA is employed, which selects a group of classifiers, applies genetic operators to create new ones, and introduces them into the population removing other classifiers if there is no room for the new ones. Other search procedures such as evolution strategies have recently been used to guide the classifier discovery system of LCSs (Morales-Ortigosa et al., 2008a,b).

Several Michigan-style LCSs have been designed since the first implementation of CS-1 by Holland and Reitman (1978), such as the EYE-EYE system (Wilson, 1981, 1985a), the Boole system (Wilson, 1985b, 1987)—which took some inspiration from Goldberg (1983) work on LCSs—, and the NewBoole method (Bonelli and Parodi, 1991). Although these systems were able to solve some specific applications, several drawbacks, mainly associated with the achievement of accurate generalizations, hindered their success. Further research resulted in the design of the *extended classifier system* (XCS) by (Wilson, 1995), supposing a tipping point in the LCSs research. As were its ancestors, XCS was originally devised to solve reinforcement learning tasks. Since then, several new Michigan-style LCSs have been designed based on the XCS's architecture. One of these derived systems can be found in UCS (Bernadó-Mansilla and Garrell, 2003), which inherits the main components of XCS, but specializes the system for supervised learning.

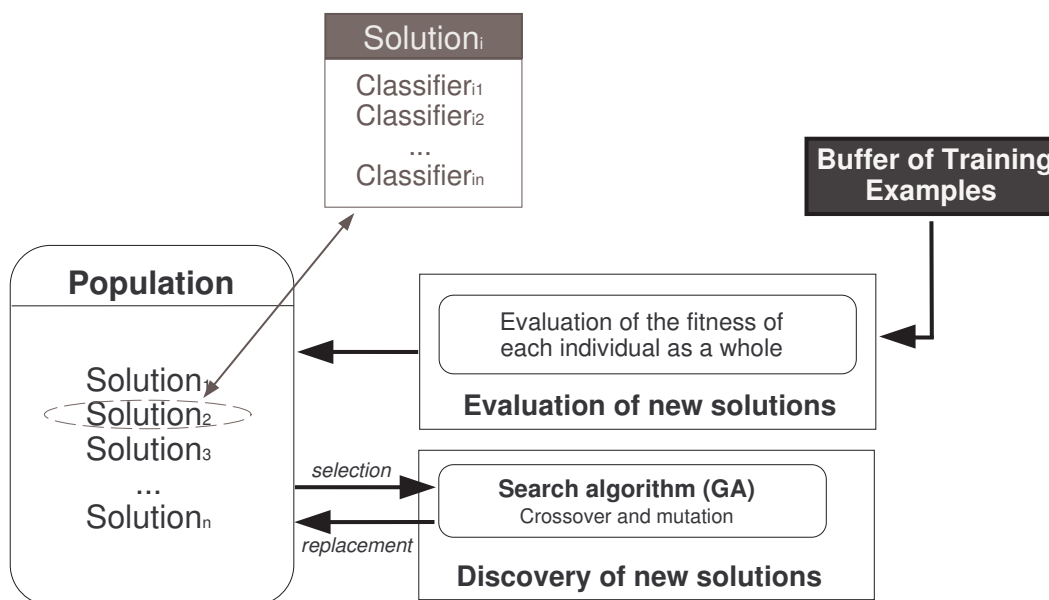


Figure 2.4: Simplified schematic of Pittsburgh-style LCSs.

2.3.2 Pittsburgh-style LCSs

Contemporaneous with the research on Michigan-style LCSs, some authors took another approach and extended GAs to machine learning, resulting in the so-called Pittsburgh-style LCSs. Pittsburgh-style LCSs have three fundamental differences with respect to Michigan-style LCSs: (1) the knowledge representation, (2) the evaluation system, and (3) the application mode of the GA and the definition of the genetic operators. In this section, we examine these differences, describe a general process organization of Pittsburgh-style LCSs, and review some of the most significant implementations in this area.

Figure 2.4 illustrates the process organization of a Pittsburgh-style LCSs, which is directly extended from the typical process organization of a simple GA. In Pittsburgh-style LCSs, individuals are complete solutions to the whole problem; that is, each individual should cover all the feature space, instead of only covering a portion of it as in the Michigan approach. Usually, Pittsburgh-style LCSs represent individuals as a disjunction of rules—which in most cases are made available as a decision list (Rivest, 1987). Nonetheless, other representations such as a set of decision trees (Llorà and Garrell, 2001; Llorà and Wilson, 2004) have also been used. In the remainder of this section, for consistency with the Michigan approach, we use the term *classifier* to refer to each one of fundamental parts of an individual.

Since each individual maintains a set of classifiers, which jointly cover the whole problem, there is no need for evaluating the quality of each of these classifiers on its own. Therefore, differently from the Michigan approach, the classifier apportionment algorithm can be sidestepped; instead, a single measure is enough to evaluate the quality of the whole individual. Different indicators have been used to evaluate the quality of individuals, the prediction accuracy and the generality of the individuals being the most common ones. Thence, immediately after its

creation, each individual is evaluated offline with a set of examples, which either have been provided at the beginning of the run in the form of a static data set or have been collected during the learning process. Note that, under this approach, there is no control about the contribution of each classifier to the performance of the whole individual.

Then, the population is evolved by means of genetic algorithm cycles. That is, at each iteration, the system selects a set of individuals, which are crossed, mutated, and inserted into the population replacing other probably low fit individuals. The crossover and mutation operators are adapted to deal with the representation of the individuals. At the end of the learning process, the best individual in the population is used to predict the output of new test examples.

After the implementation of LS-1 (Smith, 1980, 1983, 1984), the first Pittsburgh-style LCS, there have been some successful developments of Pittsburgh-style LCSs for supervised learning such as GABL (Jong and Spears, 1991) and GIL (Janikow, 1993). In GABL, each individual is encoded with a variable-length set of rules, and each rule follows a fixed-length, binary representation. Rules have no class associated since GABL performs concept learning, that is, it learns only positive or negative examples. The fitness is computed as the squared accuracy function. The system uses the typical genetic operators except for crossover, which is restricted to ensure that the operator selects the same position to cut the variables of two parents. GIL follows a similar scheme but uses rules defined in the VL_1 logic (Michalski et al., 1986) and a fitness function that tries to balance the accuracy-complexity tradeoff of the individuals. In addition, the system is provided with several operators that modify the rules at the semantic level. A more recent implementation of a Pittsburgh-style LCS, which overcomes the scalability problems detected in previous approaches (Freitas, 2002), can be found in GAssist (Bacardit, 2004).

2.3.3 Iterative Rule Genetic-based Machine Learning

Iterative rule learning (IRL) follows a *separate-and-conquer* methodology (Pagallo and Haussler, 1990) to learn a set of rules. The separate-and-conquer methodology proposes to iteratively learn rules that cover a subset of the input instances. That is, the following two steps are iteratively performed: (1) learn a rule that covers part (or all) of the training examples, and (2) remove the covered examples from the training set. This process is repeated until no training examples remain. At the end of the process, the solution is the concatenation of the rules created at each iteration. Notice that this approach incrementally creates new rules and, at the same time, reduces the search space since the covered examples are removed from the training data set. This method has also been referred to as the *covering* strategy (Michalski, 1969).

Thence, IRL defines a general learning architecture in which different learning procedures could be applied to extract the individual rules. Among others, GAs have been used to discover these rules. That is, at each learning iteration, a GA is applied to induce a population of rules. Therefore, the knowledge representation in the GA is the same as in the Michigan approach, but rules compete with all the other rules in the population and are evaluated offline as in the Pittsburgh approach.

The first proposal of IRL in the context of GAs can be found in the SIA system (Venturini, 1993). SIA generates an initial population from generalizations of randomly selected instances,

and a GA is used to evolve these rules. Rules are evaluated according to their complexity and accuracy. The process stops when the best rule remains stable for a certain number of generations. More recent approaches can be found in the HIDER system (Aguilar-Ruiz et al., 2003, 2007) and the NAX method (Llorà et al., 2007) for classification tasks and the HIRElin technique (Teixidó-Navarro et al., 2008) for function approximation tasks. A common characteristic of these three learning algorithms is that they make the rules available as a decision list (Rivest, 1987). GAs for IRL have also been extensively used in genetic fuzzy systems (Cordón et al., 2001a; González and Pérez, 1999).

2.3.4 Genetic Cooperative-Competitive Learning

Genetic cooperative-competitive learning was initially designed as a synthesis of aspects of both Michigan- and Pittsburgh-style LCSs (Greene and Smith, 1993). This approach combines the offline rule processing of Pittsburgh-style LCSs with the idea of Michigan-style LCSs that the solution is the whole population, and so, that rules need to collaborate to cover all the input space. Below, we provide a general schema of this type of GBML systems in some detail.

GCCL was born with the purpose of explicitly addressing the goal of constructing highly accurate and as-simple-as-possible decision models from a set of examples. To achieve this, GCCL systems approach this problem by assuming that the examples of the training data set correspond to niches in an *ecology*. The exact number of niches is not known, but it is assumed to be less than the total number of examples in the data set; therefore, several examples can be placed in the same ecological niche. Then, the population is considered to be the whole model, which represents all the niches of the ecology, and each individual is a representation of a particular niche. Individuals are coded as single rules, and the examples that are correctly predicted by the individual are assigned to this rule. Then, the objective is to learn the minimum number of niches or individuals that can cover all the input instances accurately.

The first proposal of a GCCL system can be found in COGIN (Greene and Smith, 1993) which was designed after several works on the application of GAs to symbolic induction problems that produced significant systems such as ADAM (Greene, 1987; Greene and Smith, 1987) and GARGLE (Greene, 1992). Later, Giordana and Neri (1995) designed a new GCCL addressed as REGAL, which was based on their previous work on concept learning based on GAs. The main novelty of the system is that it provided a new selection operator that allowed the population to converge, on average, to an equilibrium state.

2.3.5 The Organizational Classifier System

The organizational classifier system takes ideas from both Michigan- and Pittsburgh-style LCSs to debate on appropriately sizing organizations, simulating the economic idea of transaction costs. In what follows, we briefly review the architecture of OCS and discuss the novelties of this approach.

OCS inherits the main ideas of simple classifier systems and focuses on the problem of trying to distinguish rules that lead to optimal decisions from those that lead to suboptimal decisions in order to evolve ideal rule sets. For this purpose, the system distributes the classifiers of the population in different organizations of variable size. These organizations can interact among

themselves. To control the size of the organizations, OCS incorporates ideas from transaction cost theory by using reputation for organizational recruitment and by paying attention to efficient organization sizing. That is, on the one hand, OCS includes a credit-allocation scheme that distributes reputation among classifiers and organizations and a conflict-resolution method that uses rules and organizations reputation to determine the interactions among classifiers and organizations. On the other hand, the system implements an organizational growth component that controls the sizes of the organizations by applying different genetic operators to enlarge or shrink organizations, which preserve the idea that organizations with larger reputation may be larger than organizations with lower reputation.

Despite the novelty of the ideas proposed in the OCS framework, research on OCS systems alike has been scarce during the last decade. Recently, these concepts have been applied by Vallim et al. (2008) to deal with problems of multi-label classification.

In this section, we presented four branches of GBML, which share the use of a GAs for machine learning. Among them, this thesis is focused on Michigan-style LCSs. The most important reasons that led us to research on these types of LCSs is that Michigan-style LCSs

1. Evolve a distributed solution in parallel, applying local search procedures to niches instead of optimizing a set of classifiers globally.
2. Create individual classifiers whose contribution to the whole is determined by the system; therefore, each individual classifier can be regarded as an expert in the region of the feature space that it covers.
3. Learn the model online from a stream of examples. This is not only useful for reinforcement learning problems—where instances come online as the agent finds new sensorial states while moving around its environment—, but also for tackling current industrial and scientific applications in which large volumes of data are generated online, and the learning systems need to extract the key information that resides in the stream of data on the fly.

These three characteristics, together with the increasing application of LCSs to new real-world problems, encouraged us to take this approach in the present work.

2.4 Summary

This chapter provided a brief introduction to ML and to the use of GAs in ML. Starting from a brief description of ML and a classic taxonomy of the different ML tasks, we introduced evolutionary computation methods in general, and GAs in particular, as robust optimization techniques. Then, we explained different types of algorithms that use GAs to evolve their knowledge representation, placing LCSs in this context.

The present work focuses on Michigan-style LCSs, the original approach to use GAs for machine learning. While this chapter has provided a general introduction to these types of systems, the next chapter focuses on the two approaches studied in this thesis: XCS and UCS. We consider XCS since it is, by far, the most influential Michigan-style LCS, which has been widely used to solve different types of problems. Besides, this thesis is also interested in UCS,

an extension of XCS that restricts the learning architecture to supervised learning with the aim of dealing with classification problems more efficiently. In the next chapter, these two LCSs are described in detail.

Chapter 3

Description of XCS and UCS

The design of the *extended classifier system* (XCS) by Wilson (1995) supposed a milestone in the history of learning classifier systems. Wilson proposed XCS after several years of research that yielded important results such as the *boole* system (Wilson, 1985b, 1987) or the most recent *zerth-level classifier system* (ZCS) (Wilson, 1994). The success of XCS was mainly due to its “simplified” structure which addressed the different challenges of LCSs at that time. XCS avoided the evolution of an excessive number of over-general classifiers by basing fitness on the accuracy of the reward prediction instead of on the prediction itself. Besides, XCS provided intrinsic generalization capabilities due to the combination of a niche-based GA and a population-wise deletion operator.

Since its first proposal in 1995, a lot of research has been conducted on formalizing the algorithmic structure (Butz and Wilson, 2001), enhancing the system with new operators (Wilson, 1998; Kovacs, 1999; Butz et al., 2003), and deriving theory for a better understanding of its underlying processes (Butz and Pelikan, 2001; Butz et al., 2004b, 2005a, 2007; Drugowitsch and Barry, 2008; Drugowitsch, 2008). Besides, new systems have been derived from XCS for specific types of learning tasks. In the context of supervised learning, Bernadó-Mansilla and Garrell (2003) defined the *supervised classifier system* (UCS), an LCS that inherited the process organization from XCS, but was specialized for supervised learning. Since in this thesis we are especially concerned about solving supervised learning tasks, we consider both XCS—as being the general learning architecture—and UCS—as being specialized for these types of tasks.

The purpose of this chapter is to provide a concise description of both XCS and UCS. Section 3.1 introduces the XCS architecture and further details the different components of the system and the process organization; besides, we provide some theory that explains why XCS is able to generalize from a set of examples. Section 3.2 presents UCS, focusing on the modifications introduced with respect to the online architecture of XCS. In both cases, we assume a ternary representation. Section 3.3 reviews some new representations proposed to deal with new types of data more effectively. Finally, section 3.4 summarizes the chapter.

3.1 The XCS Classifier System

XCS (Wilson, 1995, 1998) is a Michigan-style LCSs that evolves a population of *classifiers*—usually, production rules—online by means the interaction with an environment. A steady state genetic algorithm is responsible for evolving these classifiers online. The main differences between XCS and other Michigan style LCSs are (1) that XCS simplifies the architecture of other Michigan-style LCSs—for example, removing the message list—and (2) that XCS computes the classifiers’s fitness based on the accuracy of the reward prediction instead of calculating the fitness from the reward prediction itself. Due to the latter aspect, XCS creates a set of maximally general and accurate classifiers that map the problem space completely; that is, classifiers with both low and high expected prediction reward are evolved by the system, creating the so-called *complete action map*.

As follows, we explain the learning architecture of XCS in detail. First, we review the knowledge representation assuming that the classifiers are represented with ternary rules, as done in the first versions of the system. Then, we explain the process organization of the system, which includes the learning interaction, the classifier evaluation system, the discovery component, and the reasoning mechanism to infer the action of a new input instance. Finally, we review theory that explains why XCS is able to generalize and learn a set of maximally general and accurate classifiers. All the explanation assumes that XCS is working in single step tasks. For more details about the architectural changes needed to deal with multiple step problems, the user is referred to (Wilson, 1995, 1998; Butz et al., 2005a).

3.1.1 Knowledge Representation

XCS evolves a distributed knowledge represented by a *population* [P] of *classifiers*, where each classifier contains a rule and a set of parameters that estimate the quality of the rule. Different rule representations have been designed for XCS so far. In general, XCS can evolve any type of rule—or even, other types of representations such as trees or neural networks—provided that the genetic operators are properly redefined. In this section, we consider the ternary rule representation, since this was the representation originally designed with the system. Later, in section 3.3, we present different types of rules representations that have been designed for XCS and UCS in the last few years.

A rule takes the form

$$\mathbf{if\ condition\ then\ action.} \tag{3.1}$$

That is, it consists of a *condition*, which is formed by a set of variables in disjunctive normal form that specify when the classifier is applicable, and an *action*, which determines the predicted action or class. Each variable of the condition can take a value of the ternary alphabet $\{0, 1, \#\}^\ell$, where ℓ is the number of input variables. The *don’t care* symbol ‘#’ allows for rule generalization; that is, ‘#’ indicates that the given variable matches any input value. Therefore, a rule k matches an input example e if for each variable v_i : $v_i^k = e_i \vee v_i^k = \#$.

Besides the rule, a classifier also contains a set of parameters that maintain different statistics of the rules. The most important parameters associated with a classifier are:

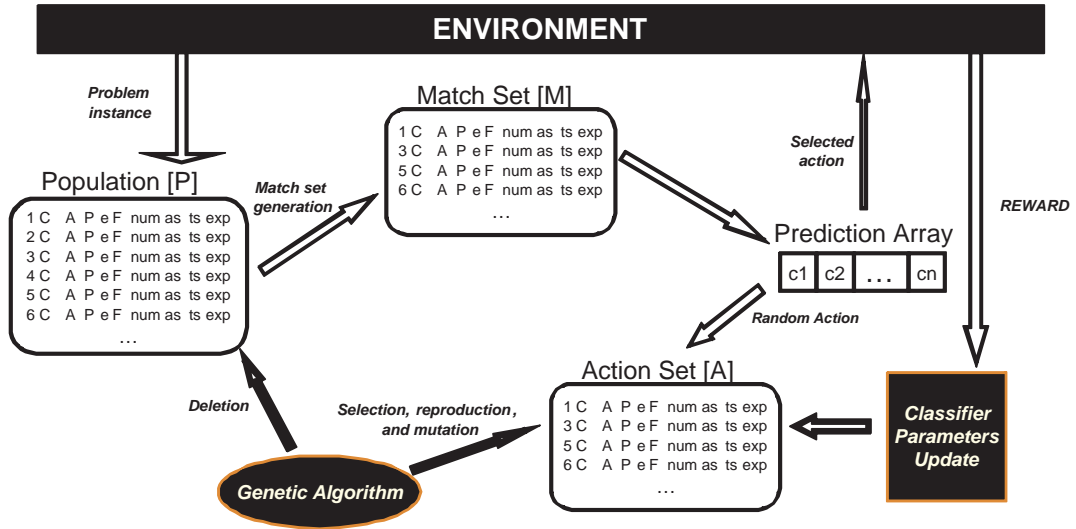


Figure 3.1: Schematic of the process organization of XCS.

1. The payoff prediction p , an estimate of the payoff that the classifier will receive if its condition matches and its action is chosen.
2. The prediction error ϵ , an estimate of the average error between the classifier's prediction and the received payoff; that is, it computes the mean absolute deviation of the prediction error with respect to the received rewards.
3. The fitness F , an estimate of the scaled, relative accuracy¹ of the payoff prediction.
4. The action set size as , an estimate of the size of the action sets in which the classifier has participated (see section 3.1.2).
5. The experience exp , which reckons the number of examples that the classifier has matched during its life.
6. The numerosity n , which indicates the number of copies of the classifier in the population. In this way, identical classifiers can be represented as a single individual in the population, speeding up the runtime since the matching time (as well as the time required for other operations) decreases.

To completely understand the knowledge representation, in the following sections we detail how the different components of XCS interact to evaluate the existing rules and to create new promising classifiers.

3.1.2 Learning Interaction

XCS learns online by interacting with an environment which provides a new training example at each iteration. Figure 3.1 schematically illustrates this process. The system works in two

¹Relative accuracy is computed with respect to other classifiers in the same action set.

different modes: *exploration* or training and *exploitation* or test. In exploration mode, XCS seeks to evolve a maximally general rule set that minimizes the prediction error of the rules. In exploitation mode, XCS uses the rule set to decide the best action for a new input example. As proceeds, we discuss in more detail how the different components of XCS interact to learn a population of maximally general and accurate classifiers from the interaction with this environment; that is, we focus on the exploration phase. In section 3.1.5, we explain how the evolved knowledge is exploited to predict the action of new inputs.

XCS usually starts the *exploration* phase with an empty population. At each learning iteration, the system is provided with a new instance e . Then, the system builds a *match set* $[M]$ containing all the classifiers in $[P]$ whose conditions match e . If the number of classes represented in $[M]$ is less than the θ_{mna} threshold (θ_{mna} is usually set to the total number of possible classes of the problem), the *covering* operator is triggered, creating as many new classifiers as required to cover θ_{mna} different classes. The condition of the new classifiers created by covering is generalized from e . That is, each variable is set to ‘#’ with probability $P_{\#}$ (where $P_{\#}$ is a configuration parameter); otherwise, the variable takes the corresponding value in e . The class of the new classifier is randomly selected among the classes that are not covered in $[M]$. The parameters of the new classifiers are set to initial values; typically, $p = 10$, $\epsilon = 0$, and $F = 0.01$. These parameters are initialized with a value close to zero to avoid an excessive influence of young classifiers in the selection and inference procedures; as long as these classifiers participate in action sets, the parameter update procedure adjusts their parameters to their real value. Besides, the numerosity is set to 1, the experience to 0, and the action set size to the size of the match set where the covering has been fired.

Next, the system computes the *system prediction* $P(c_i)$ for each possible class, which estimates the payoff that the system will receive if c_i is selected as output. $P(c_i)$ is calculated as the fitness weighted average of the predictions of the classifiers in $[M]$ that advocate class c_i ; that is:

$$P(c_i) = \frac{\sum_{cl.class=c_i \wedge cl \in [M]} cl.p \cdot cl.F}{\sum_{cl.class=c_i \wedge cl \in [M]} cl.F}, \quad (3.2)$$

where $cl.class$, $cl.p$, and $cl.F$ refer to the class, the reward prediction, and the fitness of the classifier respectively. Then, XCS selects one of the classes randomly. Thus, XCS explores the consequences of all classes for each possible input. Notice that other exploration regimes, such as giving each class c_i a selection probability proportional to P_{c_i} , could be applied as well. The chosen class determines the action set $[A]$, which consists of all classifiers advocating that class. The action set works as a *niche* where the parameters update procedure and the genetic algorithm take place. The next subsections explicate these two procedures in detail.

3.1.3 Classifier Evaluation

In training mode, after XCS sends the chosen class to the environment, a reward R is returned. R is maximal if the proposed class is the same as the training example (usually 1000), and minimal (usually zero) otherwise. Then, in single step problems, classifier parameters are updated with respect to the immediate reward in the current action set. As proceeds, we detail this parameter update procedure.

The prediction of each classifier cl is first updated according to the Widrow-Hoff rule (Widrow and Hoff, 1988) as

$$cl.p \leftarrow cl.p + \beta(R - cl.p), \quad (3.3)$$

where β ($0 < \beta \leq 1$) is the learning rate. The learning rate fixes the adaptivity of the parameters to the received rewards. That is, large values of β would produce large corrections in the prediction parameter each time the classifier participates in $[A]$, whilst low values of β will cause small corrections. A typical value for this parameter is $\beta = 0.2$ (Butz and Wilson, 2001). Next, the prediction error $cl.\epsilon$ is computed as

$$cl.\epsilon \leftarrow cl.\epsilon + \beta(|R - cl.p| - cl.\epsilon). \quad (3.4)$$

Then, the fitness is updated as follows. First, the *accuracy* $cl.\kappa$ of each classifier cl in $[A]$ is calculated as

$$cl.\kappa = \begin{cases} \alpha(cl.\epsilon/\epsilon_0)^{-\nu} & cl.\epsilon \geq \epsilon_0; \\ 1 & \text{otherwise.} \end{cases} \quad (3.5)$$

Note that $cl.\kappa$ is an inverse function of the error. The formula uses a power function with exponent ν (ν is a configuration parameter), enabling in this way to tune the pressure toward highly accurate classifiers; besides, when the classifier has a prediction error lower than the configuration parameter ϵ_0 , the system considers this classifier maximally accurate. The accuracy $cl.\kappa$ is used to compute the *relative accuracy* $cl.\kappa'$ as

$$cl.\kappa' = \frac{cl.\kappa \cdot cl.n}{\sum_{cl_i \in [A]} cl_i.\kappa \cdot cl_i.n}, \quad (3.6)$$

which reflects the relative accuracy of the classifier with respect to the other classifiers in the same action set. Thence, using this procedure, all the classifiers in the niche share the global resources of that niche. Then, $cl.\kappa'$ is employed to update the fitness as

$$cl.F = cl.F + \beta(cl.\kappa' - cl.F). \quad (3.7)$$

Thus, the fitness is an estimate of the accuracy of the classifier prediction relative to the accuracies of the overlapping classifiers. This provides fitness sharing among the classifiers belonging to the same action set. Finally, the action set size is updated as

$$cl.as = cl.as + \beta(|[A]| - cl.as), \quad (3.8)$$

where $|[A]|$ is the size of the current action set. At the end of this process, the experience of the classifier is incremented.

Classifier's parameters p , ϵ , and as are updated differently in the first iterations of XCS. That is, to let the classifier parameters move to quickly to their real values at the beginning of the classifier life, the *moyenne adaptive modifiée* technique (Venturini, 1994) is used. This technique sets the parameters of the classifiers directly to the average value computed with the instances that the classifier has matched. This process is applied while the experience of the classifier is less than $1/\beta$.

Once the parameters of the classifiers in $[A]$ have been evaluated, the GA can be applied to the current niche. The next section explains the genetic search in more detail.

3.1.4 Classifier Discovery

XCS uses a steady-state niche-based *genetic algorithm* (GA) (Goldberg, 1989a) to discover new promising classifiers. The GA is triggered in the current action set if the average time since its last application to the classifiers in [A] is greater than θ_{GA} . Here, we explain the basic mechanisms of the GA.

The GA selects two parents from the current [A] following either proportionate selection (Wilson, 1995) or tournament selection (Butz et al., 2005c) and copies them. Under proportionate selection, each classifier has a probability $p_{sel}(cl)$ to be selected proportional to its fitness; that is,

$$p_{sel}(cl) = \frac{cl.F}{\sum_{cl_i \in [A]} cl_i.F}. \quad (3.9)$$

Under tournament selection, a proportion of the action set, specified with the configuration parameter σ , is selected to participate in the tournament. The classifier with maximum fitness in the tournament is chosen.

The copies undergo crossover with probability χ and mutation with probability μ per allele. Two crossover schemes have been used for XCS: two-point crossover and uniform crossover. Two-point crossover copies the two parents into two offspring, selects two cut points in the offsprings, and swaps all the variables between the two points. Uniform crossover decides, for each variable, from which parent the information is copied. If crossover, is not applied, the offspring are exact copies of the parents. After this, mutation is applied as follows. For each input variable, the mutation operator randomly decides whether the variable needs to be changed. In this case, it randomly chooses a new value for the variable. The class of the rule also undergoes the same process.

The offspring parameters are initialized as follows. If crossover is not applied, the prediction, the error, and the fitness parameters are copied from the selected parent. Otherwise, these parameters are set to the average value between the corresponding parameters in the parents. In both cases, the fitness is decreased to 10% of the parental fitness. Experience and numerosity are initialized to 1.

The resulting offspring are introduced into the population via subsumption (Wilson, 1998). That is, if there exists a sufficiently experienced ($cl.exp > \theta_{sub}$) and accurate ($cl.\epsilon < \epsilon_0$) classifier cl in [A] whose condition is more general than the new offspring, the numerosity of this classifier is increased. Otherwise, the new offspring is introduced into the population. Two classifiers are removed if the population is full. The deletion probability of a classifier is proportional to the action set size estimate of the classifier; moreover, the deletion probability is increased if the classifier cl is experienced enough ($cl.exp > \theta_{del}$) and its fitness $cl.F$ is smaller than a proportion of the average fitness of the population \bar{F} ($cl.F < \delta\bar{F}$) (Kovacs, 1999). That is, the deletion probability p_{del} of a classifier cl is computed as

$$cl.p_{del} = \frac{cl.d}{\sum_{\forall cl_i \in [P]} cl_i.d}, \quad (3.10)$$

where

$$cl.d = \begin{cases} \frac{cl.n \cdot cl.as \cdot F_{[P]}}{cl.F} & \text{if } cl.exp > \theta_{del} \text{ and } cl.F < \delta F_{[P]}; \\ cl.as \cdot cl.n & \text{otherwise,} \end{cases} \quad (3.11)$$

where $F_{[P]}$ is the average fitness of the population. This deletion scheme biases the search toward highly fit classifiers and, at the same time, balances the classifiers' allocation in the different action sets.

Once the learning finishes, the evolved population is used to infer the class of new unlabeled instances. The next section provides the reasoning mechanism used by XCS.

3.1.5 Class Inference in Test Mode

The final solution of XCS consists of a set of rules with minimal error that cover all the problem space. This means that, in classification tasks, the system would evolve two types of rules: (1) rules with high prediction and low error and (2) rules with low prediction—thence, predicting the wrong class—and low error. For this reason, it is said that XCS evolves a *complete action map* (Wilson, 1995; Kovacs and Kerber, 2001). In test or exploitation mode, all the matching classifiers in the population are used to infer the class of a new input instance. The reasoning mechanism works as follows. Firstly, XCS creates [M] with all the matching classifiers; covering is not applied in any case. Then, the prediction array is formed as explained in section 3.1.2, and the most voted class is returned as output. Note that during test, the population is never modified.

In summary, XCS is an online system which represents individuals as classifiers that contain single rules, uses adapted reinforcement learning techniques to evaluate the quality of these classifiers, employs a steady-state niche-based GA to discover new promising rules, and applies a fitness-based voting policy to infer the class of test instances. XCS process organization is based on the activation of classifiers to form match sets and action sets, and on the application of the parameter update procedure and the GA on these action sets or niches. In the next subsection, we explain how this process leads to the evolution of accurate rule sets.

3.1.6 Why Does XCS Work?

After defining the learning process and the reasoning mechanism of XCS, we now intuitively explain the mechanisms that let XCS evolve a set of maximally general and accurate classifiers. For this purpose, we revise the work by Butz et al. (2004b), and explain the five evolutionary pressures identified by the authors that lead the system to evolving a set of optimal classifiers. The explanations are maintained in an abstract level, and the details of the mathematical formulation are not provided. The user is referred to (Butz et al., 2004b) for the details.

Butz et al. (2004b) identified five evolutionary pressures that guide the learning process in XCS:

1. The set pressure.
2. The mutation pressure.

3. The deletion pressure.
4. The subsumption pressure.
5. The fitness pressure.

In what follows, each item is briefly explained.

Set pressure. The *set pressure* is mainly due to the combination of the application of the GA in niches and the deletion in the whole population. This pressure was early explained by Wilson (1995), who formulated the following hypothesis: if two classifiers are equally accurate but have different generalizations, then the most general one will participate in more action sets, having more reproductive opportunities and finally displacing the specific classifier. This hypothesis was later investigated by Kovacs (1997), defining the *optimality hypothesis*, and formalized by Butz et al. (2004b). In brief, this supports the fact that the most general and accurate classifiers will take over their niches, displacing both over-general and most specific, accurate classifiers.

Mutation pressure. Whereas the set pressure moves the population toward generality and accuracy, Butz et al. (2004b) identified that mutation, in its own, causes the population to tend to more specific classifiers. That is, mutation changes the value of a variable, which can take one value from the ternary alphabet $\{0,1,\#\}$. As two of these values are specific, i.e., $\{0,1\}$, and the last one is general, mutation pushes toward a distribution of 66.7%/33.3% of specific/general bits. Obviously, the intensity of this pressure depends on the period of application of the GA and the mutation probability.

Deletion pressure. The population-wise deletion operator removes classifiers depending on their action set size estimate and their fitness. As classifiers that belong to large action sets are given a higher deletion probability, the operator makes pressure towards even distribution of classifiers in the different system niches. Furthermore, the deletion operator also pushes toward removing classifiers with low fitness, driving the search toward the fittest individuals.

Subsumption pressure. The subsumption pressure pushes towards generalization inside the niche. Once several accurate classifiers have been found, subsumption deletion causes the system to prefer the maximally general classifiers over the most specific ones. That is, GA subsumption checks, for each offspring, if there exists any accurate classifier in $[A]$ whose condition includes the offspring's condition; if so, the numerosity of this classifier is incremented. Therefore, subsumption produces an additional pressure toward generalization.

Fitness pressure. Finally, the fitness pressure is present in all the mechanisms of XCS, and influences the four aforementioned pressures as well. In general, fitness pressure pushes the population from over-general to more specific and accurate classifiers. It interacts with the other pressures since selection, mutation, and subsumption depend on classifier's fitness. In summary, the interaction of the five pressures drives the population toward a population of accurate maximally general classifiers.

With the explanation provided in this section we have covered the technical details and have glimpsed the ideas that explicate why XCS works. Notice that, in essence, XCS is a general

architecture—which evaluates rules online and uses a robust search mechanism to discover new promising rules—rather than a specific architecture particularly designed for solving a concrete set of tasks. For this reason, the learning architecture of XCS has been applied—sometimes with few modifications—to solve different types of problems. In the following section, we present one of this system modifications that specializes XCS to deal with data classification tasks more efficiently. The new architecture is addressed as the *supervised classifier system* (UCS).

3.2 The UCS Classifier System

UCS is an accuracy-based LCS that inherits the main components of XCS, but specializes the online learning architecture for classification tasks (Bernadó-Mansilla and Garrell, 2003). The aim of the system is to take advantage of having the class of the training instances, so focusing the exploration process toward classifiers that predict the correct class accurately. Therefore, UCS does not evolve a complete action map—including classifiers with low prediction and error—but it does create the *best action map*, which consists of a set of maximally general and accurate classifiers that predict the correct class. With this modification, UCS is expected to

1. evolve a solution quicker than XCS, since it only explores the correct class, and
2. require less population to store the solution, as it only needs to maintain the best action map.

Furthermore, UCS adapts the classifier’s parameters to supervised learning. As follows, we review the learning mechanism of UCS, especially focusing on the novelties with respect to the XCS architecture. Therefore, we first revisit the knowledge representation, which introduces new parameters to assess the quality of the rules. Then, we analyze the differences in the learning interaction, rule evaluation, rule discovery, and reasoning mechanism to infer the class of new input instances.

3.2.1 Knowledge Representation

As XCS, UCS evolves a population [P] of classifiers, where each classifier contains a rule and a set of parameters. The rule representation is copied from XCS. Therefore, rules consist of a condition that advocates a class. Besides, the classifiers have the following parameters:

1. The accuracy acc , which maintains an average of the proportion of matching examples that have been correctly classified by the rule.
2. The fitness F , which is computed as a function of the accuracy.
3. The correct set size cs , an estimate of the size of the correct sets in which the classifier has participated (see section 3.2.2).
4. The experience exp , which reckons the number of examples that the classifier has matched during his life.
5. The numerosity n , which indicates the number of copies of the classifier in the population.

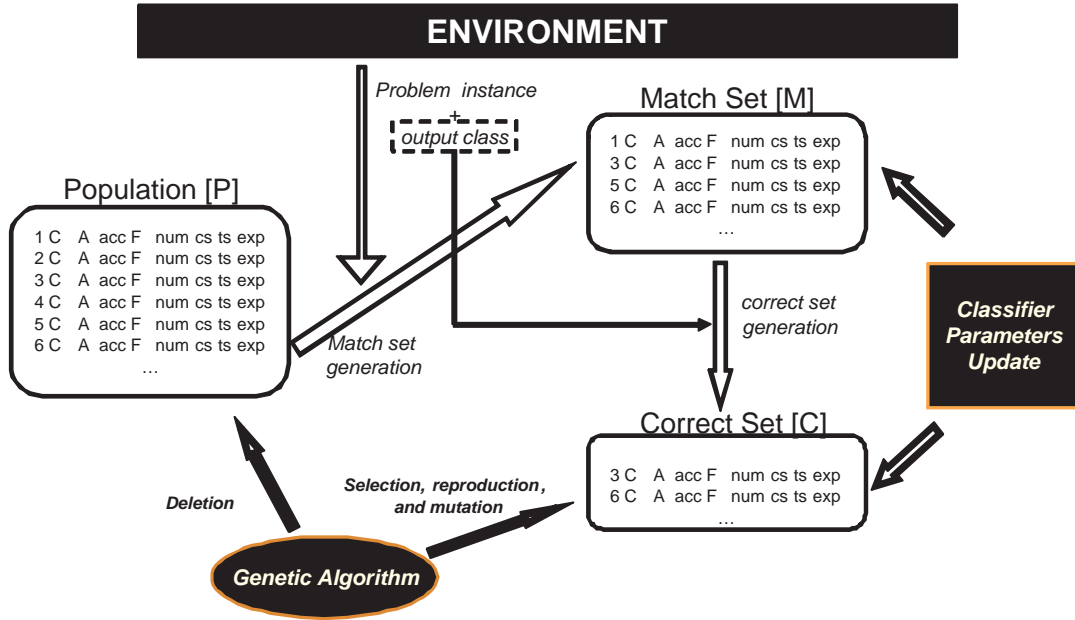


Figure 3.2: Schematic of the process organization of UCS.

Therefore, UCS inherits the experience and the numerosity parameters, redefines the accuracy and the fitness parameters, and introduces the correct set size parameter. Note that one of the key differences with respect to XCS is that, in UCS, fitness is based on accuracy, which is directly computed as the true proportion of correct predictions of the given rule. With the modification of the knowledge representation in mind, the next subsections describes the changes on the learning interaction proposed by UCS to adapt the online learning architecture to supervised learning tasks.

3.2.2 Learning Interaction

Figure 3.2 illustrates the process organization of UCS, which redefines the process organization of XCS according to a supervised learning scheme. The main difference with respect to XCS is that, in UCS, the class of each learning instance is provided by the environment. Therefore, the learning architecture takes advantage of this information to explore only the class of each sampled input example. As proceeds, this procedure is explicated in detail.

As XCS, UCS starts the *exploration phase* with an empty population. At each learning iteration the system receives a new input example e with its class c . Then, the system creates the *match set* $[M]$, which contains all the classifiers in the population $[P]$ whose condition matches e . Next, all the classifiers in $[M]$ that predict the class c form the *correct set* $[C]$. If $[C]$ is empty, the covering operator is activated, which creates a new classifier whose condition is generalized from e (as in XCS), and whose predicted class is set to c . Hence, covering aims at discovering a single classifier that predicts the sampled input instance correctly; besides, some generalization is added by setting each variable to ‘#’ with probability $P_{\#}$. The parameters of the new classifier

are set to: $exp = 1$, $num = 1$, $cs = 1$, $acc = 1$ and $F = 1$. As fitness and accuracy are estimated from a single instance—and so, the estimate may be poor—UCS does not let young classifiers have a strong participation in the genetic selection and the reasoning mechanism until they do not receive a minimum number of parameter updates.

After this, the parameters of all classifiers in the match set are evaluated, and eventually, the correct set—which defines a niche of classifiers with similar conditions and the same predicted class—receives a genetic event. Note that the parameter update procedure is applied to [M] instead of to [C]. UCS moves this procedure to the match set since the accuracy of all the matching classifiers that predict the input instance wrongly needs to be decreased. Also, notice that the correct set size acts as a niche where the genetic algorithm is applied, following the same idea of XCS. These two procedures are detailed in the following two subsections.

3.2.3 Classifier Evaluation

Each time a classifier participates in a match set, its experience, accuracy, and fitness are updated. Firstly, the experience is increased. Then, the accuracy is computed as the percentage of correct classifications:

$$cl.acc = \frac{\text{number of correct classifications}}{cl.exp}. \quad (3.12)$$

Thus, accuracy is a cumulative average of correct classifications over all matches of the classifier. Next, fitness is updated according to the following formula:

$$cl.F_{micro} = (cl.acc)^\nu, \quad (3.13)$$

where ν is a constant set by the user that determines the strength pressure toward accurate classifiers (a common value is 10). Thus, differently from XCS, fitness is calculated individually for each micro-classifier, and it is not shared. The fitness of a macro-classifier $cl.F_{macro}$ is obtained by

$$cl.F_{macro} = cl.F_{micro} \cdot cl.n. \quad (3.14)$$

Finally, each time the classifier participates in [C], the correct set size $cl.cs$ is updated. $cl.cs$ is computed as the arithmetic average of the sizes of the correct sets in which the classifier has taken part.

Once the parameters of the classifiers in [M] have been evaluated, the GA can be applied to the current correct set. The following subsections explain this process.

3.2.4 Classifier Discovery

UCS uses a steady-state niche-based GA as the primary search mechanism to discover new promising rules. The GA is applied to [C] following the same procedure as in XCS. Here, we briefly review this process and explain in more detail the rule deletion mechanism, which is slightly modified with respect to XCS's one.

The GA is triggered in the current correct set if the average time since its last application to the classifiers in [C] is greater than θ_{GA} . If so, the GA selects two parents from [C]. The

two selection schemes of XCS are also applicable here: proportionate selection and tournament selection. The only difference is that the fitness of young classifiers ($exp < \theta_{del}$) is divided by θ_{del} to avoid their influence in the selection process. Then, the two parents are copied, creating two new offspring, which are recombined and mutated with probabilities χ and μ respectively. The crossover and mutation operators are directly inherited from XCS (see section 3.1.4). The parameters of the offspring are initialized as follows. The experience and the numerosity are set to 1. The accuracy and the fitness are also set to 1 (these parameters will go quickly to their real values as they participate in successive match sets). Finally, cs is set to the size of the current correct set.

Finally, both offspring are introduced into the population. First, each offspring is checked for subsumption with the classifiers in [C]. The subsumption mechanism is adapted from XCS as follows: if one of the classifiers cl in [C] is sufficiently experienced ($cl.exp > \theta_{sub}$), accurate ($cl.acc > acc_0$) and more general than the offspring, then the offspring is not introduced into the population and the numerosity of the subsumer classifier is increased. If the offspring cannot be subsumed, it is inserted in the population, deleting another classifier if the population is full. The deletion probability p_{del} of a rule cl is calculated as:

$$cl.p_{del} = \frac{cl.d}{\sum_{\forall cl_i \in [P]} cl_i.d}, \quad (3.15)$$

where

$$cl_d = \begin{cases} \frac{cl.cs \cdot cl.n \cdot F_{[P]}}{cl.F_{micro}} & \text{if } cl.exp > \theta_{del} \text{ and } cl.F_{micro} < \delta F_{[P]}; \\ cl.cs \cdot cl.n & \text{otherwise,} \end{cases} \quad (3.16)$$

where δ and θ_{del} are configuration parameters, and $F_{[P]}$ is the average fitness of the population. In this way, deletion will tend to balance resources among the different correct sets, while removing low-fitness classifiers from the population. As fitness is computed from the proportion of correct classifications of a classifier, classifiers that predict wrong classes are not maintained in the population, and so, only the best action map is evolved.

The whole process is iterated during several learning steps in which a new instance is sampled and the processes of match set creation, parameter evaluation, and genetic algorithm application take place. After this, the system results in a population of highly general and accurate classifiers, which are used to infer the class of new input instances. The next subsection provides the reasoning mechanism implemented in UCS.

3.2.5 Class Inference in Test Mode

In test mode, a new input example e is provided, and UCS has to predict the associated class. For this purpose, UCS implements a reasoning mechanism which is similar to the one of XCS. That is, firstly, the match set is created. Then, all classifiers in [M] emit a vote, weighted by their fitness, for the class they predict. The vote of young classifiers (i.e., $exp < \theta_{del}$) is decreased by multiplying its vote by exp/θ_{del} to diminish their influence with respect to more experienced classifiers. The most-voted class is chosen. New inference schemes have been proposed in [Brown et al. \(2007\)](#), showing that the current inference schemes of XCS is one of the best among the compared ones. Under test mode, the population of UCS does not undergo any change; that is, all update and search mechanisms are disabled.

So far, we have detailed the key differences between the architectures of XCS and UCS. In essence, UCS follows the same mechanisms of XCS, but introduces some little modifications to take advantage of knowing the class of the training examples. With this new architecture, UCS is not expected to perform better than XCS, but to be able to evolve a solution spending less computational resources. That is, UCS needs to evolve and maintain a lower number of classifiers—since UCS does not evolve low rewarded rules—and is expected to solve the problem more quickly than XCS, since it only explores the “correct class”. Nonetheless, note that the key concepts of XCS, such as the accuracy-based approach, the incremental parameter update procedure, and the steady-state niche-based GA are still present in UCS. In the following subsection, we argue that the same ideas introduced in section 3.1.6 to explain why XCS works are still valid in UCS.

3.2.6 Why does UCS work?

As XCS, UCS is guided by the following five evolutionary pressures:

1. The set pressure.
2. The mutation pressure.
3. The deletion pressure.
4. The subsumption pressure.
5. The fitness pressure.

The main differences with respect to XCS is that, now, the fitness definition differs. Therefore, the system no longer pressures toward obtaining classifiers with low prediction error, but toward highly accurate classifiers. The consequences of this is that, as already discussed, UCS evolves the best action map instead of a complete action map. The other pressures do not need to be redefined as a consequence of the change in the architecture. That is, the set pressure pushes [P] toward the most general classifiers. The mutation pressure pushes toward specificity. The deletion pressure maintains an even distribution of classifiers in the different niches, giving more deletion probability to the classifiers with the lowest fitness with respect to the average fitness of the population. And finally, the subsumption pressure makes UCS prefer the most general classifiers that are still accurate to more specific, accurate classifiers. The overall interaction of these pressures, as in XCS, guides the search toward maximally general and accurate classifiers.

3.3 Rule Representations for LCSs

Thus far, we have described XCS and UCS with a ternary rule representation. During the last few years, several new rule representations have been introduced to XCS and UCS to let the systems deal with real-world problems such as interval-based representations (Wilson, 2000, 2001), hyper elipsoidal representations (Butz et al., 2006, 2008), and convex hulls (Lanzi and Wilson, 2006). Other more general approaches that have been used to codify the classifiers rules are neural networks (Bull and O’Hara, 2002), messy representations (Lanzi, 1999a) LISP

s-expressions (Lanzi and Perrucci, 1999), and gene expression program representations (Wilson, 2008). Besides, fuzzy representations have been designed for some Michigan-style LCSs (e.g., see Valenzuela-Rendón (1991)); fuzzy representations will be presented in detail in chapter 8. In here, we introduce one of the most-used representations to deal with continuous attributes in XCS, that is, the interval-based representation. As follows, we first provide some historical remarks about the different proposals of interval-based rule representation for LCSs, and introduce the one used in the present work.

3.3.1 From the Ternary to the Interval-based Rule Representation in LCSs

Initially designed with a ternary representation, LCSs faced a new challenge when dealing with continuous or quantitative attributes, which are often present in real-world problems. That is, the ternary rule representation was not suitable for directly dealing with continuous data. Data preprocessing techniques could be used to transform the continuous values into discrete values; nonetheless, this could result in an undesirable loss of information. Recently, interval-based rule representations have been designed to effectively deal with continuous attributes. The most significant of these representations are reviewed as follows.

Wilson (2000) designed one of the first interval-based representation for XCS, addressed as *center-spread representation*. The center-spread representation codifies each rule variable with a pair of values (c_i, s_i) that defines a rectangle with center in c_i and spread s_i . Besides, the representation has to satisfy the constraint that $c_i - s_i/2 \geq \max_i$ and $c_i + s_i/2 \leq \min_i$, where \max_i and \min_i are respectively the maximum and the minimum values that the attribute can take.

This representation empirically demonstrated to be able to evolve accurate models in artificial problems with continuous attributes. Nevertheless, the truncation caused by guaranteeing the constrain on the maximum and the minimum values might result in an inefficient exploration of the feature space, as later shown by Stone and Bull (2003). To overcome this problem, Wilson (2001) presented another interval-based representation, referred to as *min-max representation* in which each attribute codifies the lower ℓ_i and the upper u_i limit of the interval of values where the attribute is applicable. Although the effects of truncation are not present, this representation still has the problem of invalid intervals eventually caused by the genetic operators. That is, genetic operators may generate intervals where $\ell_i > u_i$, in which the classifier would not match any input instance. This situation could be fixed in several ways by modifying the value of the interval bounds.

A simple approach to deal with this effect was proposed by Stone and Bull (2003), who introduced the *unordered-bound representation*. The unordered-bound representation proposes to use the min-max representation but without prefixing which of the two bounds are the upper bound and the lower bound. That is, the unordered-bound representation codifies each variable as an interval (p_i, q_i) ; the smaller of these two values is considered as the lower bound of the interval and the larger value is considered the upper bound. This has been, probably, the most used representation for continuous attributes in the last few years.

New efforts in the definition of representations for continuous attributes were made after the presentation of the unordered-bound representation. For example, Dam et al. (2005) argued that the unordered-bound representation produces large changes in the semantics of the intervals

when an operator exchanges the lower bound with the upper bound of an interval. Therefore, this may thwart the correct propagation of the building blocks of the problem. In order to solve this, Dam et al. (2005) proposed to represent an attribute with the pair (m_i, p_i) , where m_i is the lower bound of the interval and p_i is a proportion used to compute the length of the interval s_i as

$$s_i = p_i(p_{max} - m_i), \quad (3.17)$$

in which p_{max} is the maximum value that the attribute i can take. Thus, the pair (m_i, p_i) can easily be translated to the lower bound ℓ_i and upper bound u_i of the interval by recognizing that

$$\ell_i = m_i, \quad (3.18)$$

$$u_i = m_i + s_i \quad (3.19)$$

However, the empirical results did not clearly show an improvement with respect to the unordered-bound representation. For this reason, we used the unordered-bound representation in the present work. The next section provides more details about this representation.

3.3.2 The Unordered Bound Representation

We now describe the unordered-bound representation in more detail and explain how the genetic operators were adapted to deal with the new representation. As aforementioned, the new representation codifies each variable with an interval (p_i, q_i) , where the minimum value between the two bounds represents the lower bound, and the maximum value represents the upper bound. For example, a classifier whose condition is defined by the two variables $\langle [1,2], [10,8] \rangle$ matches any input instance whose first variable ranges in $[1,2]$ and its second variable ranges in $[8,10]$. As proceeds, we explain how the covering operator, the different genetic operators that deal with the representation—i.e., crossover and mutation—, and the subsumption operator are redefined to deal with the new representation. For simplicity, we assume that all the input attributes have been normalized, and so, that their values range in $[0,1]$.

Covering Operator

The covering operator creates a new classifier whose condition is generalized from the input example e . For this purpose, the interval of each variable i of the new classifier is initialized as

$$p_i = e_i - rand(0, r_0) \quad \text{and} \quad (3.20)$$

$$q_i = e_i + rand(0, r_0), \quad (3.21)$$

where r_0 is a configuration parameter ($0 < r_0 \leq 1$), and $rand(0, r_0)$ returns a random number between 0 and r_0 . Therefore, this operator creates an interval that includes the value of the corresponding attribute, and r_0 controls the generalization in the initial population (it is equivalent to $P_{\#}$ in the ternary representation). An example of covering is graphically illustrated in figure 3.3.

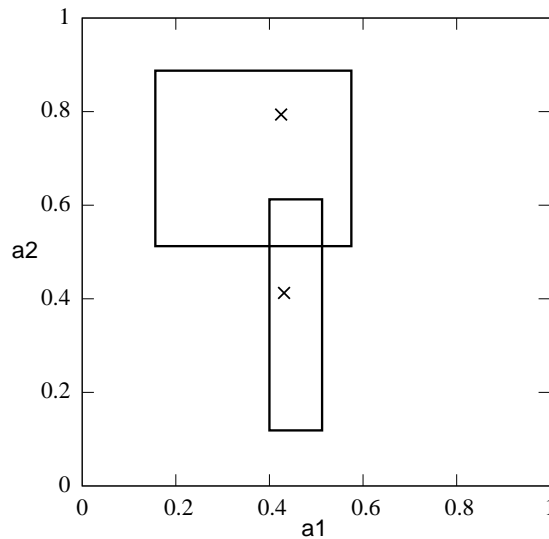


Figure 3.3: Example of covering in the hyper rectangular representation.

$$\begin{array}{ccc}
 \langle [0.20, 0.80], [0.45, 0.65] \rangle & & \langle [0.20, 0.85], [0.25, 0.65] \rangle \\
 & \implies & \\
 \langle [0.60, 0.85], [0.25, 0.75] \rangle & & \langle [0.60, 0.80], [0.45, 0.75] \rangle
 \end{array}$$

Table 3.1: Example of two-point crossover, in which the two cut points are in the middle of each interval.

Crossover Operator

In real-world problems, two-point crossover is usually applied. It randomly selects two cut points, which can occur either between or within an interval predicate. Then, the offspring are created by shuffling the information of both parents. The process is detailed in the example of table 3.1, in which the two parents are crossed, selecting a cut point in the middle of each interval and generating two new offspring. Figure 3.4 visually illustrates this example in the feature space.

Mutation Operator

The mutation operator is applied to each of the bounds of the interval. If it decides to change an interval bound, this is mutated by adding a random value that ranges in $(-m_0, m_0)$, where m_0 is a configuration parameter. Figure 3.5 shows an example of the mutation operator.

Subsumption Operator

To consider a classifier as a candidate subsumer, the same conditions defined in the ternary representation need to be satisfied. That is, the subsumer classifier has to be experienced

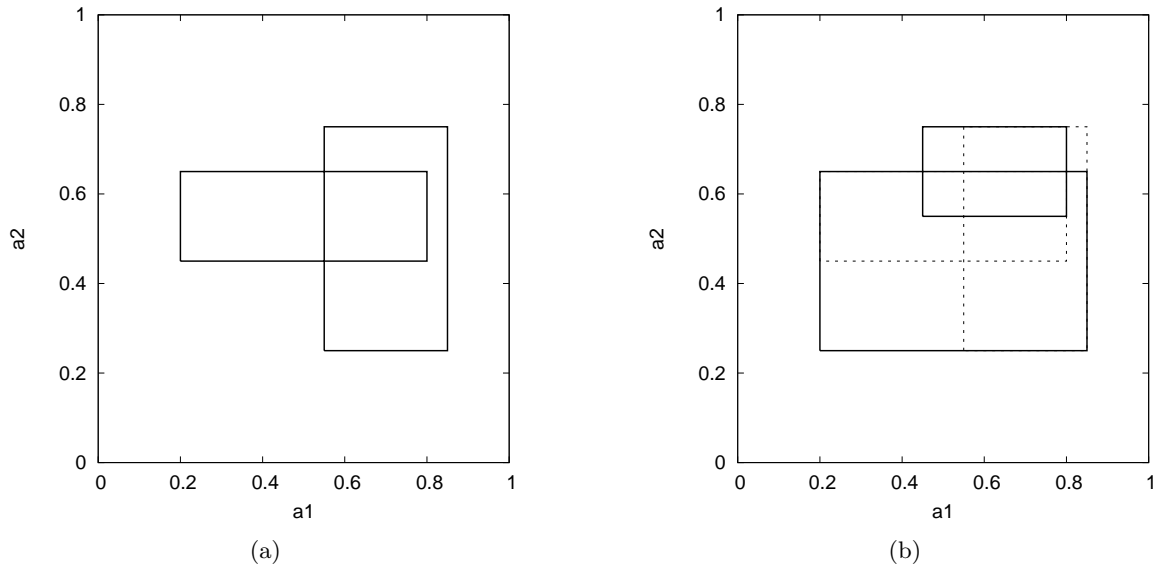


Figure 3.4: A crossover example. (a) plots the two parents and (b) shows the offspring resulting from two cut points occurring in the middle of each interval.

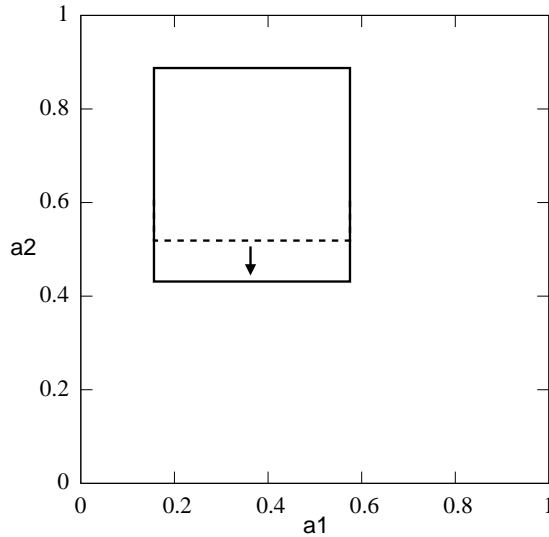


Figure 3.5: Example mutation in the hyper rectangle representation.

enough, accurate, and its condition has to include the condition of the subsumed classifier. For an interval-based representation, the condition of rule cl_1 includes that of rule cl_2 if for each variable i : $cl_1.l_i \leq cl_2.l_i$ and $cl_1.u_i \leq cl_2.u_i$, where l_i and u_i are respectively the smaller and greater value of the two bounds of the interval.

3.4 Summary and Conclusions

This chapter provided concise descriptions of XCS and UCS, which can be utilized as implementation guidelines. Besides, we explained the evolutionary pressures that guide both systems to evolve a minimal set of maximally general and accurate classifiers. Finally, we presented the interval-based representation used by the two LCSs to deal with continuous attributes.

During the explanation, we highlighted the differences between XCS and UCS, which are due to the specialization of UCS for supervised learning. Basically, the two key aspects that UCS modifies with respect to XCS are:

1. The fitness computation.
2. The exploration scheme.

The fitness computation procedure of UCS introduces two changes with respect to the one in XCS: (1) UCS computes the fitness based on the rule's accuracy on classifying the input instances instead of on the accuracy of the rule's prediction and (2) UCS does not share the fitness among the rules in the same niche. The former modification makes UCS push toward rules that predict all the matching instances correctly; therefore, the system evolves the best action map. Conversely, XCS evolves rules that are accurate in the prediction reward, regardless of whether they predict the correct class; thence, XCS builds the complete action map. Therefore, UCS spends less computational resources with respect to XCS since UCS only evolves and maintains the rules that predict the correct class. On the other hand, UCS does not use a fitness-sharing scheme; instead, it computes the fitness as a power of the raw accuracy. The advantages of not sharing fitness, if neither, are not clear and demand further investigation.

The exploration of UCS was also modified to speed up the convergence in classification problems. That is, as long as the system only needs to evolve the best action map and as the class is made available with each input example, UCS only explores the correct class. Hence, in addition to using less computational resources to store the final population, the system is also expected to obtain the optimal solution more quickly than XCS.

In the next chapter, we further investigate the impact of the architectural changes proposed by UCS. We first carefully study the effect of not sharing fitness in UCS. For this purpose, we design a fitness-sharing scheme, similar to the XCS's one, and incorporate it to UCS. Then, we empirically analyze the advantages of the new fitness computation method. Thereafter, we examine the behavior of UCS with respect to the XCS one in a collection of boundedly difficult problems. Thus, we include XCS in the comparison and analyze how the new fitness computation and the new exploration scheme of UCS affects the learning performance and the convergence to the optimal population.

Chapter 4

Revisiting UCS: Fitness Sharing and Comparison with XCS

The previous chapter explained the mechanisms of XCS and UCS in detail and reviewed the most important differences in the architecture of both systems. In brief, UCS performs a more focused search since its architecture is specialized for supervised learning tasks. The two main differences introduced by UCS affected (1) the fitness computation and (2) the exploration regime. With these changes, UCS was expected to evolve an accurate solution spending less computational resources than XCS¹. These initial hypotheses were already supported by the experimental results provided by [Bernadó-Mansilla and Garrell \(2003\)](#). In there, the authors compared the original scheme of UCS with XCS on a collection of three *boundedly difficult problems*. The experimental results enabled the authors to emphasize the differences between both architectures and to demonstrate that, in general, UCS could solve these problems more quickly than XCS. Although these were promising results, there were still some open issues that needed to be addressed. The most important aspect in this list is *fitness sharing*. That is, the lack of fitness sharing in UCS was identified as a potential weakness of the system. Nonetheless, no further study about the effect of not sharing fitness has been conducted so far.

The purpose of this chapter is to study whether fitness sharing can provide benefits to UCS and further compare UCS with XCS in an extension of the collection of boundedly difficult problems used by [Bernadó-Mansilla and Garrell \(2003\)](#). To achieve this, we design a fitness-sharing scheme which is inspired by the XCS's one. Then, we test the three systems on four boundedly difficult problems. The experimental results illustrate the benefits of fitness sharing and highlight how the modifications introduced in the learning architecture of UCS affect the system's behavior with respect to the XCS's one in classification problems.

The remainder of this chapter is organized as follows. In section 4.1, we introduce the concept of fitness sharing and motivate its use in UCS by highlighting its importance in the GAs and LCSs realms. Section 4.2 specifies the new fitness-sharing scheme for UCS and section 4.3 draws the analysis methodology. Section 4.4 compares UCS with fitness sharing with the original UCS, showing the benefits of the fitness-sharing scheme. Section 4.6 extends the comparison by analyzing the differences between UCS and XCS. Finally, section 4.6 gathers the key observations

¹Note that UCS can only be applied to supervised learning, while XCS is a much broader architecture that can be used for both supervised learning and reinforcement learning in general

drawn from the experiments, and section 4.7 summarizes and concludes this chapter. Appendix A supplies a full explanation of all the problems used along the comparison.

4.1 Fitness Sharing in GAs and LCSs

Before proceeding with the description of the fitness-sharing scheme designed for UCS, we first introduce the concept of fitness sharing and how this has been used in both GAs and LCSs. Holland (1975, 1992) initially presented the concept of fitness sharing as a way to accomplish *niching*. In his early work, Holland discussed the concept of limiting the number of individuals that occupy a niche. The underlying idea is that if each niche had associated a particular payoff or objective fitness, and if each individual in this niche were forced to share this payoff with the other individuals in the same niche, then a stable situation, where each niche contains approximately the same number of individuals, would arise. Therefore, niching methods that distribute the payoff among the individuals of the same niche are addressed as *fitness-sharing* methods.

In the GA field, fitness sharing has been used to maintain and evolve diverse solutions in multi-modal optimization problems. Goldberg and Richardson (1987) initially introduced the concept of explicit fitness sharing into GAs to optimize multi-modal functions. The proposed scheme derated the fitness of an individual by an amount related to the number of similar individuals in the population. More specifically, the method computed the *shared fitness* f_s^i of an individual i , whose objective fitness is f^i as

$$f_s^i = \frac{f^i}{\sum_{j=1}^n sh(d(i, j))}, \quad (4.1)$$

where sh is a function of the distance d between two solutions. sh returns ‘1’ if the elements are equal, and ‘0’ if they exceed some threshold of dissimilarity, σ_{share} . That is, if the distance between two solutions is greater than σ_{share} , $sh = 0$, indicating that none of the two individuals affects the fitness of the other. One of the most usual sharing functions is:

$$sh(d) = \begin{cases} 1 - \left(\frac{d}{\sigma_{share}}\right)^\alpha & \text{if } d < \sigma_{share}, \\ 0 & \text{otherwise;} \end{cases}$$

where α is a configuration parameter that permits regulation of the proximity of the solutions that are considered to be in the same niche. Further analysis investigated other sharing schemes and their capacities to solve multi-modal problems. For example, Deb (1989) empirically demonstrated that GAs with fitness sharing were able to solve a large variety of multi-modal functions. Theoretical models of the effect of different types of sharing can be found in (Mahfoud, 1995) and (Horn, 1997).

Fitness sharing has also been used in LCSs. Smith and Valenzuela-Rendón (1989) modeled a generational GA with infinite population size for LCSs and showed that fitness sharing was necessary to facilitate the coverage in difficult problems. In fact, the majority of new LCS designs present some form of niching or fitness sharing. This is the case of XCS. As seen in the previous chapter, XCS intrinsically performs niching by grouping the classifiers in action

sets. The system also incorporates fitness sharing since the classifier’s fitness depends on the classifier’s relative accuracy, and the relative accuracy of each classifier is computed with respect to the accuracies of all the other classifiers in the same niche (see section 3.1.3). Some more recent LCSs such as XCSF (Wilson, 2002b; Butz et al., 2008) also adopt the same, or a similar, fitness-sharing scheme.

Although fitness-sharing schemes have been shown to be beneficial to Michigan-style LCSs, UCS was originally designed without a fitness-sharing scheme. Fitness sharing was not incorporated in UCS to keep the initial architecture as simple as possible, which would enable a more detailed analysis of the system. Nonetheless, the combination of the niche-based GA application with the lack of resource sharing in UCS seems to be counterintuitive since both approaches have historically come tied together. Therefore, in this chapter, we provide a fitness-sharing scheme to UCS and empirically analyze its advantages over the original parameter update procedure. The next section introduces fitness-sharing scheme, similar to that of XCS, for UCS. Then, we present the experimental methodology, run UCS with both fitness sharing and the original parameter update procedure, and carefully analyze the results.

4.2 A New Fitness Sharing Scheme for UCS

XCS intrinsically performs niching by grouping similar classifiers in action sets. The niching is considered (1) by the GA, which selects and crosses classifiers from the same niche, and (2) by the parameter update procedure, which shares the fitness among all classifiers in the same niche. As does XCS, UCS performs niching in the first sense because it applies the GA to the niches. Nevertheless, in the original UCS, the resources are not shared among the classifiers in the same niche. Therefore, here, we propose to incorporate the resource sharing in UCS. As follows, a new fitness-sharing scheme for UCS, which is inspired by the one of XCS, is presented, reviewing how all the parameter update procedure works after introducing the new fitness computation. For the sake of clarity, in the remainder of the chapter, UCS without sharing is referred to as UCSns, and UCS with sharing is addressed as UCSs.

The new parameter update procedure works as follows. The experience (*exp*), the correct set size (*cs*), and the accuracy (*acc*) parameters are computed as in UCSns (see section 3.2). However, fitness is shared among all classifiers in [M]. First, a new accuracy *cl.k* is calculated, which discriminates between accurate and inaccurate classifiers. For classifiers belonging to [M], but not to [C], the accuracy is set to zero; that is, $\forall cl \in ![C] cl.k = 0$. For each classifier *cl* belonging to [C], *cl.k* is computed as follows:

$$cl.k = \begin{cases} 1 & \text{if } cl.acc > acc_0; \\ \alpha \left(\frac{cl.acc}{acc_0} \right)^\nu & \text{otherwise.} \end{cases}$$

Then, the relative accuracy *cl.k'* is calculated as

$$cl.k' = \frac{cl.k \cdot cl.n}{\sum_{cl_i \in [M]} cl_i.k \cdot cl_i.num}, \quad (4.2)$$

and the fitness is updated from *cl.k'*:

$$cl.F = cl.F + \beta(cl.k' - cl.F). \quad (4.3)$$

Let us note that, under this scheme, the computed fitness corresponds to the macro-classifier fitness, as the numerosities of the classifiers are involved in computation of the relative accuracy. Whenever the micro-classifier’s fitness is needed, we divide this value by the numerosity of the classifier.

This parameter update procedure is incorporated into UCSs, replacing the traditional scheme. In the next sections, we empirically analyze the advantages provided by this new scheme.

4.3 Methodology

The experimental analysis consists of two separate parts. First, we compare UCSns with UCSs, focusing on the advantages provided by the fitness-sharing scheme. Then, XCS is compared with UCS with the aim of highlighting the practical impact caused by the architectural changes introduced by UCS. In brief, UCS introduces (1) a new fitness computation which is based on the classification accuracy instead of on the accuracy of the prediction and (2) a new exploration scheme, which only explores the class of the input examples and maintains the best action map instead of the complete action map (see chapter 3). Therefore, we aim at analyzing how these two modifications influence the learning process in classification problems. The methodology used in both comparisons is detailed in what follows.

To analyze the behavior of the three systems, we took a systematic approach and compared their performance on four artificial problems that gather some complexity factors said to affect the performance of LCSs (Kovacs and Kerber, 2001; Bernadó-Mansilla and Garrell, 2003): a) a binary-class problem, the *parity*; b) a multi-class problem, the *decoder*; c) an imbalanced multi-class problem, the *position*; and d) a noisy problem, the *multiplexer with alternating noise*. Given a binary input of length ℓ , with k relevant bits ($k \leq \ell$), these problems are defined as follows. The parity problem returns the number of one-valued bits modulo two. The decoder problem gives the decimal value of the input as output. The position problem returns the position of the left-most one-valued bit; if the input does not contain any one-valued bit, it returns zero. The multiplexer problem takes the first $\log_2 \ell$ bits as the address bits, and the remaining bits as the position bits; then, the output is the value of the position bit referred by the decimal value of the address bits. We added noise to the training instances of the multiplexer problem by flipping the class of the input instances with probability P_x ; the test instances are free of noise. For a detailed description of each problem the reader is referred to appendix A. The parity, the decoder, and the position problems were configured with input lengths ranging from $\ell = 3$ to $\ell = 9$. The multiplexer was configured with 20 input bits. These boundedly-difficult problems permit varying the complexity along different dimensions such as input length, size of the optimal population, specificity of the optimal classifiers, number of classes, and class imbalance ratio. For further information, the reader is referred to appendix A. UCSns, UCSs, and XCS were run with each problem.

We configured the systems as follows. We used a standard configuration for XCS: $P_{\#} = 0.6$, $\beta = 0.2$, $\alpha = 0.1$, $\nu = 5$, $\theta_{GA} = 25$, $\chi = 0.8$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$. In UCS, the parameters that are shared with XCS took the same values, except for $\nu = 10$; additionally $acc_0 = 0.999$. In both cases, tournament selection was applied. Subsumption was activated in the genetic algorithm with $\theta_{sub} = 20$. The maximum population size of XCS and UCS was configured depending on the size of the optimal population that the systems were expected to

evolve. As explained in section 3.1, XCS evolves a complete action map [O], which consists of all rules with low error, regardless of whether they have high or low reward prediction. On the other hand, UCS evolves the best action map [B], which includes only highly rewarded rules. As proposed by Bernadó-Mansilla and Garrell (2003), we set population sizes to $N = 25 \cdot |[O]|$ in XCS and to $N = 25 \cdot |[B]|$ in UCS. All the results are averages of 25 runs with different random seeds.

We decided against using the training accuracy to evaluate the performance of the three LCSs since it does not provide enough evidence of effective genetic search, as highlighted by Kovacs and Kerber (2004). Instead of accuracy, the achieved proportion of the optimal action map %[O] was proposed by Kovacs and Kerber (2001) as being a better indicator of the progress of the genetic search. However, UCS and XCS evolve different optimal populations: XCS creates a complete action map, whereas UCS represents a best action map. To allow a fair comparison, we only consider the proportion of best action map %[B] achieved by each system. That is, we only count the proportion of consistently correct rules. To review the best action map of each of the four problems, the reader is referred to appendix A.

When required, we used statistical tests to compare the convergence curves between pairs of algorithms. As our aim was to compare pairs of learning systems, we used the non-parametric Wilcoxon signed-ranks test (Wilcoxon, 1945), which was fed with the performance measures taken during the learning process. For further details on this statistical test, the reader is referred to appendix B.

Having described the experimental methodology, we are now in position to analyze the results obtained on the four problems. The next section starts with the comparison of UCSns and UCSs. Later, XCS is introduced in the comparison.

4.4 Analyzing the Fitness Sharing Scheme in UCS

Figures 4.1 and 4.2 depict the proportion of the best action map %[B] achieved by UCSns and UCSs in the parity, the decoder, the position, and the 20-bit multiplexer with alternating noise problems. The results show that UCSs could discover the optimal population more quickly than UCSns in the parity, the decoder, and the position problems. These differences were significant in the decoder for $\ell > 4$, in the position for $\ell > 3$, and in the parity with any input length according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. Oppositely, these benefits could not be observed in the multiplexer with the highest levels of noise; that is, for $P_x = \{0.10, 0.15\}$, UCSns significantly outperformed UCSs. As proceeds, we examine these two different behaviors in more detail.

Benefits of fitness sharing. The improvement provided by the fitness-sharing scheme in the parity, the decoder, and the position problem—especially for the largest input lengths—can be explained as follows. Under fitness sharing, the progressive discovery of more accurate classifiers makes the fitness of less accurate, over-general classifiers that participate in the same correct sets decrease quickly. That is, when a better classifier is discovered, it gets a higher proportion of the shared fitness, causing a decrease in the fitness of less accurate classifiers that exist in the same niche. Therefore, fitness-sharing produces (1) a higher pressure toward the deletion of over-general classifiers and (2) a higher selective pressure toward the most accurate

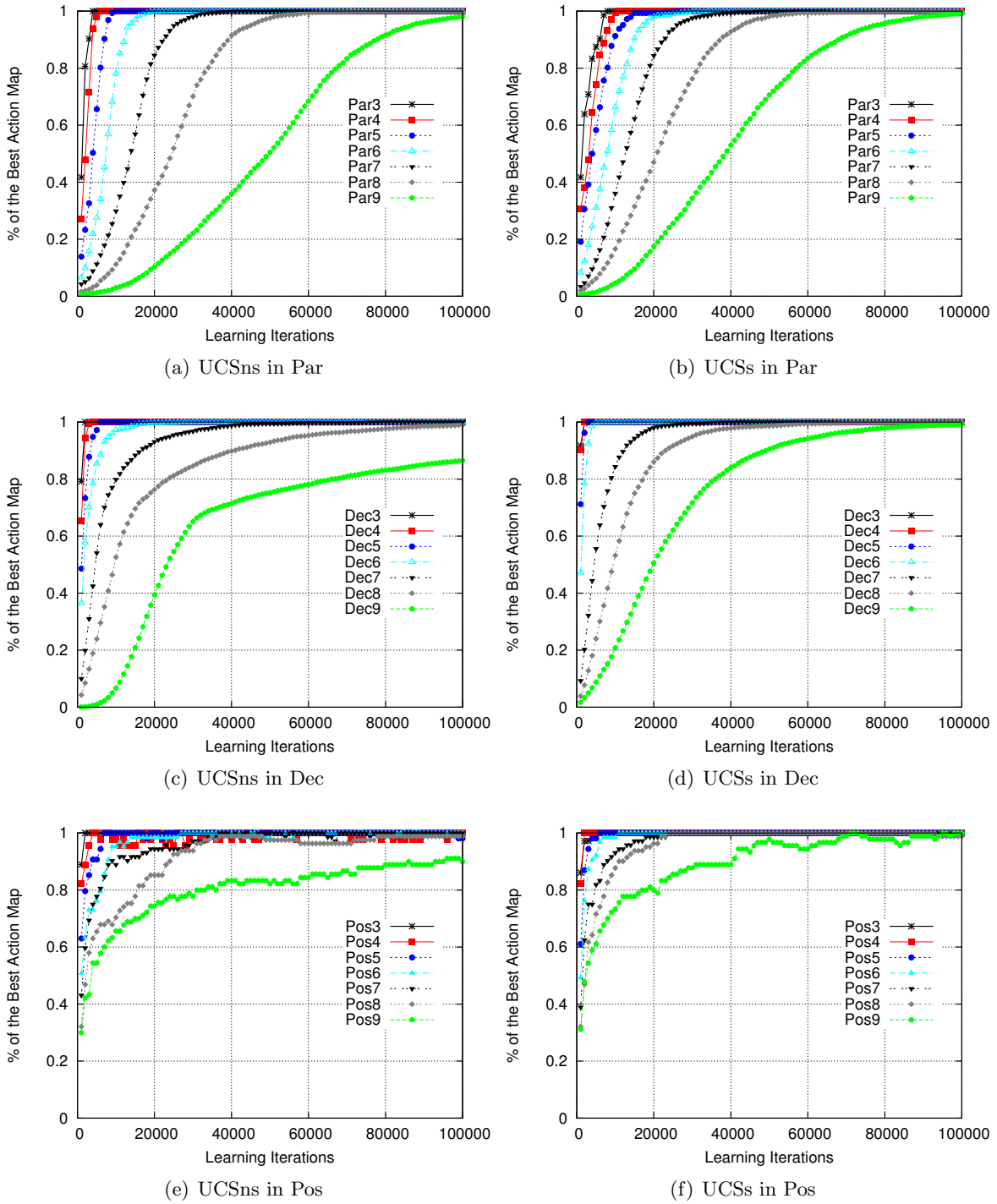


Figure 4.1: Proportion of the best action map achieved by UCSNs and UCSs in the parity, the position, and the decoder problems.

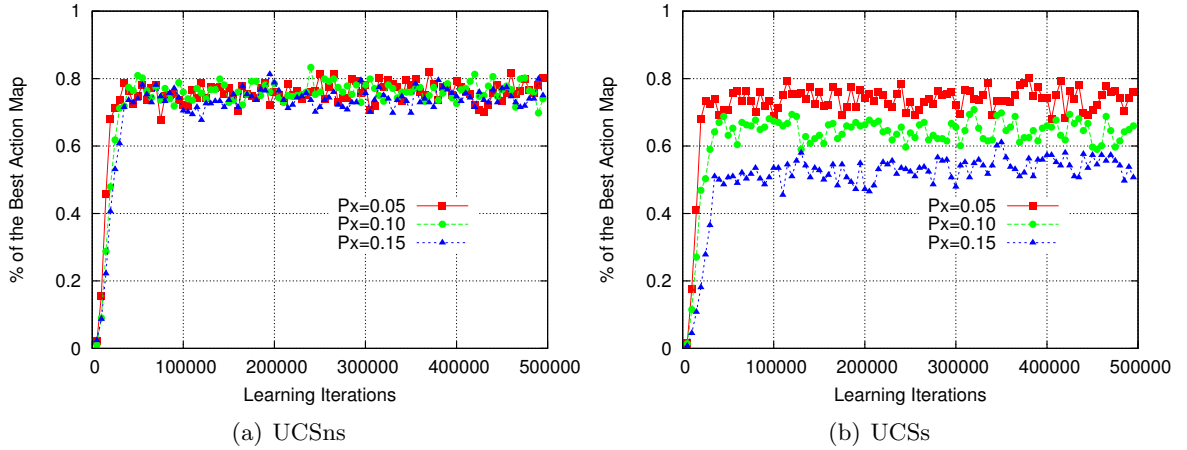


Figure 4.2: Proportion of the best action map achieved by (a) UCSns and (b) UCSs in the noisy 20-bit multiplexer with $P_x = \{0.05, 0.10, 0.15\}$.

classifiers in the GA. Without fitness sharing, over-general classifiers maintain the same fitness along the whole learning process.

The effect of fitness sharing was especially beneficial in the decoder and the position problems, while it was more modest in the parity problem. To explain the excellent results in the two former problems, let us first focus on the position problem and then extend the conclusions to the decoder problem. In the position problem, the system has to evolve a best action map that consists of classifiers with different degrees of generality ranging from a classifier in which all bits except one are set to ‘#’ to a classifier with all specific bits (see section A.3). Note that UCS with fitness sharing could solve the position problem for all the tested input lengths, while UCS without fitness sharing was not able to discover the complete action map for $\ell = 9$. A more detailed analysis of the results showed that the final populations of UCSns did not contain the most specific optimal classifiers. Moreover, it was also identified that, even though the most specific optimal classifiers were created in some of the runs, they were lost during the genetic search. This behavior was a result of the combination of the occurrence-based reproduction with the fitness computation without sharing. To illustrate this, let us assume the case that the system discovers the most specific classifier for $\ell = 9$, that is, 000000000:0. This classifier is activated once every 2^9 sampled instances. Once activated, this classifier may compete with a set of over-general classifiers. If fitness is not shared, the fitness of over-general classifiers remains constant regardless of whether the niche contains a high proportion of over-general classifiers or not. Therefore, as the number of over-general classifiers increases, it is more likely that the selection procedure chooses one of these over-general classifiers. This, coupled with the low activation rate of the most specific optimal classifiers, and so, their low reproductive opportunities, discourages their evolution. In these cases, the fitness-sharing scheme plays a key role to decrease the fitness of competing over-general classifiers, and so, promotes the maintenance of the most specific optimal classifiers.

These conclusions can be extended to the decoder problem. In this problem, all the optimal classifiers are maximally specific, and so, fitness sharing is necessary to promote the selection

Table 4.1: Accuracy and fitness of UCSNs’s classifiers along the generality-specificity dimension, depicted for the parity problem with $\ell = 4$.

	Condition	Class	Accuracy	Fitness
1	####	0	0.5	0.00097
2	0###	0	0.5	0.00097
3	00##	0	0.5	0.00097
4	000#	0	0.5	0.00097
5	0000	0	1	1

of these specific classifiers. In general, fitness sharing appears to be crucial in problems where some of the optimal classifiers are activated with a low frequency. This is the case in problems that (1) contain class imbalances or (2) permit little generalization.

On the other hand, it is worth noting that the improvement in the parity problem was not as accentuated as the one in the decoder and position problems. This is due to the lack of *fitness guidance* toward optimal classifiers in this particular problem. That is, the best action map of the parity problem consists of classifiers in which all the variables are specific. Therefore, no generalization is allowed in the optimal population (see section A.1 for further details). Nonetheless, at the beginning of the run, the covering operator introduces generalization in the initial population, and UCS has to drive the population from over-generality to optimal classifiers. However, the fitness pressure does not correctly lead to specificity. That is, specifying one bit of an over-general classifier does not increase its accuracy unless all bits are specific. Therefore, although approaching the optimum, all the classifiers in the niche receive the same amount of resources. Only when an optimal classifier is discovered, its accuracy is set to 1, and so, the fitness of all the over-general classifiers that participate in the same niche is decreased. Differently, in the decoder and the position problem, the accuracy guided to the optimal classifiers; for this reason, the improvement in the convergence time provided by fitness sharing was larger in these two problems.

To further illustrate the lack of fitness guidance in the parity problem, table 4.1 shows the evolution that the most over-general classifier needs to go through to become an optimal classifier in UCSNs for the problem with four input bits. At each step, one of the *don’t care* bits is specified. Note that accuracy, and so, fitness, remain constant during all the process until the optimal classifier is achieved. However, we would expect that the specification of one bit should result in a fitter classifier, since it approaches the optimum classifier (in which all the bits are specific). Therefore, the fitness is not guiding toward optimal solutions. This problem is a type of *needle-in-a-haystack* problem, in which an optimal classifier can only be obtained randomly. For this reason, the improvement provided by fitness sharing, although existing and being statistically significant, is not as strong as the one in the decoder and the position problems.

Fitness sharing in noisy problems. Figure 4.2 shows that UCSNs achieved higher performance than UCSs in the multiplexer problem, especially for the highest levels of noise. In particular, UCSNs significantly outperformed UCSs for $P_x = \{0.10, 0.15\}$ according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. This behavior cannot be directly attributed to the fitness-sharing

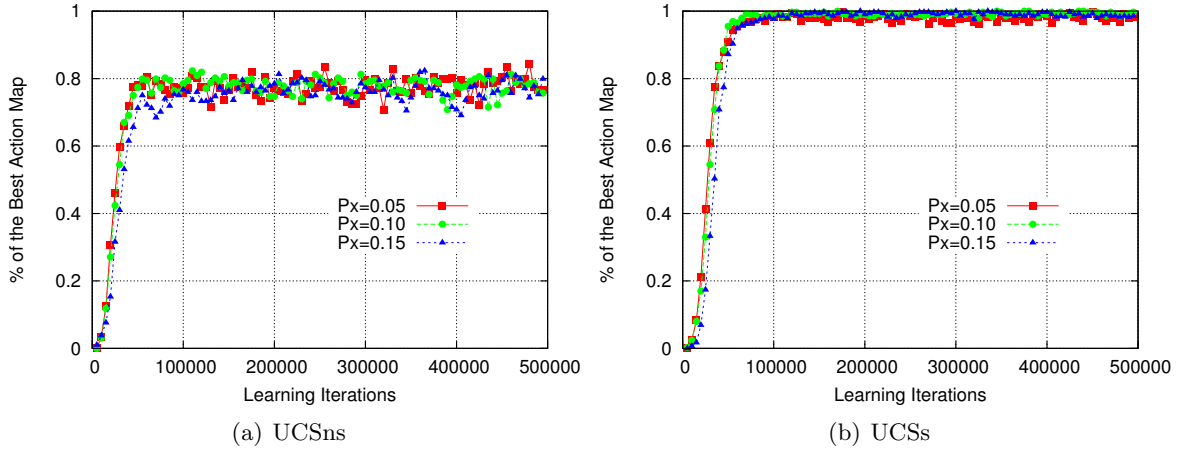


Figure 4.3: Proportion of the best action map achieved by (a) UCSns and (b) UCSs in the noisy 20-bit multiplexer with $P_x = \{0.05, 0.10, 0.15\}$ and using $\beta = 0.01$ and $\theta_{GA} = 100$.

scheme, but to the fact that, with the new parameter computation, UCSs calculates a windowed average of fitness by means of the learning parameter β . In noisy environments, the parameter averages oscillate and cannot stabilize properly, especially with the value of β employed in this configuration (i.e., $\beta = 0.2$). So, high levels of noise require low values of β . As UCSns computes fitness as a power of the accuracy, the parameter values of experienced classifiers remained steady.

To confirm this hypothesis, we repeated the same experiments but decreased β . That is, we configured UCSns with $\beta = 0.01$. In addition, we set $\theta_{GA} = 100$ to have more reliable parameters estimates before the GA triggered. Figure 4.3 shows the proportion of the best action map achieved by UCSns and UCSs. The results show a clear improvement of UCSs with respect to the original configuration. This supports our hypothesis that higher levels of noise require lower β values to allow for better parameter stabilization, coupled with higher θ_{GA} to let the genetic algorithm operate with better estimates. UCSs especially benefits from this, reaching almost 100% of the best action map in few iterations. Note that, for all the noise proportions, UCSs was still able to classify about 99% of the input instances correctly. That is, even when UCSs was trained with environments with 15% of alternating noise—i.e., 15% of the incoming instances are wrongly labeled—UCSs was able to classify nearly all the new free-noise instances correctly. This shows up the robustness provided by the inference scheme of UCS, which infers the class by means of a rule vote policy. The results of UCSns are practically the same as those obtained with the original configuration, as UCSns does not use β in the fitness estimate. Under this configuration, UCSs significantly outperformed UCSns for all the proportions of noise according to a Wilcoxon signed-ranks test at $\alpha = 0.05$.

Overall, this section evidenced the benefits of fitness sharing in the four boundedly difficult problems. This promotes the use of UCS with fitness sharing in the remainder of this thesis. In the next section, we extend this study and compare UCSs with XCS.

4.5 Comparing UCSs with XCS

This section introduces XCS in the comparison and empirically analyzes the effect of the architectural changes proposed by UCS with the aim of solving classification problems more effectively. Figures 4.4 and 4.5 show the proportion of the best action map achieved by UCSs and XCS on the four boundedly difficult problems. The results of UCSs are the same as those provided in the previous section. In all the problems, except for the parity problem, UCSs was significantly quicker in evolving the best action map than XCS according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. UCS's improvement on these problems is mainly result of (1) the exploration regime and (2) the new accuracy computation, and so, the fitness guidance of UCSs. As follows, these advantages, as well as the slightly poorer results obtained by UCSs in the parity problem, are analyzed in more detail.

Advantages due to the exploration regime. The first aspect that made UCSs evolve the optimal population more quickly than XCS is the exploration regime. That is, there are two reasons that explain, in general, why XCS requires more time than UCS to evolve the optimal population: (1) XCS needs to explore all the possible actions for a given input instead of only the class of the input and (2) XCS has to maintain the complete action map instead of only maintaining the best action map. This behavior is necessary in reinforcement learning problems where the consequences of each action have to be explored and modeled. Notwithstanding, this results in a delay in the convergence curves for the decoder, the position, and the multiplexer problems with respect to those of UCSs. Especially, note the large differences between XCS and UCSs in the decoder and the position problems. Although not as noticeable, UCSs also significantly outperformed XCS in the multiplexer problem with the two tested configurations according to a Wilcoxon signed-ranks test at $\alpha = 0.05$.

To exemplify the disadvantages of exploring the complete action map—as XCS does—, let us consider the decoder problem. The decoder problem is defined by 2^ℓ classes. As XCS explores uniformly each class, only 1 of each 2^ℓ explores will be made on the class of the input instance. The other $2^\ell - 1$ explores will be focused on classifiers that predict wrong classes. Therefore, the system will spend the proportion of $(2^\ell - 1)/2^\ell$ iterations to explore regions of the feature space that do not need to be explored in supervised learning problems. This issue takes on especial importance as the number of classes increases. This example is applicable to the other problems, but is especially important in the decoder and the position problems since the number of classes increases exponentially (for the decoder) and linearly (for the position) with the number of input bits.

On the other hand, the exploration regime seems not to provide UCSs with any advantage in the parity problem. That is, XCS was able to achieve the best action map in an amount of time that was significantly smaller than the time required by UCSs according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. Our hypothesis is that the exploration of consistently incorrect rules may help XCS discover consistently correct rules in this particular problem. That is, the parity problem has the particularity that, for each optimal classifier that predicts the correct class, there exists another one with the same condition but the wrong class in the complete action map. Therefore, mutation can easily generate a highly rewarded optimal classifier while exploring a niche with low rewarded classifiers. For example, if a consistently incorrect classifier such as 0001:0 is evolved, XCS may discover the consistently correct classifier 0001:1 by mutating the

4.5. COMPARING UCSS WITH XCS

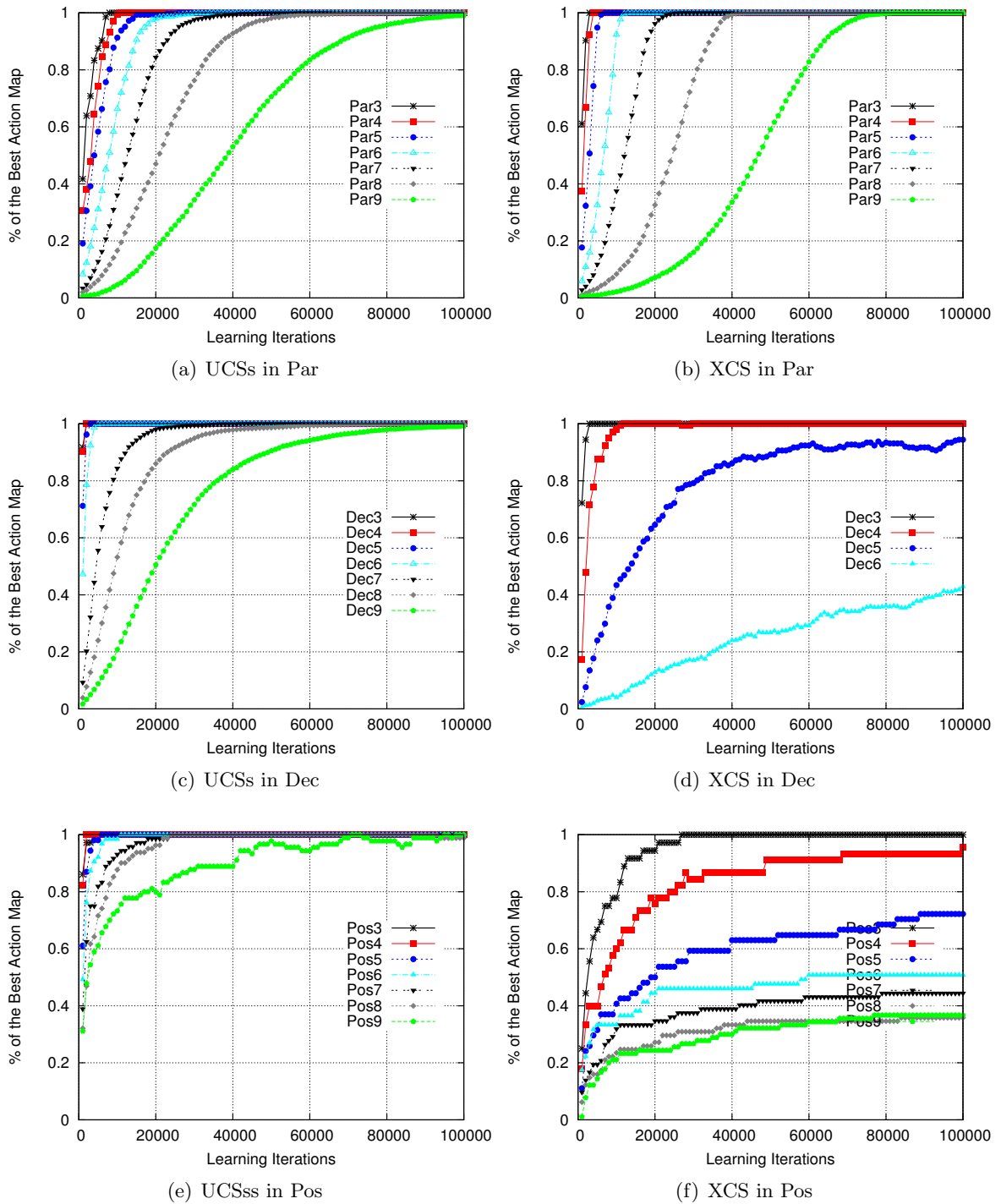


Figure 4.4: Proportion of the best action map achieved by UCSs and XCS in the parity, the position, and the decoder problems.

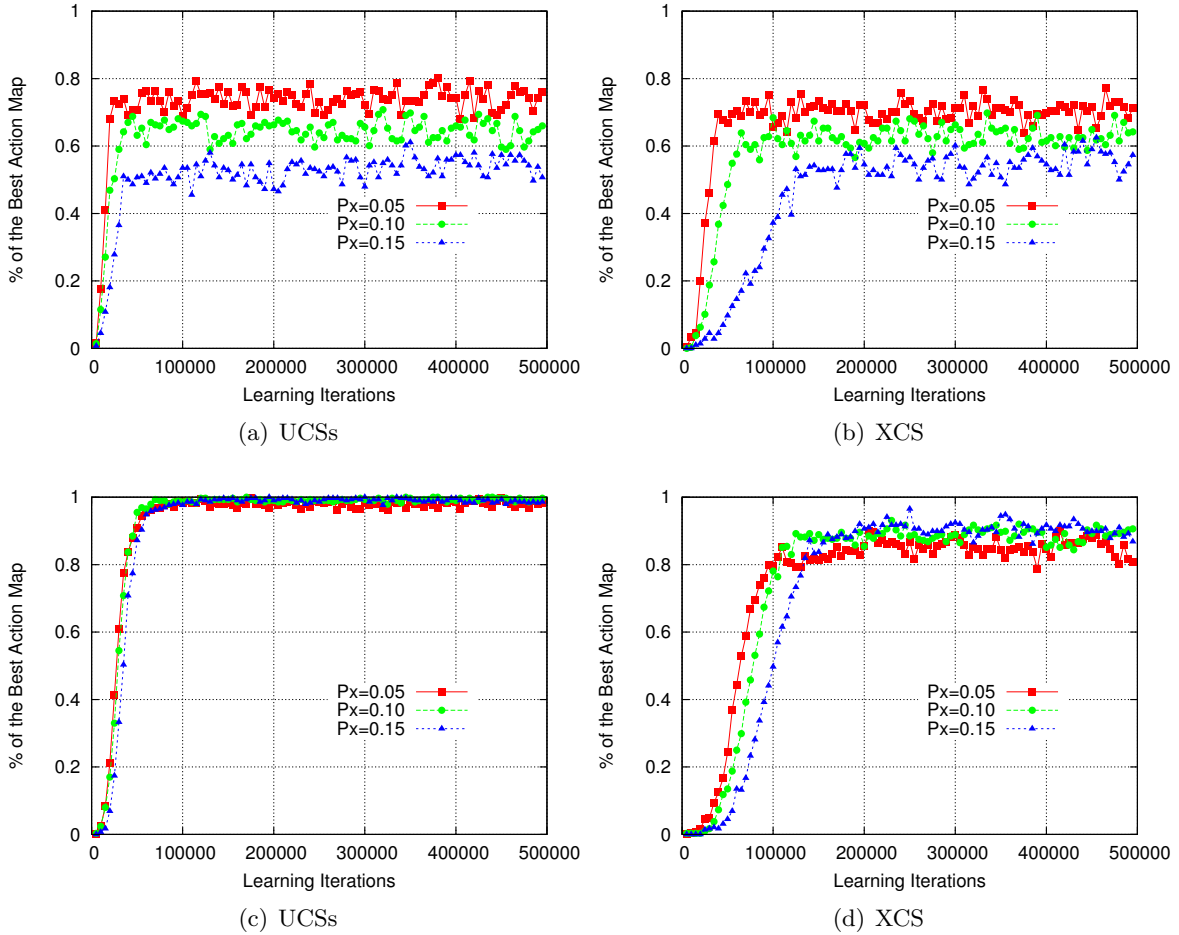


Figure 4.5: Proportion of the best action map achieved by (a,c) UCSs and (b,d) XCS in the noisy 20-bit multiplexer with $P_x = \{0.05, 0.10, 0.15\}$ with (a,b) the original configuration and with (c,d) the original configuration but setting $\beta = 0.01$ and $\theta_{GA} = 100$.

class of the rule. Conversely, UCSs would never maintain an inconsistently correct rule, since the selection operator would never choose a rule with $acc = 0$, and the deletion procedure would eventually remove this rule. As a consequence, UCSs cannot benefit from exploring low rewarded niches. This, coupled together with the fact that XCS is configured with a larger population size, results in that XCS can evolve the optimal population more quickly than UCSs. Nonetheless, note that, still, UCSs uses less computational resources since it is configured with a smaller maximum population size.

Advantages due to the fitness guidance. Although the explore regime explicates why UCSs can converge more quickly than XCS to the best action map, this may not be enough to explain the large differences observed in the decoder and the position problems. Here, we show that, as indicated by Butz et al. (2003), the fitness computation of XCS may provide a deceptive guidance toward the obtention of optimal classifiers in these two problems—and problems with

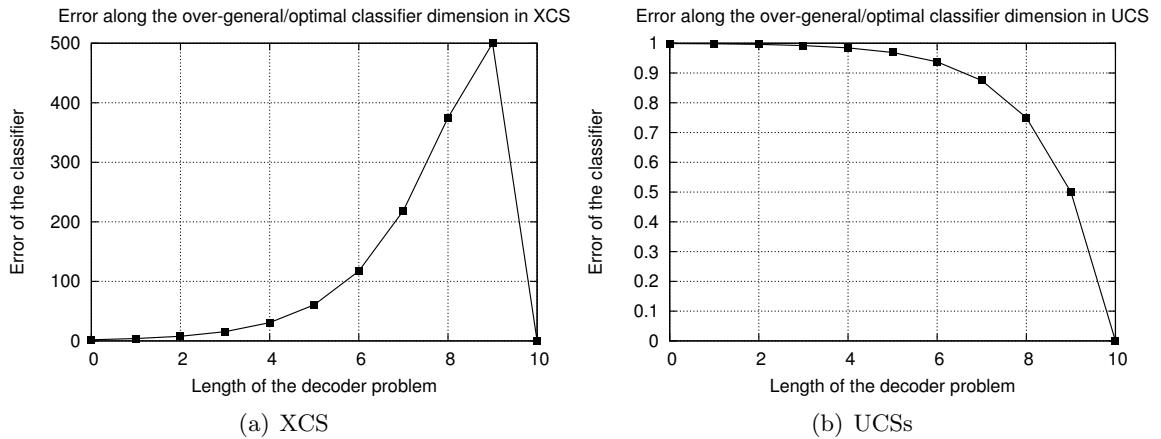


Figure 4.6: Error of XCS's classifiers along the over-general/optimal classifier dimension. The curve depicts how the error of the most over-general classifier #####:0 evolves as the bits of the classifier are specified, until obtaining the maximally accurate rule 000000000:0.

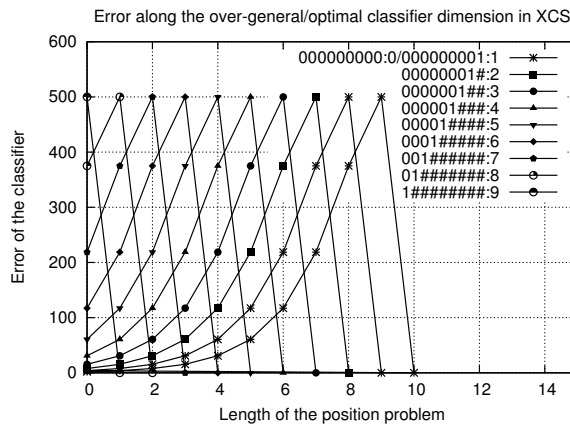


Figure 4.7: Error of XCS's classifiers along the over-general/optimal classifier dimension.

similar characteristics. Furthermore, we show that the fitness computation of UCSs overcomes this problem.

To exemplify the fitness misguidance in XCS in some particular problems, let us first focus on the decoder problem. The best action map of the decoder consists of maximally specific classifiers whose class is the decimal value of the binary input. Then, let us suppose that we have an over-general classifier cl_1 1###:8, whose theoretical prediction and prediction error estimates are $P=125$ and $\epsilon = 218.75$. XCS is expected to drive cl_1 to the classifier 1000:8. Imagine now that the genetic search generates the classifier cl_2 10##:8, whose prediction estimate is $P=250$ and whose prediction error is $\epsilon = 375$. Note that the error increases as we are approaching the optimal classifier, that is, 1000:8. Figure 4.6(a) extends this example and shows the evolution of the error from the over-general to the maximally general dimension

for the decoder problem with $\ell = 10$. Note that the error curve has an exponential increase as the classifier moves from the over-general to the maximum general side; the error abruptly decreases to zero when the optimal classifier is reached. Therefore, as long as the classifier moves toward the right direction, it gets smaller fitness, which consequently means fewer genetic opportunities and higher deletion probabilities. Thus, there is a misleading fitness pressure toward optimal classifiers.

This fitness misguidance is also present in the position problem, although the effect is not as important as in the decoder problem. To illustrate this, figure 4.7 shows an example of how the prediction error increases for each one of the optimal classifiers in the position problem with $\ell = 9$. Note that the misleading pressure is larger as the optimal classifier has more specific bits; that is, the Hamming distance from the maximally over-general classifier to the optimal classifier is larger as the optimal classifier is more specific, and so, the deception also increases.

This misleading pressure, which was termed as the *fitness dilemma* by Butz et al. (2003), does not only appear in these two problems, but in any problem whose optimal population contains specific classifiers. UCSs overcomes the fitness dilemma because the accuracy is calculated as the proportion of correct classifications instead of as a function of the accuracy of the prediction estimate. To exemplify this, figure 4.6(b) depicts the evolution of the error (that is, one minus the classifier's accuracy) along the over-general/maximally general dimension for the decoder problem with $\ell = 10$. Note that, in UCSs, the error diminishes as the classifiers approach the optimal one. In this way, UCSs accuracy guidance does not mislead the genetic search. This conclusion is also applicable to the original UCS, that is, UCSns.

Hence, the results provided in this section highlight that, as hypothesized in the previous chapter, UCS can evolve the best action map more quickly than XCS and spending fewer computational resources. Moreover, we also showed that the architecture of UCS does not suffer from the fitness dilemma. In the following section, we summarize all the observations provided in the present and the previous sections, identifying the key conclusions of the comparative analysis.

4.6 Lessons Learned from the Analysis

The empirical study performed in the last two sections provided many useful insights about the advantages of fitness sharing and the benefits of the UCS's architecture with respect to that of XCS in supervised learning problems. The purpose of this section is to gather and summarize all the observations. In particular, we first group the observations about the new fitness-sharing scheme of UCS. Then, we summarize the advantages provided by the explore regime and the accuracy guidance of UCS with respect to those of XCS. Each one of these aspects is elaborated in the following subsections.

4.6.1 Fitness Sharing

Fitness sharing speeded up UCS's convergence in all the tested problems. Especially, we argued that fitness sharing was the key to solve problems with class imbalances, where some optimal classifiers are activated with a lower frequency than other optimal and over-general classifiers. UCSns only yielded better performance in a single configuration of the 20-bit multiplexer with

alternating noise, which corresponded to a parameter setting that was not suited to solve the problem. In any case, this effect cannot be attributed to the presence or absence of fitness sharing, but rather to the way in which fitness is estimated. Recall that UCSns computes fitness as a power of accuracy, while UCSs computes fitness as a weighted windowed average with learning parameter β .

4.6.2 Explore Regime

In the comparison between XCS and UCS, the explore regime of UCS has shown to be a crucial aspect to speed up the convergence time of UCS with respect to the one presented by XCS in classification problems. That is, in general, exploring the best action map instead of the complete action map enables UCS to find the optimal solution more quickly than XCS, while spending fewer computational resources. On the other hand, we empirically illustrated that XCS may benefit from exploring the complete action map in problems, such as the parity problem, where the Hamming distance among the optimal classifiers that predict the wrong class and those that predict the correct class is small. Nevertheless, even in this case, UCS presented similar convergence times while using half of the maximum population size during the learning process, thus, saving computational resources.

Finally, it is worth noting that other exploring mechanisms could be adopted in XCS. That is, XCS uses a pure explore regime, where each available class is uniformly explored for each possible input. Nonetheless, as already pointed out by [Wilson \(1995\)](#) in the original design of the system, other exploration/exploitation schemes could be easily adapted to the system. For example, training in XCS could be based on an exploration regime that gradually changes from pure exploration toward increasing exploitation, similar to schemes such as ϵ -greedy or softmax that can be found in the reinforcement learning literature ([Sutton and Barto, 1998](#)). Changing the exploration regime may help XCS converge more quickly to the optimal solution; nonetheless, even with a new exploration/exploitation scheme, XCS would need to explore the different actions and evolve a complete action map, which prevent the system from being as competitive as UCS in solving hard classification tasks.

4.6.3 Accuracy Guidance

The results provided along the experiments showed that XCS may suffer from not only a lack of fitness guidance toward accurate classifiers, but also a deceptive guidance toward the optimal classifiers in some classification domains. This problem, already identified by [Butz et al. \(2003\)](#), was termed the fitness dilemma. Here, we showed that the problem appeared in almost all the tested problems, especially as the number of specific bits of the optimal classifiers increased. More specifically, we showed that XCS strongly suffered from the fitness dilemma in the decoder and, to a lower degree, in the position problems. As these problems gather some characteristics of real-world problems, this seems to indicate that XCS could suffer from the fitness dilemma in complex real-world classification problems. To alleviate the effect of the fitness dilemma in XCS, [Butz et al. \(2003\)](#) proposed a new error computation scheme that was addressed as bilateral accuracy. On the other hand, note that UCS could overcome the fitness dilemma since it computes the accuracy as the proportion of correct classifications.

4.6.4 Population Size

In the tested problems, UCS evolved best action maps with fewer learning iterations. Also, smaller population sizes were used in UCS in all the tested problems. The population evolved by XCS is generally larger, but comparable to that of UCS in terms of readability. In fact, by removing low-rewarded classifiers from XCS’s final population, we get a set of rules similar to that of UCS. Thus, the advantage of having smaller populations in UCS is the reduction of computational resources.

4.7 Summary and Conclusions

In this chapter, we set the double objective of (1) revisiting the architecture of UCS by introducing a fitness-sharing scheme and (2) empirically comparing XCS with UCS. With the first objective, we aimed at improving UCS to deal effectively with new challenging problems. The purpose of the second objective was to illustrate empirically how the modifications introduced by UCS to solve classification problems more scalably than XCS actually affected the system’s behavior.

We illustrated that the new fitness-sharing scheme enabled UCS to solve the four boundedly difficult problems more quickly. In essence, the fitness-sharing scheme helped UCS eliminate over-general classifiers as long as the first competing optimal classifiers were discovered; therefore, this speeded up the learning process. This was especially important in imbalanced domains such as the position problem, where highly specific, optimal classifiers had to compete with over-general classifiers. In this case, fitness sharing played a key role in decreasing the fitness of over-general classifiers, and so, in preventing these over-general classifiers from taking over a large proportion of the population, removing more specific, optimal classifiers. Thence, these observations promote the use of UCS with the new sharing scheme, instead of the original fitness computation, as a competitive tool for supervised learning, and, in particular, for learning from imbalanced domains. For this reason, we adopt this system in the remainder of our work.

The comparison with XCS allowed for a better understanding of the implications produced by the architectural changes introduced to UCS. In brief, we showed that UCS could solve the four classification problems spending fewer computational resources than XCS. Also, the analysis highlighted that UCS does not suffer from the fitness dilemma detected in XCS. Although the results and conclusions are limited to artificial problems, the experimental test bed contained many complexity factors present in real-world problems: multiple classes, noisy instances, and imbalanced classes, among others. Therefore, this encourages the use of UCS as a competent tool for supervised learning. It is worth noting that XCS is a broader architecture that can be applied to reinforcement learning, in general, and to some other tasks such as function approximation.

After the improvement of UCS and the empirical analysis provided herein, now we are in position to address the second and third objectives of this thesis, which study the performance of the two LCSs on domains that contain rare classes. Therefore, in the three subsequent chapters, we conduct a detailed analysis of the behavior of both LCSs on imbalanced domains and improve the ability of LCSs to extract accurate models from rare classes.

Chapter 5

Facetwise Analysis of XCS for Domains with Class Imbalances

The previous two chapters described XCS and UCS in detail, enhanced UCS with a new fitness-sharing scheme, and empirically compared the performance of these systems. This provided background information on how the two systems work and the key differences between them and led to the choice of using the fitness-sharing scheme instead of the original parameter update procedure in UCS. Although the experimentation empirically showed that both XCS and UCS can effectively solve boundedly difficult problems, there are still some challenges that need to be addressed to solve, scalably and efficiently, real-world problems.

A particular important challenge—shared by traditional machine learning techniques and GBML systems alike—is learning from domains that contain class imbalances. Learning from rare classes is a crucial aspect since the key knowledge usually resides in the minority class, and it has been shown that many traditional learning techniques are not able to extract accurate models from rare classes (Weiss, 2004). Therefore, the machine learning community has recently started to design new approaches that aim at improving the model discovery from rare classes (Chawla et al., 2004). Nonetheless, this aspect has been largely overlooked in online learning architectures. Imbalanced domains still pose more challenges to online learning systems, since the learner receives a stream of examples from which rare classes have to be modeled on the fly. The dearth of examples of rare classes may bias the parameter update procedure, forgetting the feedback provided by these examples, and so, hampering LCSs from evolving accurate classifiers that represent these rarities. In the following three chapters, we address this problem in the context of XCS and UCS, going from an analytic approach to the application of the system to solve real-world imbalanced problems.

This chapter studies the behavior of XCS on imbalanced domains and takes the lessons provided by the analysis to improve the modeling of rare classes. Although the analysis is centered on XCS in this chapter, we methodologically analyze the critical elements that any LCSs should satisfy to extract knowledge effectively from rare classes. Thence, we aim at providing a methodological framework rather than an analysis centered on a particular system. That is, we *decompose* the problem of learning from imbalanced domains in several *critical* elements and derive *facetwise models* (Goldberg, 2002) for each one of them. The integration of these models permits us to detect several crucial conditions that need to be met to ensure

that the system would extract the key knowledge from rare classes; in addition, we also identify *critical bounds* on the system behavior (Orriols-Puig et al., 2008a). The lessons learned from this analysis result in several configuration recommendations that, when followed, enable XCS to solve highly imbalanced classification problems that previously eluded solution. In the next section, we show that the whole framework can also be applied to UCS.

The remainder of this chapter is organized as follows. Section 5.1 presents the class-imbalance problem, reviews how the machine learning community has faced the problem with offline learning techniques, and places the problem in the context of online learning. Section 5.2 tests XCS on an imbalanced problem, intuitively discusses the complexities that learning from imbalanced domains may pose to the system, and empirically shows limits on the class-imbalance degree that the system can handle. Section 5.3 introduces the *facetwise analysis* methodology and specifies the steps that we follow in the present analysis. In sections 5.4, 5.5, 5.6, 5.7, and 5.8, we derive the different facetwise models. Section 5.9 integrates all the facetwise models, highlights the lessons learned along the analysis, and uses them to solve the 11-bit multiplexer problem (Wilson, 1995) with large degrees of class imbalance, which previously eluded solution. Finally, section 5.10 summarizes and concludes the chapter.

5.1 The Challenges of Learning from Imbalanced Domains in Machine Learning

During the last few decades, the increasing research on machine learning has led to the application of several learning techniques to real-world problems with the aim of extracting novel, interesting, and useful knowledge from these domains (Duda et al., 2000). One of the main characteristics of real-world problems is that some of the sub-concepts or classes may be poorly represented in the training data set due to either the scarcity of these concepts in nature or the cost—or inadequacy of the techniques used—to extract positive samples that represent these concepts. The purpose of this section is to highlight the importance of extracting accurate models from these rare concepts or classes in machine learning tasks. We begin emphasizing the high number of real-world applications in which we have class imbalances. Then, we briefly review how the machine learning community in general, and the GBML field in particular, has approached this problem, accentuating the differences between the challenges that learning from rare classes poses to offline learners and to online systems.

Examples of problems that contain rare classes abound in literature and include identifying fraudulent credit card transactions (Chan and Stolfo, 1998), learning word pronunciation (den Bosch et al., 1997), predicting pre-term births (Grzymala-Busse et al., 2000), detecting oil spills from satellite images (Kubat et al., 1998), and predicting telecommunication equipment failures (Weiss and Hirsh, 1998) among others. In these domains, while regularly occurring patterns can be modeled easily, learners tend to fail to extract accurate models from the rare classes (Japkowicz and Stephen, 2002; Japkowicz and Taeho, 2004; Weiss, 2004). Nonetheless, these rare classes are of primary interest, since they usually contain key knowledge. For this reason, strong research has recently been conducted on designing new approaches, or modifying existing machine learning techniques, with the aim of creating models that represent the rare classes more accurately (Weiss and Provost, 2003; Fawcett, 2008). All these approaches have been designed for offline supervised learning techniques, that is, methods that learn from static data

sets in which the examples are provided at the beginning of the learning process. In online learning, knowledge acquisition from imbalanced domains poses more severe challenges, since systems receive instances and rare classes have to be detected on the fly.

The many different approaches that have been designed to enhance the discovery of useful models of rare classes in offline learning can be grouped in methods working at (1) the learner level or at (2) the sampling level. Learner-level methods usually modify the error calculation of an existing system by either introducing a more appropriate inductive bias (Carvalho and Freitas, 2002) or assigning a misclassification cost per class (Pazzani et al., 1994). Their main drawback is that they are designed for specific learning algorithms, and so, they cannot be adapted to other learning techniques in a straightforward manner. For this reason, sampling-level methods have received much more attention than learner-level approaches. Sampling-level methods, usually known as *re-sampling techniques*, eliminate rare classes by balancing the proportion of examples per class of the training data set. As they are data-preprocessing methods, they can be generally used in any learning architecture. Notwithstanding, these methods can only be applied to static data sets. Many works have shown the benefits of re-sampling the training data sets in some specific imbalanced problems (Chawla et al., 2002; Japkowicz and Stephen, 2002; Batista et al., 2004; García and Herrera, 2008).

While the class-imbalance problem has been extensively analyzed for classical machine learning techniques that learn from static data sets, analyses and new approximations to overcome the problem in online learning are scarce. The typical approaches designed for offline learning to overcome the class-imbalance problem can barely be applied to online learning schemes, since they need to know the distribution of examples in the training data set. That is, in online learning, strategies such as over-sampling the occurrence of instances of the minority class or introducing a higher misclassification cost for minority class instances are impractical solutions. In this case, as we do not have a static data set, we can neither estimate the imbalance ratio to re-balance the training data nor assign a misclassification cost per class to favor the rare classes. Therefore, models for the minority class have to be learned online from a set of samples that come infrequently. This is the case of Michigan-style LCSs.

Although some ad hoc strategies have been proposed to alleviate this problem in particular LCSs (Holmes, 1998; Orriols-Puig and Bernadó-Mansilla, 2005a,b), these strategies cannot be directly extended to other LCSs. In general, these approaches have shown to improve LCSs performance on imbalanced domains. However, it is not clear how they affect the learning processes of LCSs; besides, they do not help increase our understanding of the actual problems that imbalanced domains pose to LCSs. Thence, in this chapter, we propose to start with a systematic analysis that explains the capabilities and limitations of the online learning architecture of LCSs to extract useful information from instances that come very infrequently. Specifically, we first apply the analysis to XCS and, in the next chapter, we carry it over to UCS. Before proceeding with the analysis, the next section provides an intuitive description of the problems that Michigan-style LCSs may face when learning from domains with rare classes. Then, the analysis methodology is introduced and all the study is developed in the subsequent sections.

5.2 The XCS Classifier System in Imbalanced Domains

As mentioned in the previous section, LCSs may have other problems, in addition to those presented by classical machine learning techniques in learning from imbalanced domains, since the model is learned online. With the system description provided in chapter 3 in mind, in this section we first appeal to the intuition to discuss the possible difficulties that XCS may find when learning from class imbalances, and we take advantage of the discussion to provide some notation that will be used in the remainder of this work. Then, before proceeding with a systematic study, we empirically analyze whether XCS is robust to class imbalances; to do this, we test the system on an artificial problem that is unbalanced by progressively removing instances of the minority class and check the maximum amount of under-sampling that the system can accept before failing to discover the concepts of the minority class.

5.2.1 Hypotheses of XCS Difficulties in Learning from Imbalanced Domains

Holland (1975) early defined the term of *schema* as a template that identified a subset of individuals, and used this notion to derive theory that explained how the good solutions are propagated along a GA run. Here, we take the same notion to define the concepts of *problem niche* and *representative* of a niche in LCSs, which will be used to study the effect of class imbalances in XCS. With these definitions, we review the components of the online learning architecture that may malfunction when learning from rare classes.

XCS evolves a distributed set of sub-solutions. In the remainder of this analysis, we use the term *problem niche* (or simply *niche*) to refer to a problem subspace where a maximally general sub-solution applies¹. Each niche is represented by a *schema* (Holland, 1975), which defines the value of the relevant attributes of the given problem niche, and the class or action of region covered by the schema. The order o of the schema is the number of relevant attributes of the niche. Then, we define that a *representative* of a niche is a classifier whose condition specifies the o relevant attributes of the niche schema correctly and that predicts the class or action of the sub-solution that the niche represents. Conversely, classifiers that do not match any problem schema and cover examples of different classes are referred to as *over-general classifiers*.

For instance, let us suppose that we have a niche represented by the schema $01*0**$ and whose class is 0. This means that all the examples matching $01*0**$ belong to class 0. Generalizing any of the o specific bits of the schema will cause the schema to cover instances of other classes, thence, not representing an accurate sub-solution anymore. Any classifier whose condition has the same value for the o specific bits and predicts the niche class is a representative of the niche. For example, classifiers $01\#0\#\#:0$, $0110\#\#\#:0$, and $010010:0$ are representatives of the niche. The maximally general representative of a niche is the classifier that only fixes the o relevant bits of the schema and predicts the action of the sub-solution, i.e., $01\#0\#\#:0$. An example of an over-general classifier is $\#1\#0\#\#\#:0$, since it generalizes the first fixed bit of the schema.

Therefore, Michigan-style LCSs evolve niches in a distributed manner, and the population consists of classifiers that represent niches and over-general classifiers. Then, a niche—and all the matching classifiers—is activated every time that an instance that matches the niche schema is sampled and the class niche is selected for exploration. In imbalanced domains, instances that

¹In XCS and UCS terms, an action set and a correct set, respectively, represent a niche.

belong to rare classes are sampled with a lower frequency; thence, niches that match these instances are activated with a lower frequency than the other niches of the system. In the remainder of this analysis, we address the niches that are activated by instances of the majority class as *nourished niches*. Conversely, niches activated by instances of the minority class are referred to as *starved niches*. We also define the *imbalance ratio* ir as the ratio of the number of examples of the majority class to the number of instances of the minority class that are sampled to the system.

Provided these definitions, we now intuitively analyze the possible difficulties that XCS may need to face in imbalanced domains by reviewing how the online architecture works. In the beginning of the learning process, the covering operator initializes the population with a set of classifiers that are generalized from the first sampled input instances. Therefore, the initial population consists mostly of over-general classifiers. Then, the system relies on (1) the parameter update procedure to obtain trusty estimates of classifier parameters online and (2) the evolutionary pressures (see section 3.1.6) to drive the population from a set of over-general classifiers to a set of maximally general and accurate classifiers that represent all niches. Both processes may be biased by the presence of rare classes.

The first peril that appears is that the parameter update procedure provides poor estimates of the parameters of the classifiers that match examples of different classes, that is, over-general classifiers. In XCS, classifier’s parameters are estimated by means of a weighted window average. Therefore, the system gives more importance to recently received rewards as opposed to older rewards, with the result that the reward provided by a particular example is forgotten after a certain number of learning iterations. Therefore, in imbalanced domains, it could be that examples of the minority class come so infrequently that the rewards provided by them are forgotten before receiving the next minority class instance, biasing the estimation of the parameters of over-general classifiers.

The evolutionary pressures may also be misled by the scarce sampling of minority class instances. Due to the occurrence-based reproduction of XCS, intuition seems to indicate that the system may suffer to discover representatives of starved niches, since these niches will be infrequently activated with respect to the other niches of the system. Hence, this low number of genetic opportunities combined with the other evolutionary pressures—which promote the most general classifiers—may discourage XCS from evolving representatives of starved niches.

In our study, we take all these hypotheses, systematically analyze the difficulties that XCS may face as the imbalance ratio increases, and provide solutions to let the system learn from highly imbalanced domains. To do this, we define several elements that need to be satisfied and derive models that relate these conditions with the imbalance ratio of the problem. Before proceeding with this analysis, in the next subsection we empirically show the behavior of XCS on an artificially imbalanced domain. The same experimentation is repeated after conducting the facetwise analysis and integrating all the models, emphasizing the importance of the lessons obtained from the analysis.

5.2.2 Empirical Observations of XCS Behavior on Class Imbalances

Here, we show the performance of XCS on the imbalanced multiplexer problem (Orriols-Puig and Bernadó-Mansilla, 2006a), a redefinition of the multiplexer problem where one of the classes is

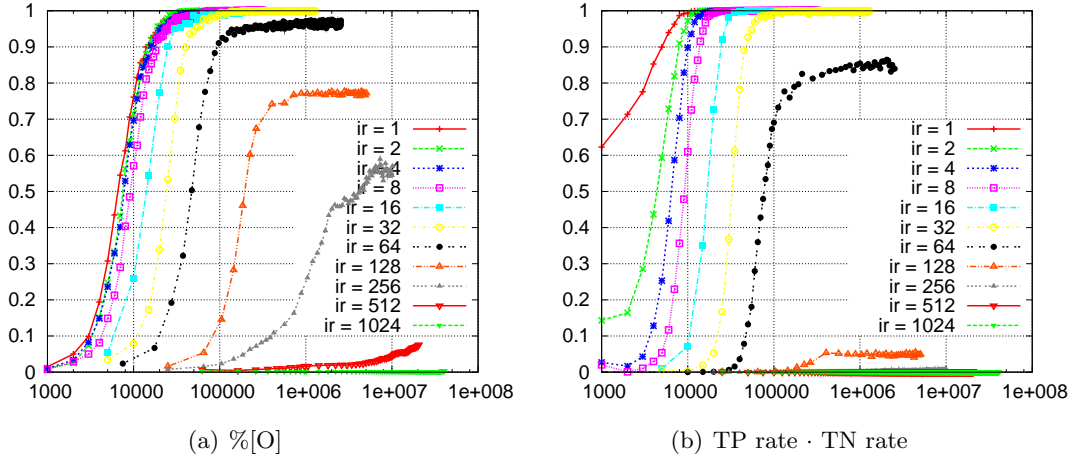


Figure 5.1: Evolution of (a) the proportion of the optimal population and (b) the product of TP rate and TN rate in the 11-bit multiplexer with imbalance ratios ranging from $ir=1$ to $ir=1024$.

progressively unbalanced with respect to the configuration parameter ir . That is, given a certain imbalance ratio ir , the sampling process of the multiplexer problem is modified such that the system receives ir instances of the majority class for each instance of the minority class. Thence, we empirically show the maximum imbalance ratio ir that XCS, using a standard configuration reported in the literature, could solve.

For this purpose, we ran XCS on the imbalanced multiplexer problem with imbalance ratios of $ir = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$. We set XCS with standard values used in the literature for its configuration parameters, that is:

$$\alpha = 0.1, \epsilon_0 = 1, \nu = 10, \chi = 0.8, \mu = 0.04, \theta_{del} = 20, \delta = 0.1, \theta_{sub} = ir, P_{\#} = 0.6.$$

We used tournament selection, two point crossover with $\chi = 0.8$, and bitwise mutation with $\mu = 0.04$. We applied GA subsumption, setting $\theta_{sub} = ir$ to avoid having poorly evaluated over-general classifiers considered as accurate subsumers. We ran XCS during $40\,000 \cdot ir$ iterations; thus, given a problem, we ensured that the system received the same number of genetic opportunities for all imbalance ratios. Finally, to prevent having young over-general classifiers with poorly estimated parameters in the final population, we introduced $5\,000 \cdot ir$ iterations with the GA switched off at the end of the learning process.

Figure 5.11(a) illustrates the evolution of the proportion of the optimal population %[O] achieved by XCS. That is, XCS was expected to evolve 32 optimal classifiers, each one representing a different niche. In this way, we measured the capacity of XCS to generalize and obtain the best representative of each niche at high imbalance ratios. Figure 5.1(b) depicts the evolution of the product of TN rate—i.e., the proportion of correct classifications of the over-sampled class—and TP rate—i.e., the proportion of correct classifications of the under-sampled class. Note that XCS can evolve all the optimal population and yield 100% of the product of TP rate and TN rate only for $ir \leq 32$. As the imbalance ratio increases, XCS is able to discover a lower proportion of the optimal population; particularly, the classifiers that represent starved niches

are not created and maintained by the system. For $ir \geq 64$, the system cannot discover the knowledge that resides in the minority class.

Therefore, these preliminary experiments show that XCS is robust at moderate imbalance ratios, but that it fails to provide accurate representatives of the minority class for high imbalance ratios. In the next section, we explicate the facetwise methodology used to analyze the possible causes of this failure. We enumerate the different elements that need to be guaranteed to learn accurate models from rare classes; then, models for each one of the elements are elaborated in the subsequent sections.

5.3 Facetwise Analysis of XCS in Imbalanced Domains

In this section, we follow a design decomposition approach to systematically analyze the different sources of difficulty that XCS may find when learning from imbalanced domains. We first briefly introduce the design decomposition methodology adopted by Goldberg (2002) as a design approach to competent genetic algorithms and review how it has been transported to XCS. Then, we follow this approach to decompose the problem of learning from imbalanced domains in XCS.

5.3.1 Design Decomposition in GAs

Goldberg (2002) emphasizes the relevance of *design decomposition* and *facetwise analysis* for advancing in the design and the understanding of complex systems. The design decomposition methodology separates the working process of complex systems into different elements, and each one of these elements is analyzed separately assuming that all the others are behaving in an ideal manner. All these models provide key insights into the working of the complex system, increasing our understanding of the underlying processes of the system. Furthermore, they also can be used as a tool for designing new competent and efficient complex systems that satisfy the requirements identified by the different models. Besides, the individual facetwise models can be combined, identifying the sweet spot where the algorithm actually scales. For further details about the application of this methodology to the design of competent GAs, the reader is referred to section 2.2.4.

As described in the previous section, similarly to GAs, LCSs are complex systems in which several components interact to evolve a set of maximally general and accurate classifiers. Due to this complexity, efforts have recently been made to apply the design decomposition and the facetwise analysis methodology to LCSs. In the next section, the existing work on carrying the design decomposition approach to LCSs domain is briefly reviewed.

5.3.2 Carrying the Design Decomposition from GAs to XCS

The application of facetwise modeling to LCSs, and specifically XCS, has provided key insights into XCS working processes, enabling the solution of more complex problems. As follows, we review these analysis in more detail.

One of the first works toward the definition of a theory based on facetwise analysis for XCS can be found in (Butz et al., 2004b). In this work, the authors studied the learning pressures in

XCS and derived critical bounds on the system convergence. In particular, the authors derived two boundaries beyond which the convergence of XCS could not be guaranteed, which were addressed as the *covering challenge* and the *schema challenge*. Continuing the analysis of the different pressures of XCS, Butz et al. (2003) analyzed the fitness pressure and derived the so-called *reproductive opportunity bound*, which sets the population size required to warrant that a classifier, with a certain specificity, will have reproductive opportunities. Later, Butz et al. (2004a) complemented the previous study by deriving models of the learning time in XCS. The models were simplified by not considering crossover and by assuming a domino-convergence model (Thierens et al., 1998). More recently, Butz et al. (2007) presented a Markov chain analysis of niche support in XCS. The analysis showed that the number of classifiers of a niche followed a binomial distribution, which yielded another population size bound to ensure effective problem sustenance.

However, these facetwise models do not explain all the aspects of XCS. Whereas these models have provided considerable insights and increased our understanding of XCS, they do not fully capture the effect of dealing with problems that contain class imbalances. Hence, in this thesis, we follow a design decomposition methodology to study whether XCS can efficiently deal with rare classes. As follows, we first present an artificial problem that will enable us to identify the different difficulties that we may face when learning from imbalanced domains. Later, we present the problem decomposition.

5.3.3 A Boundedly Difficult Problem for LCSs: The Imbalanced Parity Problem

The first step before proceeding to the facetwise analysis is to design a proper test problem that highlights the difficulties that are to be studied and that serves to validate the developed models. Following this analogy, we designed the imbalanced parity problem, whose description is provided as follows.

The imbalanced parity problem extends the parity problem (Kovacs and Kerber, 2001) by introducing a parameter that permits controlling the complexity along the imbalance dimension. The problem is defined as follows. Given a binary string of length ℓ , where there are k relevant bits ($0 < k \leq \ell$), the output is the number of one-valued bits in the k relevant bits modulo two. This corresponds to the original definition of the parity problem. We introduce the imbalance complexity to this definition by starving the class labeled as ‘1’. That is, ir denotes the ratio of examples of the majority class to the number of instances of the minority class. For $ir = 1$, the problem has, approximately, the same number of instances per class. For $ir > 1$, there are ir instances of the majority class for each instance of the minority class. Independent of the imbalance ratio, the optimal population for this problem consists of 2^{k+1} classifiers that have specific values for the k relevant attributes and all the remaining attributes are set to ‘#’ (see appendix A.1).

Note that the complexity of the problem can be moved along two dimensions: the building block size k and the imbalance ratio ir . Larger values of k pose more challenges to XCS, since the system needs to discover larger, more complex building blocks whose bits have to be processed together. Moreover, k also defines the number of irrelevant attributes which XCS must generalize to obtain the optimal population. On the other hand, increasing ir implies a progressive under-sampling of instances of the minority class, which may hinder XCS in discovering optimal

classifiers for this class.

Now that we have defined the parity problem and analyzed its possible sources of complexity, the next section introduces the particular decomposition proposed for Michigan-style LCSs in general.

5.3.4 Decomposition of the Class Imbalance Problem in XCS

With the intuitive difficulties of XCS in learning from rare classes discussed in section 5.2.1 and adhering to the design decomposition methodology proposed by Goldberg, we are now in position to articulate the different elements that need to be satisfied to successfully deal with problems that contain class imbalances. Our major concern is to guarantee that representatives of starved niches will win in competition with over-general classifiers even for high imbalance ratios. Thus, we decompose the problem and consider all the facets that are likely to be affected by the dearth of examples of the minority class. Then, we derive facetwise models that model each one of the subproblems. More specifically, we consider the following five subproblems:

1. Estimate the classifier parameters correctly—prediction, error, and fitness of classifiers.
2. Analyze whether representatives of starved niches can be provided in initialization.
3. Ensure the generation and growth of representatives of starved niches.
4. Adjust the GA application rate.
5. Ensure that representatives of starved niches will take over their niches.

Estimate the classifier parameters correctly. The primary factor that needs to be satisfied is that the evaluation procedure obtains accurate estimates of the parameters of all classifiers, and especially of over-general classifiers. In XCS, classifier parameters are updated online according to the reward received at each time step by means of the Widrow-Hoff rule (Widrow and Hoff, 1988). This method makes a temporal windowed average of the received rewards, which gives more importance to the last received rewards as opposed to the older rewards. Consequently, the dearth of sampling of examples that represent one of the classes may cause poor estimations in over-general classifiers, since infrequent negative rewards can be forgotten. This aspect is really important since it may completely mislead the genetic search. That is, if the error of over-general classifiers is underestimated, XCS may promote these over-general classifiers when they are competing with accurate classifiers in the same niche. We investigate the accuracy of the parameter update procedure and provide some alternative parameter evaluation methods in section 5.4.

Analyze whether representatives of starved niches can be provided in initialization. Once ensuring that classifier parameters can be properly evaluated, we have to analyze whether XCS initialization process can supply the initial population with classifiers that contain schemas of starved niches in the beginning of the run (Orriols-Puig et al., 2007c). That is, XCS starts with an empty population. Then, the covering operator is applied in the first iterations of the learning process, providing classifiers whose conditions are generalized from the first instances that are sampled to the system. Covering should initialize the population with several classifiers

that, although not being maximally accurate, contain schemas that represent niches of different classes. Nonetheless, intuition seems to indicate that, for highly imbalanced domains, covering is mainly triggered on instances of the majority class; therefore, covering may fail to provide schemas of niches that represent the minority class. Furthermore, this problem is even more severe as the schema of starved niches gets larger (in the parity problem, this translates to having larger values of k). In section 5.5, we derive models that formally explain this behavior. The remainder of the analysis is derived assuming a covering failure.

Ensure the generation and growth of representatives of starved niches. After the population is initialized, the genetic algorithm drives the search toward more accurate and general classifiers. We already appealed to the intuition that the occurrence-based reproduction of XCS may hamper the discovery of maximally general and accurate classifiers for starved niches. In section 5.6, we derive facetwise models that systematically analyze the effect of class imbalances in the (1) generation and (2) growth of representatives of starved niches. Consequently, we derive bounds on the population size to guarantee that XCS will be able to learn classifiers that belong to starved niches. All the analysis is performed assuming that the GA is applied at each learning iteration.

Adjust the GA application rate. Having ensured the discovery of representatives of starved niches, section 5.7 introduces the frequency of application of the GA into the analysis and theoretically shows that decreasing the frequency of application of the GA results in a counterbalancing effect that may help XCS discover representatives of starved niches.

Ensure that representatives of starved niches will take over their niches. Discovering the first representatives of starved niches is not enough. That is, once discovered, the accurate representatives of starved niches should take over their niches, removing competing over-general, less accurate classifiers. Nonetheless, the effect of the occurrence-based reproduction may produce the opposite effect, resulting in situations where over-general classifiers take over starved niches. In section 5.8, we study the takeover time of the best representative in starved niches, following the methodology used in the genetic algorithms literature for these types of analyses (Goldberg and Deb, 2003). Takeover time expressions are derived for the two most-used selection techniques in XCS: proportionate selection (Wilson, 1995) and tournament selection (Butz et al., 2005c). Moreover, conditions for starved niches extinction, i.e., deletion of the best representatives of starved niches in favor of over-general classifiers, are also derived for both selection schemes.

In the remainder of this chapter, each of these facets is covered in a different section in which the technical argument will be developed more completely. Moreover, all the models are experimentally validated with the *imbalanced parity problem*. Then, in section 5.9, we unify all the models, emphasizing the lessons derived from each one. Specifically, the patchquilt integration of the models results in a domain of applicability of XCS for imbalanced domains, which (1) determines under which conditions and imbalance ratios the system will be able to successfully represent the minority class in the evolved model and (2) provides guidelines of how to set the system for different imbalance ratios. The insights supplied by these models are crucial to increase our understanding of how XCS evolves classifiers that represent starved niches, enabling the solution of highly imbalanced problems that previously eluded solution. We

show, as example, that all the acquired knowledge enables us to appropriately set XCS so that it can solve the multiplexer problem (Wilson, 1995) with large amounts of class imbalances.

5.4 Estimation of Classifier Parameters

In this section, we analyze whether the Widrow-Hoff rule provides accurate estimates of the parameters of over-general classifiers as the imbalance ratio increases. We especially focus on the *prediction error* of over-general classifiers, since it determines the classifier's fitness. If the prediction error is underestimated, over-general classifiers may be considered as accurate classifiers by the system; hence, they will win in competition with more specific but accurate classifiers. Thence, as follows, we first theoretically relate the error of over-general classifiers with the imbalance ratio, deriving a bound beyond which the system will not be able to distinguish between over-general classifiers and accurate representatives. Then, we empirically analyze whether the parameter update procedure of XCS can provide estimates that accurately predict the theoretical bound.

5.4.1 Imbalance Bound

We start the derivation of the maximum imbalance bound by considering that, according to Butz et al. (2003), the prediction p of a classifier can be approximated by

$$p = P_c(cl) \cdot R_{max} + (1 - P_c(cl)) \cdot R_{min}, \quad (5.1)$$

where $P_c(cl)$ is the probability that a classifier predicts the matching input correctly, R_{max} is the maximum reward, and R_{min} the minimum reward given by the environment. Then, the error of a classifier can be approximated as

$$\epsilon = |p - R_{max}| \cdot P_c(cl) + |p - R_{min}| \cdot (1 - P_c(cl)). \quad (5.2)$$

For classification problems, R_{min} is usually 0, so that the prediction of a classifier can be estimated by $p = P_c(cl) \cdot R_{max}$. Substituting p into formula 5.2, we get the following prediction error estimate:

$$\epsilon = 2R_{max} \cdot (P_c(cl) - P_c(cl)^2). \quad (5.3)$$

Now, let us relate $P_c(cl)$ with ir (Orriols-Puig and Bernadó-Mansilla, 2006a, 2008a). In average, over-general classifiers will match ir examples of the majority class for each example of the minority class. Assuming that p is correctly estimated, a classifier would correctly predict the output for the ir instances of the majority class, and would give an erroneous prediction for the example of the minority class. Thus, $P_c(cl)$ can be approximated as

$$P_c(cl) = \frac{ir}{1 + ir}, \quad (5.4)$$

and its error estimate as

$$\epsilon = 2 \cdot R_{max} \frac{ir}{(1 + ir)^2}. \quad (5.5)$$

An over-general classifier will be considered inaccurate as long as

$$\epsilon \geq \epsilon_0. \tag{5.6}$$

Using equation 5.5, we obtain that

$$2 \cdot R_{max} \frac{ir}{(1 + ir)^2} \geq \epsilon_0, \tag{5.7}$$

which can be written as

$$-ir^2\epsilon_0 + 2ir(R_{max} - \epsilon_0) - \epsilon_0 \geq 0. \tag{5.8}$$

This represents a parabola where ϵ takes values higher than ϵ_0 for ir ranging between ir_ℓ and ir_u , where $ir_\ell < ir_u$. We are concerned about the maximum imbalance ratio up to which XCS would consider over-general classifiers as inaccurate; that is, ir_u . Solving equation 5.8, and assuming that $\epsilon_0 \ll R_{max}$, we obtain the following expression:

$$ir_u \approx \frac{2R_{max}}{\epsilon_0}. \tag{5.9}$$

That is, the maximum imbalance ratio up to which XCS will be able to detect over-general classifiers grows linearly with R_{max} and decreases linearly with ϵ_0 . Substituting $\epsilon_0 = 1$ and $R_{max} = 1000$, the maximum imbalance ratio is: $ir_u \approx 2000$. Nonetheless, the experiments provided in section 5.2.2 illustrated that XCS failed to extract all the knowledge that resides in the minority class for $ir > 32$. As proceeds, we analyze whether this deviation between theory and experiments can be caused by a deviation of the real value of the error with respect to its theoretical estimate.

5.4.2 Does the Widrow-Hoff Rule Provide Accurate Estimates?

Here, we empirically analyze if XCS can obtain reliable estimates as ir increases. For this purpose, we ran the imbalanced parity problem with $\ell = 11, k = 4$, and $ir = \{1, 10, 100\}$. We initialized XCS with the optimal population plus the most over-general classifier predicting class 0 (i.e., #####:0) and deactivated the GA. We set $\beta = 0.2$, which is a typical value used in the literature. Figure 5.2 shows a histogram of the error estimate of the most over-general classifier along a complete run. Results are averages over 10 runs. The vertical line shows the theoretical value for each case.

Theoretically, the error of the most over-general classifier should be $\epsilon = \{500, 165.28, 19.60\}$ for imbalance ratios $ir = \{1, 10, 100\}$ respectively. For a completely balanced domain (see figure 5.2(a)), the error oscillates around the theoretical value, i.e., $\epsilon = 500$. For $ir = 10$, the error of the most over-general classifier is around zero with high frequency. For $ir = 100$, the classifier error is zero most of the time. That is, as the classifier receives instances of the majority class, its error keeps on decreasing until becoming approximately zero. At some point, an instance of the minority class is sampled, which causes an increase in the error of the over-general classifier. For $ir = 100$, as 100 instances of the majority class are sampled for each instance of the minority

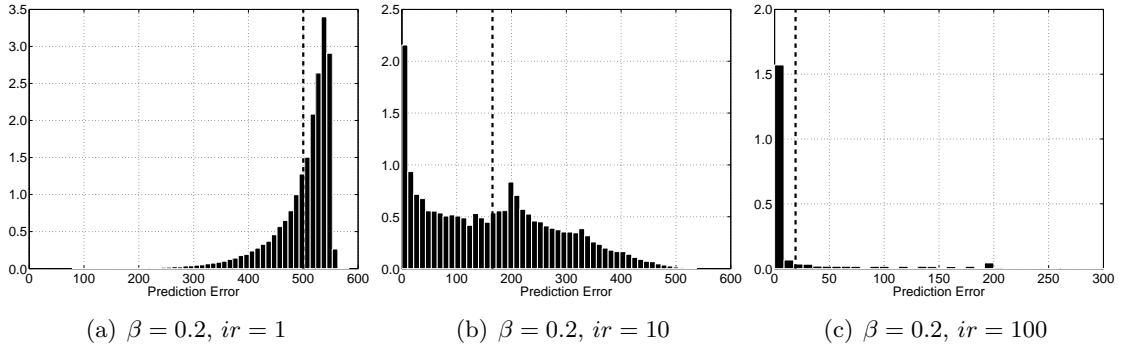


Figure 5.2: Histogram of the error of the most over-general classifier with Widrow-Hoff delta rule at $\beta = 0.2$ and different imbalance ratios.

class, the error of the most over-general classifier is underestimated during most of the time. Note that when $\epsilon < \epsilon_0$, XCS considers that the classifier is accurate (a typical value for $\epsilon_0 = 1$). This is the case during large part of the training time for $ir = 10$ and, especially, for $ir = 100$. Besides, as the classifier is the most over-general possible, it will be favored to the detriment of highly accurate but more specific classifiers.

The problem of having non-stable estimates for over-general classifiers does not appear exclusively in highly imbalanced domains. This topic has been studied for multi-step problems with large delayed rewards, and several approaches have been designed to propagate the error effectively along the previous action sets when several steps have to be taken before reaching a reward. Herein, we consider two methods to obtain better parameter estimates. First, we show that the Widrow-Hoff rule can obtain better parameter estimates if β is properly tuned. Then, we adapt one of the most relevant methodologies for parameter evaluation designed for multi-step problems to single step tasks: *gradient descent* (Butz et al., 2005a). The next two sections show that the two methods allow for better estimates in highly imbalanced domains.

5.4.3 Obtaining Better Estimates with the Widrow-Hoff Rule

In the Widrow-Hoff rule, the parameter β determines the proportion of update in the classifier parameters. As the Widrow-Hoff rule works as a temporal windowed average, β also fixes the capacity to forget past rewards. That is, high values of β produce large modifications of classifier parameters every time a new reward is received, forgetting perviously received rewards quickly. Usually, this allows for a faster convergence of the classifier parameters to their real values. However, we have already seen the harmful effect in imbalanced domains.

Here, we show that a simple solution to prevent the oscillation of the parameters of over-general classifiers is to decrease β , considering in this way a longer history of rewards. Lower values of β would cause smaller corrections, and so, less oscillations. Nonetheless, they would also imply slower convergence. For very small values of β , accurate offspring classifiers may lose against over-general parents at the beginning of the run, since their fitness increases slowly. This may impair XCS’s ability to discover new accurate and more specific rules. Thence, β should

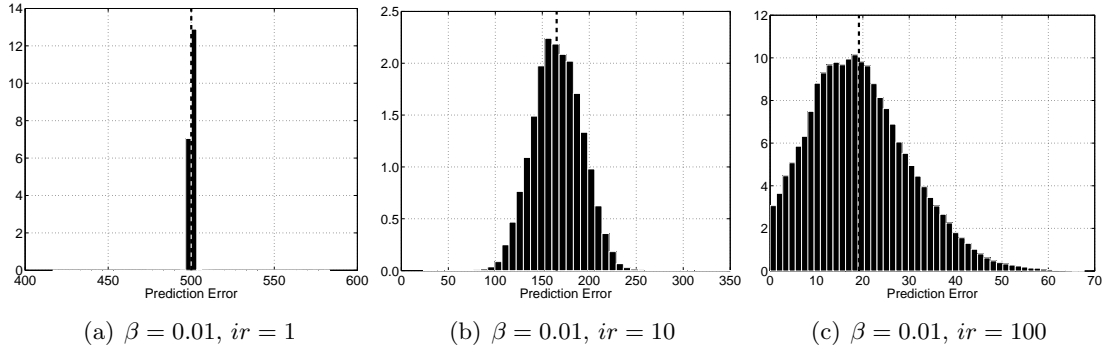


Figure 5.3: Histogram of the error of the most over-general classifier with Widrow-Hoff delta rule at $\beta = 0.01$ and different imbalance ratios.

be the highest value that prevents over-general classifiers from having zero error.

Figure 5.3 shows a histogram of the error estimate of the most over-general classifier along a complete rule for $\beta = 0.01$. For all the cases, the histograms are centered on the theoretical value of the error. Thus, over-general classifiers have precise estimates of their parameters most of the time. The standard deviation of the histograms increases with the imbalance ratio, since the rewards generated by the minority class examples are less frequent.

5.4.4 Obtaining Better Estimates with Gradient Descent Methods

Here, we study another approach, originally adapted from a gradient descent methodology, to obtain better parameter estimations. Butz et al. (2005a) introduced a gradient descent method to improve the parameter estimation of XCS in multi-step problems that involve a large number of state-action pairs. The new approach relies on identifying that the gradient term for a classifier is $\frac{F}{\sum_{[A]} F_i}$, in which F_i is the fitness of classifier i of the action set. Thus, classifier prediction is updated as

$$p = p + \beta(R - p) \frac{F}{\sum_{[A]} F_i}, \quad (5.10)$$

where R is the received reward. The rest of the parameters are updated as usual (see section 5.2). Note that this procedure aims at stabilizing classifier prediction. In consequence, the classifier error will also be stabilized since it tracks the prediction error.

The results provided by Butz et al. (2005a) illustrate that the gradient descent technique enables XCS to solve multi-step problems that eluded solution in XCS with Widrow-Hoff rule. Such improvement is explained by noting that the gradient term in the prediction update is actually an adaptive learning rate that prevents parameters from large corrections when the fitness of the updated classifier is much lower with respect to the fitness of the other classifiers in the same action set. Thereupon, gradient descent uses a heuristic procedure to automatically tune β depending on the fitness of each classifier. Then, the difference between gradient descent and decreasing β is that gradient descent automatically uses the aforementioned heuristic to

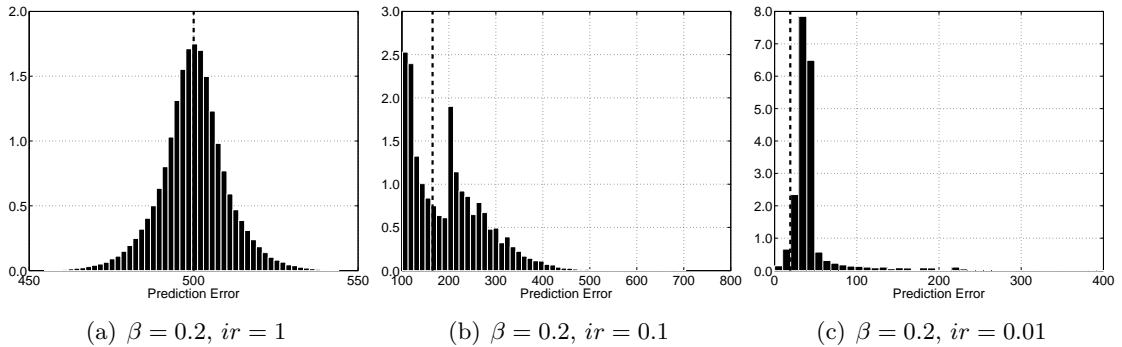


Figure 5.4: Histogram of the error of the most over-general classifier with gradient descent at $\beta = 0.2$ and different imbalance ratios.

determine the most suitable value for β instead of requiring the user to tune this parameter.

We repeated the same experiments with the imbalanced parity problem but now used gradient descent with $\beta = 0.2$. Figure 5.4 plots the histograms of the error estimate of the most over-general classifier along a complete run for gradient descent. XCS with gradient descent maintains fairly accurate estimates of the error of the most over-general classifier for all the imbalance ratios, even though a high value of β is used. However, note that these estimates are not as accurate as the ones obtained with Widrow-Hoff rule with a proper configuration of β .

In summary, this section showed that the Widrow-Hoff rule may provide poor estimates of the error of over-general classifiers for high class imbalances. This effect is undesirable since this may cause XCS to consider over-general classifiers as accurate. Two approaches, i.e., decreasing β for Widrow-Hoff rule and gradient descent provided more reliable parameter estimates. Although these methods would result in a slower convergence of XCS, their use is critical to guarantee that XCS will be able to obtain reliable estimates and converge to an optimal population. In the remainder of the analysis, we consider that classifier parameters are accurately estimated by the procedures presented above, and thus, that the genetic search is not misled. With this assumption, the next sections study the generation and growth of representatives in starved niches.

5.5 Supply of Schemas of Starved Niches in Population Initialization

Here, we study whether the covering operator is able to supply classifiers that represent schemas of starved niches for high imbalance ratios. As explained in section 5.2, covering is activated in the first stages of the learning process, creating new classifiers from the first sampled instances. To provide representatives of starved niches, the covering operator has to be triggered on minority class instances. However, as ir increases, fewer minority class instances are sampled. Thus, covering will be mainly activated from majority class instances. Then, most of the classifiers will be generalized from majority class instances, and so, classifiers representing schemas of the

minority class will be scarce. To analyze this effect, here we derive the probability that covering is triggered on the first minority class examples sampled to the system.

For this purpose, we first consider the probability that one instance is covered by, at least, one classifier $P(\text{cover})$. According to Butz et al. (2004b), this probability is

$$P(\text{cover}) = 1 - \left[1 - \left(\frac{2 - \sigma[P]}{2} \right)^\ell \right]^N, \quad (5.11)$$

where ℓ is the input length, N is the population size, and $\sigma[P]$ is the specificity of the population. During the first learning stage of XCS, we can approximate $\sigma[P] \approx 1 - P_\#$.

Now, let us relate this probability to the imbalance ratio ir . We consider the worst case where (1) XCS receives ir instances of the other classes before receiving the first instance of the minority class and (2) the covering operator is triggered for each instance supplying n classifiers per instance (where n is the number of classes; thus $\theta_{mna} = n$). In this case, the probability that the population contains, at least, a matching classifier for each class is

$$P(\text{cover}) = 1 - \left[1 - \frac{1}{n} \left(\frac{2 - \sigma[P]}{2} \right)^\ell \right]^{n \cdot ir}. \quad (5.12)$$

In this equation, we assumed that $N > n \cdot ir$, i.e., that XCS will not delete any classifier during the first ir iterations. This assumption is usually satisfied since covering is only applied for the first input examples, when there is room in the population. It is worth noting that, given a fixed ℓ and $\sigma[P]$, the term in brackets in the right hand of the equation decreases exponentially as the imbalance ratio increases. Thus, the probability of having classifiers in the population that match the first minority class instances tends to one exponentially with the imbalance ratio. Notice that the matching classifiers would have been generated from majority class instances, and so, would not represent schemas of starved niches.

With equation 5.12, we can derive the probability of activating covering having sampled a minority class instance. Provided that the probability of activating covering is $1 - P(\text{cover})$, and recognizing that $(1 - r/n)^n \approx e^{-r}$, we obtain that

$$P(\text{activate cov. on. min.}) = 1 - P(\text{cover}) \approx e^{-ir \cdot e^{-\frac{\ell \sigma[P]}{2}}}, \quad (5.13)$$

which decreases exponentially with the imbalance ratio and, in a higher degree, with the condition length and the initial specificity. Figure 5.5 depicts the equation for $\ell = 20$, $n = 2$, and different initial specificities, showing that the probability of activating covering on the first sampled instance of the minority class decreases exponentially with the imbalance ratio.

Thence, this analysis manifests that the covering operator fails to supply classifiers representing correct schemas of the minority class for moderate and high imbalance ratios. In the next section, we assume a covering failure in providing schemas of starved niches and study whether the genetic search can discover representatives of starved niches.

5.6 Generation of Classifiers in Starved Niches

Assuming a covering failure to provide classifiers that represent schemas of starved niches, we now study how the GA can evolve representative classifiers for these starved niches. As follows,

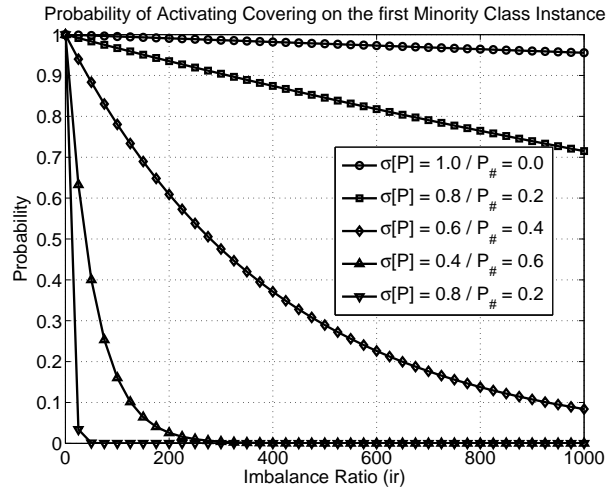


Figure 5.5: Probability of activating covering on a minority class instance given a certain specificity $\sigma[P]$ and the imbalance ratio ir . The curves have been drawn from equation 5.13 with $\ell = 20$ and different specificities.

we first enumerate the assumptions of the models and then analyze the probabilities of creating and maintaining representatives of starved niches. Finally, we use the different models to derive a population size bound to ensure the discovery of starved niches.

5.6.1 Assumptions for the Model

Before proceeding with the theoretical derivations, we first enumerate the assumptions made to develop the models. The analysis is focused on the evolution of starved niches. We assume a simplified scenario model where: (1) we do not consider crossover and contemplate mutation as the main operator for discovery, assuming low probabilities of mutation μ ($\mu < 0.5$) as usual in practice; (2) we assume that the GA is applied at each learning iteration (i.e., $\theta_{GA} = 0$); and (3) we consider random deletion. Subsequently, we relax all these constraints and experimentally analyze the impact of breaking each one of the assumptions. We experimentally examine the effect of introducing two point crossover. Furthermore, we investigate the biases caused by the enhanced deletion technique used currently in XCS (see section 3.1.4). In the next section, we study the effect of θ_{GA} theoretically and empirically.

5.6.2 Genetic Creation of Representatives of Starved Niches

In the first step of the analysis, we derive the time until the creation of the first representatives of starved niches, assuming that covering has not provided any of them. To achieve this, we first derive the probability to obtain the first accurate representative cl_{min} of the starved niche i , which is represented by a schema with order k_m . Thence, we study the probabilities of creating cl_{min} when sampling (1) instances of the minority class and (2) instances of the majority class. Recognizing that the probability of sampling a minority class instance is $1/(1 + ir)$ and the

probability of sampling a majority class instance is $ir/(1+ir)$, we can write that

$$P(cl_{min}) = \frac{1}{1+ir}P(cl_{min}|\text{min. inst}) + \frac{ir}{1+ir}P(cl_{min}|\text{maj. inst}). \quad (5.14)$$

Let us first derive $P(cl_{min}|\text{min. inst})$, that is, the probability of generating a representative of a starved niche when sampling a minority class instance. As we assumed that there are no representatives of starved niches in the population, the match set will only consist of over-general classifiers. Then, the system will choose a class randomly and will explore it, running a genetic event on the selected action set. Regardless of the selected class, and considering that there are only over-general classifiers in $[M]$, a representative of a starved niche can be created if all the k_m bits are correctly set to the values of the niche schema. Here, we consider the worst case and assume that all the k_m bits need to be mutated. Thence, the probability of getting the correct schema is $(\frac{\mu}{2})^{k_m}$. Besides, the class of the rule needs to be set to the class of the niche. If the system selected to explore the minority class (which will be selected with probability $1/n$, where n is the number of classes), we have to ensure that the mutation operator would not change this class (that is, with probability $(1-\mu)$). Otherwise, we have to require that the mutation operator change this class to the niche class (with probability $\mu/(n-1)$). Therefore,

$$P(cl_{min}|\text{min. inst}) = \frac{1}{n} \left(\frac{\mu}{2}\right)^{k_m} \cdot (1-\mu) + \frac{n-1}{n} \left(\frac{\mu}{2}\right)^{k_m} \cdot \frac{\mu}{n-1} = \frac{1}{n} \left(\frac{\mu}{2}\right)^{k_m}. \quad (5.15)$$

The same procedure can be followed to derive the probability of creating cl_{min} when sampling an instance of the majority class, i.e., $P(cl_{min}|\text{maj. inst})$. In this case, the match set will consist of both over-general classifiers and representatives of nourished niches. Again, we consider the worst case and assume that, to create a representative of a starved niche, all the k_m bits of the niche schema need to be mutated. Moreover, if the system chooses to explore the minority class, the class of the classifier must be preserved; otherwise, the class has to be changed to the minority class. This results in exactly the same probability as before, i.e.,

$$P(cl_{min}|\text{maj. inst}) = \frac{1}{n} \left(\frac{\mu}{2}\right)^{k_m}. \quad (5.16)$$

Substituting equations 5.15 and 5.16 into equation 5.14 we obtain that

$$P(cl_{min}) = \frac{1}{n} \left(\frac{\mu}{2}\right)^{k_m}. \quad (5.17)$$

Then, we can derive the time required to discover the first representatives of starved niches $t_{cl_{min}}$ as

$$t_{cl_{min}} = \frac{1}{P_{cl_{min}}} = n \left(\frac{2}{\mu}\right)^{k_m}, \quad (5.18)$$

which depends linearly on the number of classes and exponentially on the order of the schema, but does not depend on the imbalance ratio.

Thus, even though covering fails to provide classifiers representing schemas of the minority class, XCS will be able to generate the first correct classifiers of the minority class independent of the imbalance ratio. In the following, we derive the time until the deletion of these classifiers. With both the generation and deletion time, we calculate the minimum population size to maintain these classifiers and ensure that the best representatives of starved niches will receive, at least, one genetic event.

5.6.3 Deletion of Representatives of Starved Niches

We now provide an approximate time to the extinction of representatives of starved niches. The time to extinction of classifiers mainly depends on the applied deletion procedure. The current deletion scheme of XCS (Kovacs, 1999) gives the classifiers a deletion probability proportional to their action set estimate as . This approach permits balancing the allocation of rules in the different niches in problems for which the frequency of the different niches is similar, i.e., balanced problems. Nonetheless, in highly imbalanced problems, the action set size estimate of accurate classifiers of starved niches may be negatively biased by over-general classifiers. That is, as over-general classifiers participate in the same action sets as accurate classifiers of starved niches, the action set size of these accurate classifiers may be overestimated.

As our model is developed for highly imbalanced domains, we consider the worst case, i.e., that the deletion procedure gives the same probability to each classifier to be deleted. Since two classifiers are deleted at each GA application, we obtain that the deletion probability is $P(\text{delete cl}) = 2/N$, where N is the population size. From this formula, we derive the time until deletion:

$$t(\text{delete cl}) = \frac{N}{2}. \quad (5.19)$$

In the next section, we use both the creation and the extinction time of representatives of starved niches to derive the minimum population size that guarantees the discovery, maintenance, and growth of starved niches.

5.6.4 Bounding the Population Size

The population size is a critical aspect that determines the niches that the system could maintain. In this section, we use the formulas calculated above and derive population size bounds to guarantee (1) that XCS will be able to maintain accurate representatives of starved niches, and (2) that representatives of starved niches will receive genetic events before being removed.

Minimum Population Size to Guarantee Representatives

In the previous section, we theoretically showed that XCS would be able to create representatives of starved niches regardless of the imbalance ratio. To guarantee that these classifiers will be preserved in the niche, we require that, before deleting any representative of a starved niche, another representative for the given niche be generated. Therefore, we use the formulas derived in the previous section and require that the time until deletion be greater than the time until a new representative of a the starved niche is created. That is, we require that

$$t(\text{delete } cl_{min}) > t(cl_{min}). \quad (5.20)$$

Using formulas 5.18 and 5.19, the expression can be rewritten as

$$N > 2n \left(\frac{\mu}{2} \right)^{k_m}. \quad (5.21)$$

which indicates that the population size has to increase linearly with the number of classes and exponentially with the order of the schema to guarantee that representatives will be maintained in starved niches. Note that this formula does not depend on the imbalance ratio. This means that XCS will be able to maintain accurate classifiers in starved niches regardless the imbalance ratio.

Population Size Bound to Guarantee Reproductive Opportunities

To ensure the growth of starved niches, we not only should guarantee that XCS would maintain representatives of starved niches, but that these representatives receive, at least, a genetic opportunity. Otherwise, XCS could be continuously creating and removing classifiers from starved niches, but not searching toward better classifiers. Therefore, here we derive a population size bound to ensure this condition.

In our model, we assume that the selection procedure chooses one of the strongest classifiers in the niche (the effect of different selection schemes will be studied in more detail in the next section). Then, the time required for a classifier of a starved niche to receive a genetic event is inversely proportional to the probability of activation of the niche to which it belongs, i.e.,

$$t(\text{GA } niche_{min}) = n \cdot (1 + ir), \quad (5.22)$$

which depends on the imbalance ratio and the number of classes.

To guarantee that these accurate classifiers of starved niches receive a genetic opportunity before being deleted, we require that $t(\text{delete } niche_{min}) > t(\text{GA } niche_{min})$, from which we derive the population size bound

$$N > 2n \cdot (1 + ir). \quad (5.23)$$

That is, the population size has to increase linearly with the number of classes and the imbalance ratio to warrant that accurate classifiers of starved niches will receive, at least, a genetic event before being deleted.

Thereupon, the models derived in this section explained the creation, the maintenance, and the growth of starved niches, showing that XCS is able to maintain representatives of starved niches regardless of the imbalance ratio and that the population size has to increase linearly with the imbalance ratio if we want to ensure that the niche will grow. In the next section, we empirically validate the population size bounds with a set of artificial problems.

5.6.5 Experimental Validation of the Models

In this section, we experimentally evaluate whether the population size bound increases linearly with ir as predicted by the bound in equation 5.23. We first analyze whether the theory fits the experimental results when all the assumptions are made. Later, we study the impact of breaking each one of the assumptions.

Experimental Validation When the Assumptions Are Satisfied

To examine whether the theory approximates accurately the empirical results when all the assumptions are met, we performed the following experiments. We ran XCS on the imbalanced

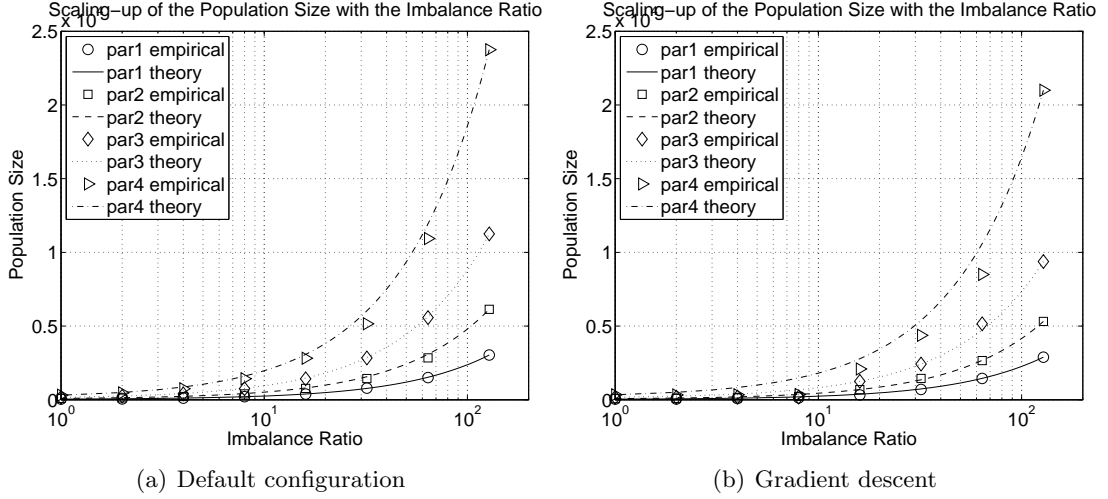


Figure 5.6: Scalability of the population size with the imbalance ratio in the k -parity problem with $k=\{1,2,3,4\}$ and the default configuration with (a) Widrow Hoff rule update with adjusted β according to ir and (b) gradient descent parameter update with $\beta = 0.2$. The dots show the empirical results and lines plot linear increases with ir (according to the theory).

parity problem with $k = \{1, 2, 3, 4\}$, $\ell = 10$, and $ir = \{1, 2, 4, 8, 16, 32, 64, 128\}$, and we used the bisection procedure to obtain the minimum population size required to solve the problem. That is, for each parity problem and imbalance ratio, we ran XCS with an initial, randomly selected population size. If the run succeeded, we decreased the population size. Otherwise, we increased the population size. This procedure was repeatedly applied until we obtained the minimum population size with which XCS was able to solve the problem. We employed the following procedure to determine if an XCS run was successful. After training, we tested XCS with all the training instances and measured the proportion of correct classifications of instances of the majority class (TN rate) and the proportion of correct classifications of the minority class (TP rate). All these results were averaged over 50 different random seeds. We considered that XCS succeeded if the product of TP rate and TN rate was greater than a certain threshold θ (we set $\theta = 0.95$).

XCS was configured so that all the assumptions of the model were satisfied. Therefore, crossover was deactivated ($\chi = 0$), random deletion was used, and the GA was applied every time a niche was activated ($\theta_{GA}=0$). The other parameters were set as $\alpha = 0.1$, $\epsilon_0 = 1$, $\nu = 10$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = ir$, $P_{\#} = 0.6$. We used tournament selection for the GA. We ran XCS during $\{10\,000 \cdot ir, 20\,000 \cdot ir, 40\,000 \cdot ir, 80\,000 \cdot ir\}$ iterations for the parity problem with $k = \{1, 2, 3, 4\}$ respectively; thus, given a problem, we ensured that the system received the same number of genetic opportunities for all imbalance ratios. Finally, to prevent having young over-general classifiers with poorly estimated parameters in the final population, we introduced $5,000 \cdot ir$ iterations with the GA switched off at the end of the learning process. In the remainder of this chapter, this configuration is referred to as the *default configuration*.

The two parameter update procedures proposed in section 5.4 were used: (1) Widrow-Hoff rule and (2) gradient descent. For the former method, we used the following heuristic procedure

to tune β . For each run, we supposed the worst case and assumed that over-general classifiers received 1 instance of the minority class and then ir instances of the majority class. Thus, we set β so that the error calculated for the most over-general classifier was approximately the same as the theoretical error provided by equation 5.9. We followed an iterative approach that incrementally discounted the value of β until a value that yielded error estimates that were close to the theoretical ones was found.

Figure 5.6 shows the minimum population size required to solve the parity with different building block sizes ($k = \{1, 2, 3, 4\}$) and imbalance ratios from $ir = 1$ to $ir = 128$ for Widrow-Hoff rule (figure 5.6(a)) and gradient descent (figure 5.6(b)). For each plot, the points depict the empirical values and the lines show the theoretical bounds. Note that the theory approximates the empirical results accurately for the two parameter update procedures and the different configurations and imbalance ratios. These results also permit establishing a comparison among the two parameter update procedures. The pairwise Wilcoxon statistical test (Wilcoxon, 1945), at $\alpha = 0.05$, indicated that gradient descent needed significantly smallest populations to solve the different configurations of the parity problem.

The results provided herein indicated that the theory nicely predicts the experiments when the assumptions of the models are met. In the next section, we investigate whether the population size bound is still valid when the different assumptions are not satisfied.

Impact of Breaking the Assumptions

Here, we repeated the experiments done in the previous section, but breaking each assumption. That is, we used the default configuration specified in the last section. Widrow-Hoff rule was employed for parameter estimation. Then, we ran XCS with (1) crossover, setting $\chi = 0.8$, and (2) the typical deletion scheme of XCS, configuring $\theta_{del} = 20$ and $\delta = 0.1$. Moreover, we also analyzed (3) the effect of replacing tournament selection with proportionate selection and (4) the impact of increasing the specificity in the initial population by setting $P\# = 0.4$. Figure 5.7 shows the minimum population size required in each configuration.

Several conclusions can be drawn from these results. First of all, it is worth noting that the theory nicely approximates the empirical results for all the experiments, although the initial assumptions were not satisfied. Figure 5.7(a) shows the curves obtained by XCS with crossover, which are equivalent to those evolved with the default configuration (see figure 5.6(a)) according to a Wilcoxon signed-ranks test at a significance level of 0.05. This suggests that the models are still valid although crossover is used in the experimental runs. Figure 5.7(b) plots the curves resulting from running XCS with the typical deletion scheme of XCS. The results clearly evidence the decrease in the population size required to solve the different configurations (the Wilcoxon signed-ranks test confirmed this observation). In any case, note that the the minimum population size still increases linearly with the imbalance ratio, as predicted by the theory. The configuration with proportionate selection (see figure 5.7(c)) yielded equivalent results to those obtained with the default configuration according to a Wilcoxon signed-ranks test at a significance level of 0.05. Finally, figure 5.7(d) illustrates the results obtained when there was a higher specificity in the initial populations. The pairwise analysis supports the hypothesis that a higher initial specificity requires larger population sizes to solve the parity with $k = 1$. For the other parity problems, no statistical differences were found.

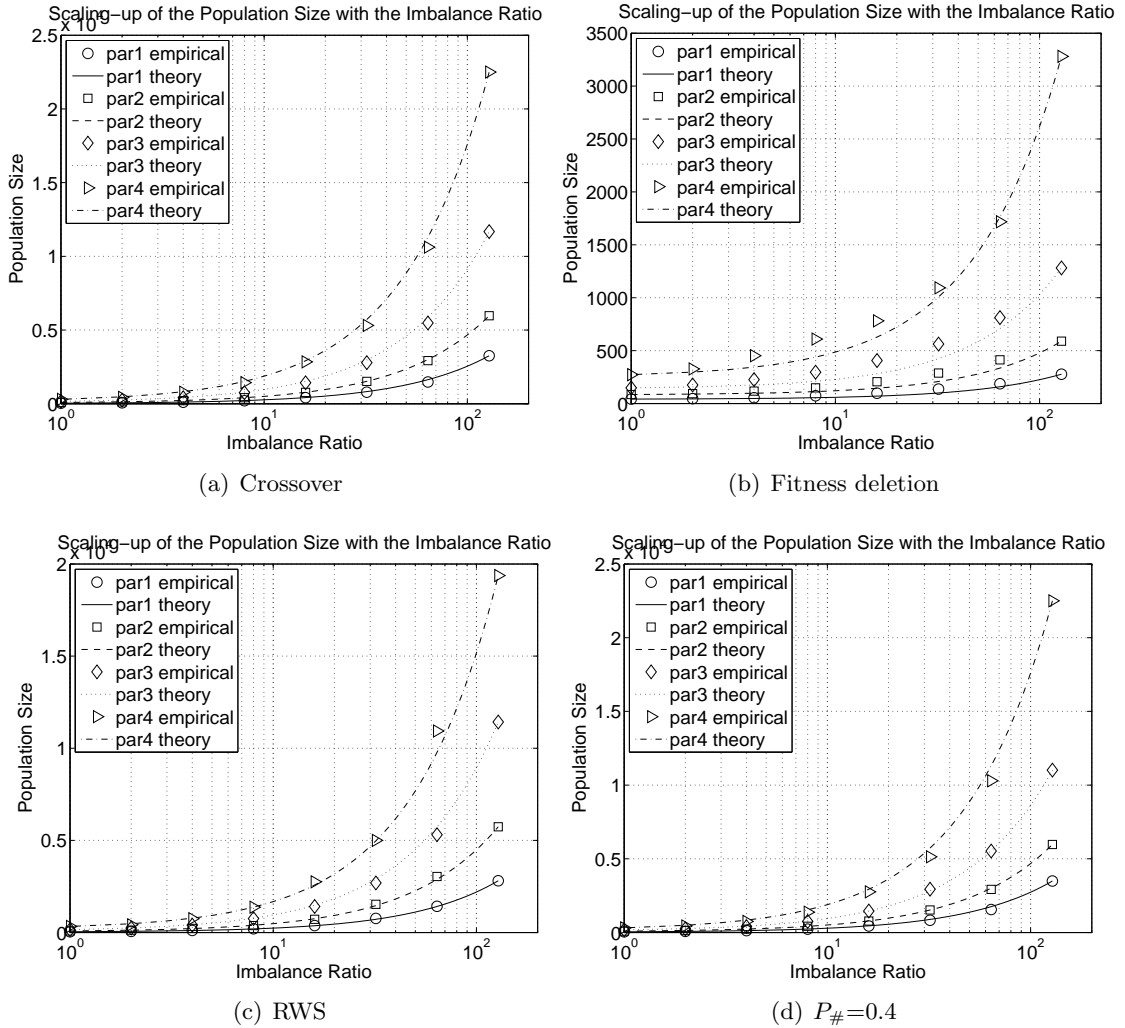


Figure 5.7: Scalability of the population size with the imbalance ratio in the k-parity problem with $k=\{1,2,3,4\}$ and different XCS's configurations. The dots show the empirical results and lines plot linear increases with ir (according to the theory).

The overall experimentation conducted in this section showed an agreement between theory and experiments, even when the initial assumptions were not satisfied. Notice that no experiment broke the assumption that the GA is applied at each learning iteration. The effect of varying the frequency of application of the GA is carefully studied in the next section.

5.7 Occurrence-based Reproduction: The Role of θ_{GA}

Throughout all the analysis performed in the last section, we assumed that the different niches of the system receive a genetic opportunity each time they are activated (i.e., $\theta_{GA}=0$). Consequently, nourished niches received more genetic events, and so, generated more offspring. This

section takes in consideration the effect of having $\theta_{GA} > 0$, revisits the models derived in the previous section, and shows that we can use θ_{GA} to re-balance the number of genetic opportunities that both starved and nourished niches receive. Finally, the new models are validated with the imbalanced parity problem.

5.7.1 Including θ_{GA} in the Generation Models

To analyze the impact of varying θ_{GA} , let us consider again the occurrence-based reproduction of both types of niches and calculate the period of application of the GA to the different niches. The frequency of activation of nourished niches ($F_{occ_{maj}}$) and the frequency of activation of starved niches ($F_{occ_{min}}$) are

$$F_{occ_{maj}} = \frac{1}{n \cdot m} \frac{ir}{1 + ir} \quad (5.24)$$

and

$$F_{occ_{min}} = \frac{1}{n \cdot m} \frac{1}{1 + ir}, \quad (5.25)$$

where m is the number of niches². From these frequencies, we can compute the period of activation of each type of niche as

$$T_{occ_{maj}} = n \cdot m \frac{1 + ir}{ir} \quad (5.26)$$

and

$$T_{occ_{min}} = n \cdot m(1 + ir). \quad (5.27)$$

Once activated, the niche will receive a genetic event if the time since the last application of the GA on the niche exceeds θ_{GA} . Therefore, the period of application of the GA (T_{GA}) on a niche is

$$T_{GA} = \begin{cases} T_{occ} & \text{if } T_{occ} > \theta_{GA} \\ \theta_{GA} & \text{otherwise.} \end{cases} \quad (5.28)$$

That is, if the period of activation of a niche is greater than θ_{GA} , the classifiers that belong to the niche will receive a genetic event every time the action set is formed; thus, the period of application of the GA equals the period of niche activation. This is the case of the theoretical model, in which we assumed $\theta_{GA} = 0$. On the other hand, if $T_{occ} \leq \theta_{GA}$, T_{GA} is approximately θ_{GA} .

To give all niches the same number of genetic events, T_{GA} should be approximately the same for all the niches. Note that this can be easily satisfied by setting $\theta_{GA} = T_{occ}^*$, where T_{occ}^* is the period of the niche that is activated less frequently, i.e., $T_{occ}^* = T_{occ_{min}}$. Therefore, θ_{GA} should be set as follows:

$$\theta_{GA} \approx n \cdot m \cdot (1 + ir). \quad (5.29)$$

²We introduce the number of niches in these equations since we are now modeling the occurrence of a specific niche

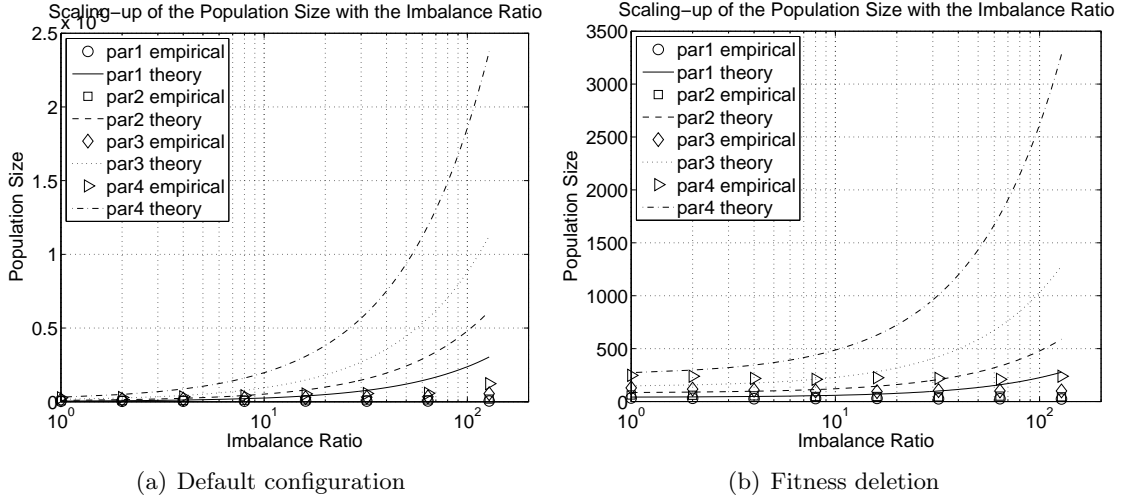


Figure 5.8: Scalability of the population size with the imbalance ratio in the k -parity problem with $k=\{1,2,3,4\}$ and different XCS's configurations with $\theta_{GA} = n \cdot m \cdot ir$. The points indicate the empirical values of the minimum population size required by XCS. The lines depict the theoretical increase calculated with the previous models, which assumed $\theta_{GA} = 0$.

Note that if the restriction of equation 5.29 is satisfied, all niches will receive approximately the same number of genetic events; moreover, as deletion is only activated after a GA application, the time of deletion of a classifier (see equation 5.19) would now increase linearly with the imbalance ratio. Therefore, XCS will be able to maintain starved niches without increasing the population size. The next section experimentally validates this assertion.

5.7.2 Experimental Validation

To validate the theory derived in the previous section, we ran the same experiments with the parity problem proposed in section 5.6.5. We configured the system with the default configuration, but we set $\theta_{GA} = n \cdot m \cdot ir$; Widrow Hoff rule was used to update classifier parameters. Figure 5.8(a) shows the minimum population size required to solve the parity problem with different building block sizes ($k = \{1, 2, 3, 4\}$) and imbalance ratios from $ir = 1$ to $ir = 128$. The points depict the empirical values. To analyze the differences introduced by adjusting $\theta_{GA} = n \cdot m \cdot ir$, the lines depict the population size increase predicted by the theoretical model calculated for the same configurations but with $\theta_{GA} = 0$ (see figure 5.6(a)).

The figure shows that, with the default configuration, the population size remained nearly constant for all the tested parity problems and imbalance ratios. This is because the effect of the imbalance ratio was counter-balanced by the increase of the period of application of the GA, as deduced in the previous section. The population size only presented a slight increase for $ir = 128$. This behavior can be easily explained as follows. At such imbalance ratios, the parameter update procedure decreases the value of β to prevent the oscillation of the parameters of over-general classifiers. For very small values of β , accurate offspring classifiers may lose against over-general parents at the beginning of the run, since their fitness increases slowly. As

the deletion procedure is random and these offspring receive a low number parameter updates, they may be removed before their parameters are correctly adjusted to the real value. Therefore, slightly larger populations may be set to let new accurate offspring persist in the population until their parameters are sufficiently updated.

To contrast this hypothesis, we ran the same experiments but used the typical deletion scheme of XCS. Figure 5.8(a) illustrates the minimum population sizes required for each configuration of the parity problem. The experimental results show that the population size remains constant as the imbalance ratio increases, even for the largest imbalance ratios. That is, the typical deletion scheme of XCS protects the young classifiers by giving more deletion probability to over-general, experienced classifiers.

With the present study, we have theoretically and empirically demonstrated that representatives of starved classifiers will be evolved independent of the population size. In the next section, we analyze the takeover time of these classifiers in more detail.

5.8 Takeover Time of Accurate Classifiers in Starved Niches

The study provided so far showed that XCS is able to create accurate classifiers of starved niches, and that these classifiers will receive, at least, a genetic opportunity before being deleted. This facet of the analysis set the population size requirements to guarantee that starved niches are represented. Also, the effect of increasing θ_{GA} was analyzed in detail. However, the conditions required in the previous models are not enough; to ensure full convergence, we have to warrant not only that starved niches will not be extinct but also that accurate classifiers will take over starved niches, removing the majority of over-general classifiers. Therefore, we have to analyze the competition between accurate classifiers of starved niches and over-general classifiers. This analysis is crucial because it permits extracting the upper bound on the admissible imbalance ratio under which XCS will be able to extract the key knowledge that resides in the minority class.

The purpose of this section is to model the takeover time of the best representatives of starved niches and determine the conditions under which starved niches will be extinguished. We first calculate the takeover time of accurate classifiers, which depends on (i) the initial stock of accurate classifiers in the niche and (2) the type of selection used by the GA. In LCSs, two selection procedures have mainly been considered: proportionate selection (Wilson, 1995) and tournament selection (Butz et al., 2005c). In this section, we model the takeover time of the best classifier of a niche for both selection schemes. Although we focus the analysis on the effect of class imbalances, note that the derived models can be used as general models for the two selection schemes. Then, we use the takeover time equations to calculate the extinction conditions of a niche, i.e., the conditions under which all representatives of a given starved niche will be removed from the population due to an overpressure toward generalization. As follows, we present the assumptions made for the analysis, develop the models for each type of selection, and experimentally validate the takeover time and extinction models.

5.8.1 Model Assumptions

Here, we provide the assumptions of the models. We derive takeover time models for problems with an arbitrary number of niches m . The models consider that all the m niches appear with the same frequency. In fact, in section 5.7, we showed that this could be easily achieved by setting θ_{GA} according to ir . Then, we incorporate the effect of the imbalance ratio in the error of the over-general classifier. That is, imagine that we have a two-class problem. For $ir = 1$, the error ϵ_o of the most over-general classifier will be $\epsilon_o \approx 500$, since this classifier will correctly predict half of the instances. As the imbalance ratio increases, ϵ_o decreases as shown in section 5.4. Thence, the imbalance ratio is intrinsically included in the difference between the errors of the over-general and the accurate classifiers. Thus, the takeover time models derived herein compute whether high imbalance ratios may discourage XCS to promote representatives of starved niches in favor of over-general classifiers.

To simplify the mathematical derivation of the models, we make the following assumptions. We consider that XCS has evolved a maximally general and accurate classifier cl_b , with error ϵ_b and numerosity n_b , for each niche of the system (this is ensured by the models provided in the last sections). Moreover, we assume that there is a single over-general classifier n_o , with error ϵ_o and numerosity n_o , which matches all the niches. As cl_b is maximally accurate, $\epsilon_o > \epsilon_b$. The same expression can be written in function of the classifiers accuracy (a inverse function of the error) as $\kappa_b > \kappa_o$. Therefore, our aim is to model the competition between accurate representatives of the different niches and the over-general classifier. For the analysis, we assume random deletion. We also consider that the GA is applied at each learning iteration and that both crossover and mutation are switched off. Therefore, the GA only selects two parents, copies and introduces them into the population, removing two other classifiers. The subsequent sections model the takeover time for proportionate and tournament selection under these assumptions.

5.8.2 Takeover Time for Proportionate Selection

In this section, we first derive the probability of selecting the best representative of a niche under proportionate selection, and then, we use this information to develop equations that model the evolution of the numerosity of this classifier in the niche. Under proportionate or roulette wheel selection (RWS), the selection probability of a classifier i depends on the ratio of its fitness F_i over the fitness of all classifiers in the action set. Without loss of generality, we assume that the classifier's fitness is a simple average of the classifier's relative accuracy. Thus, focusing on a single niche, we compute the fitness of classifiers cl_b and cl_o as

$$F_b = \frac{\kappa_b n_b}{\kappa_b n_b + \kappa_o n_o} = \frac{1}{1 + \rho n_r}$$

and

$$F_o = \frac{\kappa_o n_o}{\kappa_b n_b + \kappa_o n_o} = \frac{\rho n_r}{1 + \rho n_r},$$

where $n_r = n_o/n_b$ and ρ is the ratio between the accuracy of cl_o and the accuracy of cl_b ($\rho = \kappa_o/\kappa_b$). ρ can also be viewed as the fitness separation between cl_o and cl_b . The probability P_s of selecting the best classifier cl_b in the niche is computed as

$$P_{sRWS} = \frac{F_b}{F_b + F_o} = \frac{1}{1 + \rho n_r}.$$

Once selected, each classifier is copied and inserted into the population while one classifier is randomly deleted from the niche with probability $P_{del}(cl_j) = n_j/N$, where N is the population size. With this information, as proceeds we model the evolution of the best classifier in a niche.

Evolution of the Best Classifier

The numerosity of the best classifier cl_b at time $t+1$, $n_{b,t+1}$, given the numerosity of the classifier at time t , $n_{b,t}$, will

- increase in the next generation if the GA is applied to the niche, cl_b is selected by the GA, and another classifier is selected for deletion;
- decrease if (a) the GA is applied to the niche, cl_b is not selected by the GA, but cl_b is selected for deletion or if (b) the GA is not applied to the niche and cl_b is selected for deletion;
- remain the same, in all the other cases.

More formally,

$$n_{b,t+1} = \begin{cases} n_{b,t} + 1 & \frac{1}{m} \frac{1}{1+\rho n_{r,t}} \left(1 - \frac{n_{b,t}}{N}\right), \\ n_{b,t} - 1 & \frac{1}{m} \left(1 - \frac{1}{1+\rho n_{r,t}}\right) \frac{n_{b,t}}{N} + \frac{m-1}{m} \frac{n_{b,t}}{N}, \\ n_{b,t} & \text{otherwise.} \end{cases}$$

where m is the number of niches in the problem. Grouping the above equations, we obtain

$$n_{b,t+1} = n_{b,t} + \frac{1}{m} \cdot \frac{1}{1 + \rho n_{r,t}} \left(1 - \frac{n_{b,t}}{N}\right) - \frac{1}{m} \left(1 - \frac{1}{1 + \rho n_{r,t}}\right) \frac{n_{b,t}}{N} - \frac{m-1}{m} \frac{n_{b,t}}{N}, \quad (5.30)$$

which can be expressed as

$$n_{b,t+1} = n_{b,t} + \frac{1}{m} \frac{1}{1 + \rho n_{r,t}} - \frac{n_{b,t}}{N}. \quad (5.31)$$

This expression can be rewritten in terms of the proportion P_t of classifiers cl_b in the whole population, i.e.,

$$P_t = \frac{n_{b,t}}{N}. \quad (5.32)$$

Considering that the numerosity of the best classifier in each niche is $n_{b,t}$, we write that the numerosity of the over-general classifier $n_{o,t}$ is $n_{o,t} = N - m \cdot n_{b,t}$, and thus, $n_{r,t}$ can be expressed as

$$n_{r,t} = \frac{n_{o,t}}{n_{b,t}} = \frac{1 - m \cdot P_t}{P_t}. \quad (5.33)$$

Replacing equations 5.33 and 5.32 into equation 5.31, we obtain

$$P_{t+1} = P_t + \frac{1}{Nm} \frac{P_t}{P_t + \rho(1 - mP_t)} - \frac{1}{N} P_t. \quad (5.34)$$

Assuming $P_{t+1} - P_t \approx dp/dt$, we have

$$\frac{dp}{dt} \approx P_{t+1} - P_t = \frac{1}{Nm} \frac{P_t}{P_t + \rho(1 - mP_t)} - \frac{1}{N} P_t = \quad (5.35)$$

$$= \frac{P_t - mP_t^2 - \rho mP_t(1 - mP_t)}{Nm[P_t + \rho(1 - mP_t)]}. \quad (5.36)$$

That is,

$$\frac{P_t(1 - m\rho) + \rho}{P_t[(1 - \rho m) - mP_t(1 - \rho m)]} dp = \frac{1}{Nm} dt, \quad (5.37)$$

which can be solved by integrating each side of the equation between the initial proportion P_0 of cl_b and the final proportion P_F of cl_b up to which cl_b has taken over the population. Note that, assuming m balanced niches, cl_b will take over, at most, a proportion $1/m$ of the population.

$$\int_{P_0}^{P_F} \frac{P_t(1 - m\rho) + \rho}{P_t[(1 - \rho m) - mP_t(1 - \rho m)]} dp = \frac{1}{Nm} \int dt = \frac{t}{Nm}. \quad (5.38)$$

This integral can be solved as follows

$$\frac{t}{Nm} = \int_{P_0}^{P_F} \frac{1}{1 - mP_t} dp + \frac{\rho}{1 - m\rho} \int_{P_0}^{P_F} \frac{1}{P_t[1 - mP_t]} = \quad (5.39)$$

$$= \left[-\frac{1}{m} \ln(1 - mP_t) - \frac{\rho}{1 - m\rho} \ln \left| \frac{1 - mP_t}{P_t} \right| \right]_{P_0}^{P_F} = \quad (5.40)$$

$$= -\frac{1}{m} \ln \left(\frac{1 - mP_0}{1 - mP_F} \right) + \frac{\rho}{1 - m\rho} \ln \left| \frac{P_F(1 - mP_0)}{P_0(1 - mP_F)} \right|. \quad (5.41)$$

Since $P_t < 1/m$, we can rewrite the expression as

$$\frac{t}{Nm} = -\frac{1}{m} \ln \left(\frac{1 - mP_0}{1 - mP_F} \right) + \frac{\rho}{1 - m\rho} \ln \left(\frac{P_F(1 - mP_0)}{P_0(1 - mP_F)} \right), \quad (5.42)$$

from which we derive the takeover time of cl_b in roulette wheel selection

$$t_{RWS}^* = Nm \left[\frac{1}{m} \ln \left(\frac{1 - mP_0}{1 - mP_F} \right) + \frac{\rho}{1 - m\rho} \ln \left(\frac{P_F(1 - mP_0)}{P_0(1 - mP_F)} \right) \right]. \quad (5.43)$$

The takeover time formula depends on (1) the fitness separability ρ and (2) the number of niches m . If ρ increases, the influence of the second logarithm also increases; therefore, as the accuracies of cl_b and cl_o get closer, the takeover time increases.

In this subsection we provided a closed-form solution of the takeover time for proportionate selection. In the next subsection, we derive the conditions under which the best classifier will not take over its niche.

Conditions for the Extinction of Starved Niches under Proportionate Selection

With the formulas derived above, we analyze under which circumstances the best classifier will not be able to take over the population, and then, we relate it to the imbalance ratio. For

this purpose, we take equation 5.35 and analyze under which conditions the increment of the numerosity of the best classifier will be negative; in this case, the best classifier will lose copies in favor of over-general classifiers. We can write this expression as

$$\frac{P_t - mP_t^2 - \rho mP_t(1 - mP_t)}{Nm[P_t + \rho(1 - mP_t)]} < 0, \quad (5.44)$$

that is,

$$\frac{P_t(1 - m\rho)(1 - mP_t)}{Nm[P_t(1 - m\rho) + \rho]} < 0. \quad (5.45)$$

This condition holds when the numerator is positive and the denominator is negative and viceversa. Thus, we search for values of ρ and m that result in combinations of positive numerator and negative denominator and viceversa. Assuming that $P_t < \frac{1}{m}$, we have the following cases. For $\rho < \frac{1}{m}$, both numerator and denominator take positive values. Therefore, for $\rho < \frac{1}{m}$, the best classifier will always take over the population. For $\rho = \frac{1}{m}$, the expression is 0, indicating that numerosity of the best classifier would remain constant. For $\rho > \frac{1}{m}$, the numerator is always negative. Then, the whole expression will be negative if the denominator is positive, i.e.,

$$Nm[P_t(1 - m\rho) + \rho] > 0, \quad (5.46)$$

which can be written as

$$P_t < \frac{\rho}{m\rho - 1}. \quad (5.47)$$

Having that $0 < P_t < \frac{1}{m}$ and $\frac{1}{m} < \rho \leq 1$, this expression is always satisfied for $m \geq 2$. Thence, this theoretically demonstrates that for $m \geq 2$ the best representative will not be able to take over the niche if

$$\rho > \frac{1}{m}, \quad (5.48)$$

That indicates that the best classifier will not take over its niche if the ratio of the accuracy of the over-general classifier to the accuracy of the best classifier is greater than $1/m$, that is, the fitness separability between both classifiers is small.

Note that the expression in equation 5.47 can be easily related to the imbalance ratio by identifying that $\rho = \frac{\kappa_o}{\kappa_b}$, where the accuracy of the over-general classifier can be computed from its error, which is expressed in equation 5.9. That is, recognizing that the accuracy of the representative of each niche is one ($\kappa_b = 1$), and that the accuracy of the over-general classifier κ_o is

$$\kappa_o = \alpha \left(\frac{\epsilon_o}{\epsilon_0} \right)^{-\nu}, \quad (5.49)$$

and considering the relation between the imbalance ratio and the error computed in equation 5.9, we obtain that

$$\rho = \alpha \left(\frac{(1 + ir)^2 \epsilon_0}{2 \cdot ir \cdot R_{max}} \right)^{\nu}. \quad (5.50)$$

Replacing equation 5.50 into equation 5.48, we can write that the best classifier will not take over its niche if

$$\alpha \left(\frac{(1 + ir)^2 \epsilon_0}{2 \cdot ir \cdot R_{max}} \right)^\nu > \frac{1}{m}. \quad (5.51)$$

This expression can be derived as

$$ir^2 + \left(2 - \frac{(m\alpha)^\nu 2R_{max}}{\epsilon_0} \right) ir + 1 > 0. \quad (5.52)$$

Recognizing that

$$1 \ll \frac{(m\alpha)^\nu 2R_{max}}{\epsilon_0}, \quad (5.53)$$

and making further simplifications, we can get that, under proportionate selection, the best classifier will not take over its niche if

$$ir > \frac{(m\alpha)^\nu 2R_{max}}{\epsilon_0}. \quad (5.54)$$

Note that the maximum accepted ir can be modified by decreasing ϵ_0 .

After developing the same models for tournament selection, in section 5.8.4, we experimentally validate the takeover time under proportionate selection, and show that the best classifier cannot take over the niche for $\rho > 1/m$.

5.8.3 Takeover Time for Tournament Selection

To develop takeover time models for tournament selection, we assume that the tournament size s is fixed during all the learning process. That is, in each GA event, tournament selection randomly chooses s classifiers in the action set and selects the one with the highest fitness. As before, we assume that cl_b is the best classifier in the niche and cl_o is the over-general classifier; in terms of tournament selection, this translates into requiring that $f_b > f_o$, where f_i is the fitness of the micro-classifiers associated with cl_i (i.e., $f_i = F_i/n_i$). Given this scenario, the probability of selecting the best classifier is

$$P_{sTS} = \left[1 - \left(1 - \frac{n_o}{n} \right)^s \right], \quad (5.55)$$

where n is the numerosity of the niche, i.e., $n = n_b + n_o$. Thus, the probability of selecting the best classifier is one minus the probability that this classifier does not participate in the tournament. With this information, the next subsections model the evolution of the best classifier, provide some particular expressions of the takeover time for tournament selection, and extract the critical bounds beyond which the best representative will not take over its niche.

Evolution of the Best Classifier

We first model the evolution of the numerosity of the best classifier cl_b at time $t + 1$, $n_{b,t+1}$, given the numerosity of the classifier at time t , $n_{b,t}$, which will

- increase if the GA is applied to the niche, cl_b is selected to participate in the tournament, and another classifier in the population is selected for deletion;
- decrease if (1) the GA is applied to the niche, cl_b is not selected to participate in the tournament, but it is selected for deletion; or if (2) the GA is applied to another niche, and cl_b is selected for deletion;
- remain the same otherwise.

More formally,

$$n_{b,t+1} = \begin{cases} n_{b,t} + 1 & \frac{1}{m} \left[1 - \left(1 - \frac{n_{b,t}}{n} \right)^s \right] \left(1 - \frac{n_{b,t}}{N} \right), \\ n_{b,t} - 1 & \frac{1}{m} \left(1 - \frac{n_{b,t}}{n} \right)^s \frac{n_{b,t}}{N} + \frac{m-1}{m} \frac{n_{b,t}}{N}, \\ n_{b,t} & \text{otherwise.} \end{cases}$$

Grouping the above equations we can derive the expected numerosity of cl_b ,

$$n_{b,t+1} = n_{b,t} + \frac{1}{m} \left[1 - \left(1 - \frac{n_{b,t}}{n} \right)^s \right] \left(1 - \frac{n_{b,t}}{N} \right) - \frac{1}{m} \left(1 - \frac{n_{b,t}}{n} \right)^s \frac{n_{b,t}}{N} - \frac{m-1}{m} \frac{n_{b,t}}{N}, \quad (5.56)$$

from which we obtain

$$n_{b,t+1} = n_{b,t} + \frac{1}{m} \left[1 - \left(1 - \frac{n_{b,t}}{n} \right)^s \right] - \frac{n_{b,t}}{N}. \quad (5.57)$$

As done for proportionate selection, we rewrite the formula above in function of the proportion P_t of classifiers cl_b in the whole population, i.e.,

$$P_t = \frac{n_{b,t}}{N}. \quad (5.58)$$

From which we can calculate n_o as

$$n_o = N - mn_b = N - mNP_t = N(1 - mP_t), \quad (5.59)$$

and n as

$$n = n_b + n_o = NP_t + N(1 - mP_t). \quad (5.60)$$

Substituting equations 5.59 and 5.60 into equation 5.57, we obtain

$$NP_{t+1} = NP_t + \frac{1}{m} \left[1 - \left(1 - \frac{P_t}{1 + P_t(1 - m)} \right)^s \right] - P_t, \quad (5.61)$$

$$P_{t+1} = P_t + \frac{1}{mN} \left[1 - \left(1 - \frac{P_t}{1 + P_t(1 - m)} \right)^s \right] - \frac{1}{N} P_t. \quad (5.62)$$

Assuming $\frac{dp}{dt} \approx P_{t+1} - P_t$, we derive

$$\frac{dp}{dt} \approx P_{t+1} - P_t = \frac{1}{mN} \left[1 - \left(1 - \frac{P_t}{1 + P_t(1 - m)} \right)^s - mP_t \right]. \quad (5.63)$$

That is,

$$\frac{1}{mN} dt = \frac{1}{1 - \left(1 - \frac{P_t}{1+P_t(1-m)}\right)^s - mP_t} dp. \quad (5.64)$$

If we integrate each side of the expression, we obtain

$$\frac{1}{mN} dt = \frac{t}{mN} = \int_{P_0}^{P_F} \frac{1}{1 - \left(1 - \frac{P_t}{1+P_t(1-m)}\right)^s - mP_t} dp \quad (5.65)$$

The above integral cannot be solved in general for any value of s and m . Nonetheless, note that this analysis still provides essential information since (1) it permits calculating particular expressions of the takeover time and (2) enables the derivation of the conditions for the extinction of starved niches. With the aim of showing some particular cases of the takeover time in tournament selection, the next section provides (1) a closed-form solution of the integral for problems with a single niche and any selection pressure s and (2) a closed-form solution for problems with two niches ($m = 2$) and tournament size $s = 2$, since $s = 2$ is the lowest pressure that can be applied.

Particular Expressions of the Takeover Time for Tournament Selection

In this section, we use equation 5.65 to derive some particular expressions of the takeover time. Expressions for other tournament sizes and niche sizes can be computed by replacing the corresponding values into equation 5.65.

Number of Niches $m=1$

Replacing $m = 1$ into equation 5.65 we obtain the following expression

$$t = N \int_{P_0}^{P_F} \frac{1}{1 - (1 - P_t)^s - P_t} dp = \quad (5.66)$$

$$= \int_{P_0}^{P_F} \frac{1}{1 - P_t} dp + \int_{P_0}^{P_F} \frac{(1 - P_t)^{s-2}}{1 - (1 - P_t)^{s-1}} dp = \quad (5.67)$$

$$= N \left[\ln \left(\frac{1 - P_0}{1 - P} \right) + \frac{1}{s-1} \ln \left[\frac{1 - (1 - P)^{s-1}}{1 - (1 - P_0)^{s-1}} \right] \right]. \quad (5.68)$$

Thence, the takeover time of cl_b for tournament selection is

$$t_{TS_{m=1}}^* = N \left[\ln \left(\frac{1 - P_0}{1 - P_F} \right) + \frac{1}{s-1} \ln \left[\frac{1 - (1 - P_F)^{s-1}}{1 - (1 - P_0)^{s-1}} \right] \right], \quad (5.69)$$

which depends on the initial P_0 and final P_F proportion of classifiers, and it is always positive regardless of the values of P_0 and P_F . Therefore, if the problem contains a single niche—a situation that is nonrealistic in real-world problems—the best classifier always will take over the niche (Orriols-Puig et al., 2007d).

Number of Niches $m=2$, Tournament Size $s=2$

Substituting $m = 2$ and $s = 2$ into equation 5.65, we obtain

$$t = 2N \int_{P_0}^{P_F} \frac{1}{1 - \left(1 - \frac{P_t}{1-P_t}\right)^s - 2P_t} dp = \quad (5.70)$$

$$= \int_{P_0}^{P_F} \frac{(1 - P_t)^2}{(1 - P_t)^2 - (1 - 2P_t)^2 - 2P_t(1 - P_t)^2} dp = \quad (5.71)$$

$$= \int_{P_0}^{P_F} \frac{(1 - P_t)^2}{P_t^2(1 - 2P_t)} dp = \int_{P_0}^{P_F} \frac{1}{P_t^2} dp + \int_{P_0}^{P_F} P_0 \frac{1}{1 - 2P_t} dp = \quad (5.72)$$

$$= \left[-\frac{1}{P_t} - \frac{1}{2} \ln(1 - 2P_t) \right]_{P_0}^{P_F} = 2N \left[\frac{1}{P_0} - \frac{1}{P_F} + \frac{1}{2} \ln \frac{1 - 2P_0}{1 - 2P_F} \right]. \quad (5.73)$$

Then, the takeover time for tournament selection for $m = 2$ and $s = 2$ is

$$t_{TS_{m=2,s=2}} = 2N \left[\frac{1}{P_0} - \frac{1}{P_F} + \frac{1}{2} \ln \frac{1 - 2P_0}{1 - 2P_F} \right], \quad (5.74)$$

which depends linearly on the initial and the final proportion of the best classifier in the population and logarithmically on the difference between them. However, it does not depend on any scale between the fitness of cl_b and cl_o . Moreover, as $P_F > P_0$, the takeover time is always positive; this indicates that the best classifier always will be able to takeover its niche, regardless of the imbalance ratio of the problem. Note that this analysis has been made for the lowest possible selection pressure. Therefore, this conclusion can be extended to any tournament size for problems with two niches.

Finally, we compare the conclusions provided by this analysis with those obtained with proportionate selection. In 5.8.2, we theoretically demonstrated that, with proportionate selection, the best classifier would not be able to take over its niche if $\rho \geq 0.5$ for problems with two niches (see equation 5.48). Thus, tournament selection appears to be more robust in highly imbalanced data sets when the fitness separation between accurate and over-general classifiers is low.

This subsection provided some specific expressions of the takeover time for tournament selection. Note that, although the general closed-form solution could not be extracted, the analysis is still crucial since it enables us to detect the critical bounds of the system behavior when learning from imbalanced domains, which are derived in the next subsection.

Conditions for the Extinction of Starved Niches under Tournament Selection

To derive the conditions for niche extinction for tournament selection, we consider the differential equation obtained during the derivation of the model (see equation 5.57) and require that the increase in the numerosity of the best classifier be negative. That is,

$$\frac{1}{m} \left[1 - \left(1 - \frac{n_{b,t}}{n}\right)^s \right] - \frac{n_{b,t}}{N} < 0, \quad (5.75)$$

which can be rewritten as

$$1 - m \frac{n_{b,t}}{N} < \left(1 - \frac{n_{b,t}}{n}\right)^s. \quad (5.76)$$

This expression depends on the number of niches m , the number of accurate classifiers in the niche $n_{b,t}$, the niche size n , and the population size N . Note that the left-most term decreases linearly with m , whilst the right-most term decreases exponentially with s . Therefore, the condition will be satisfied, i.e., the best classifier will not be able to take over its niche, for low values of s combined with moderate and large number of niches m .

Thence, different from proportionate selection, the imbalance ratio does not appear as a decision variable in the extinction model for tournament selection. This is normal since we assumed that the parameters of classifiers are accurately estimated; thus, the error of cl_o will be always greater than the error of cl_b , and tournament selection will select cl_b when both classifiers compete in the same tournament. Thereupon, the extinction condition is basically guided by the selection pressure s —that is, the number of classifiers that will participate in the tournaments—and the size of the niches, which models the effect of population-based deletion.

5.8.4 Experimental Validation of the Takeover Time Models

Here, we experimentally validate (1) the theoretical models of takeover time and (2) the conditions for the extinction of starved niches for both proportionate and tournament selection. For this purpose, we ran XCS on the parity problem setting the number of niches m of the system. We initialized the population with $P_0 \cdot N$ copies of maximally accurate classifiers (equally distributed in the m niches) and $(1 - m \cdot P_0) \cdot N$ copies of an over-general classifier that appeared in all the niches. The prediction error of the over-general classifier was deterministically calculated as $\epsilon_{ovg} = \epsilon_0 \left(\frac{\rho}{\alpha}\right)^\nu$. In our experiments, we set $\alpha=0.1$ and $\nu = 10$. Note that varying ρ , we are changing the fitness scaling between cl_o and cl_b . This could be equivalently done by maintaining ρ and varying ν , as done by Kharbat et al. (2005).

Figure 5.9 shows the evolution of the proportion of the best classifier in one of the niches for RWS and (a) $m=1$, (b) $m=2$, and (c) $m=3$ number of niches. The empirical data are averages over 50 runs. According to the model, we computed the proportion of the best classifier in the population. Therefore, the average of this proportion would tend to $1/m$, as approximately the same resources would be placed in each niche. In the figure, we plot the proportion of the best classifier in the niche, which ranges from 0 to 1. Figure 5.9 shows a perfect match between the theory and the empirical results. It also shows that, as predicted by the models derived in section 5.8.3, the ratio of the accuracy of the over-general classifier to the accuracy of the best classifier is a crucial aspect. For the problem with two niches, the best classifier could not take over its niche if $\rho \geq 0.5$ (see figure 5.9(b)), as predicted by the niche extinction model provided in equation 5.47. This behavior was also present in the problem with three niches 5.9(c), where neither $\rho = 0.4$ nor $\rho = 0.5$ let the best classifiers take over their niches.

Figure 5.10 shows the evolution of the proportion of the best classifier in the niche for TS and (a) $m=1$, (b) $m=2$, and (c) $m=3$ number of niches. For the problem with one niche, we depict the selection pressures of $s=\{1,2,10\}$. Figure 5.10(a) shows a perfect match between the theoretical model and the experiments. Tournament selection is not influenced by the ratio of the accuracy of the best classifier to the accuracy of the over-general classifier. That is, we ran experiments with different values of ρ , obtaining equivalent results to those plotted in the figure. Moreover, it is also shown that increasing the tournament size results in faster convergence times. For $s>10$, the takeover time barely decreases (these curves are not depicted in the figure for clarity).

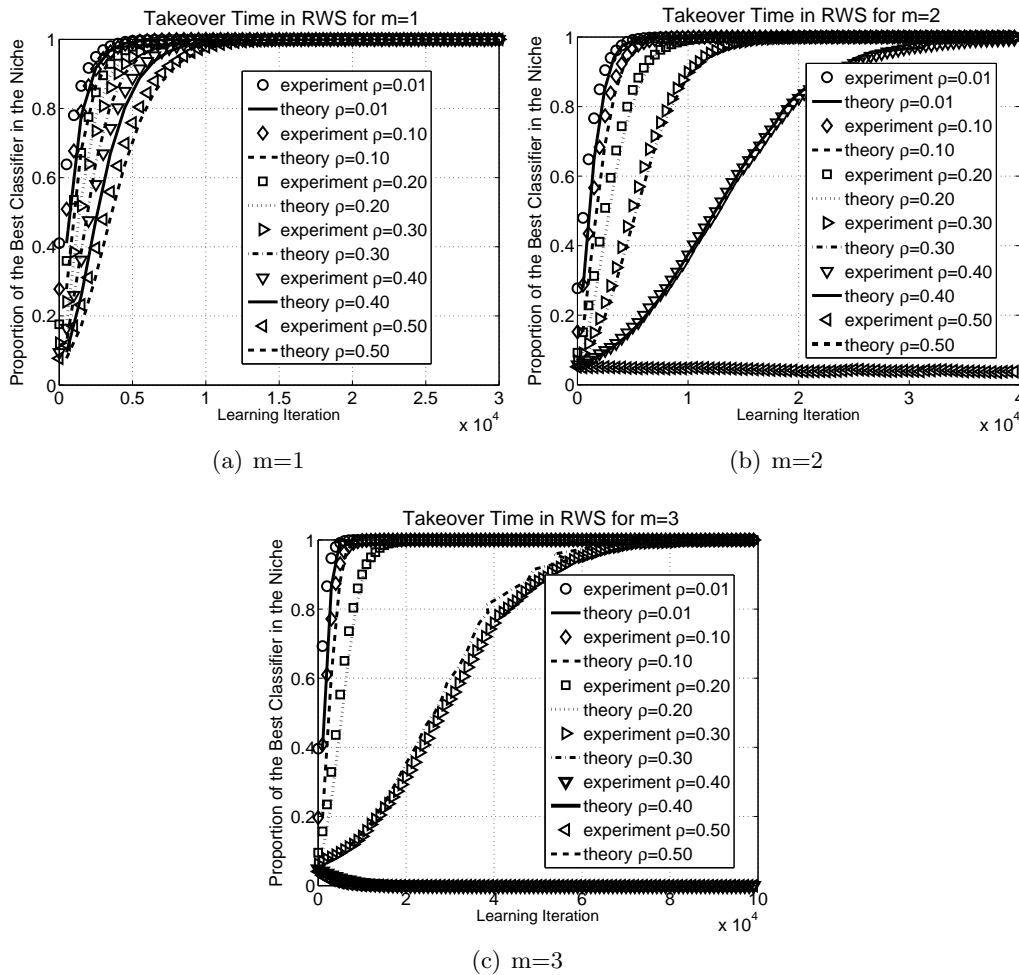


Figure 5.9: Takeover time in proportionate selection for (a) $m=1$, (b) $m=2$, and (c) $m=3$ and $\rho = \{0.01, 0.10, 0.20, 0.30, 0.40, 0.50\}$.

For the problems with two and three niches, we plot the evolution of the best classifier under tournament selection for $s=2$ and $s=3$ (see figures 5.10(b) and 5.10(c)). The theoretical model was calculated for each case by replacing m and s into equation 5.65 and solving the integral. Again, the theory nicely approximates the experimental results. For $m=2$, the best classifier can take over its niche, even for the lowest possible selection pressure. For $m=3$, the best classifier can take over its niche only if $s \geq 3$. This experimental evidence agrees with the niche extinction model supplied in equation 5.76. That is, as the number of niches increases, the selection pressure needs to be stronger to let the best classifier emerge, regardless of the initial proportion of this classifier in the population.

The overall analysis also permits comparing the two selection approximations and relating them to the class-imbalance problem. For low values of ρ , that is, when the fitness of the best classifiers is much higher than the fitness of the over-general classifiers, proportionate selection can yield the fastest takeover times. Therefore, proportionate selection appears as the most

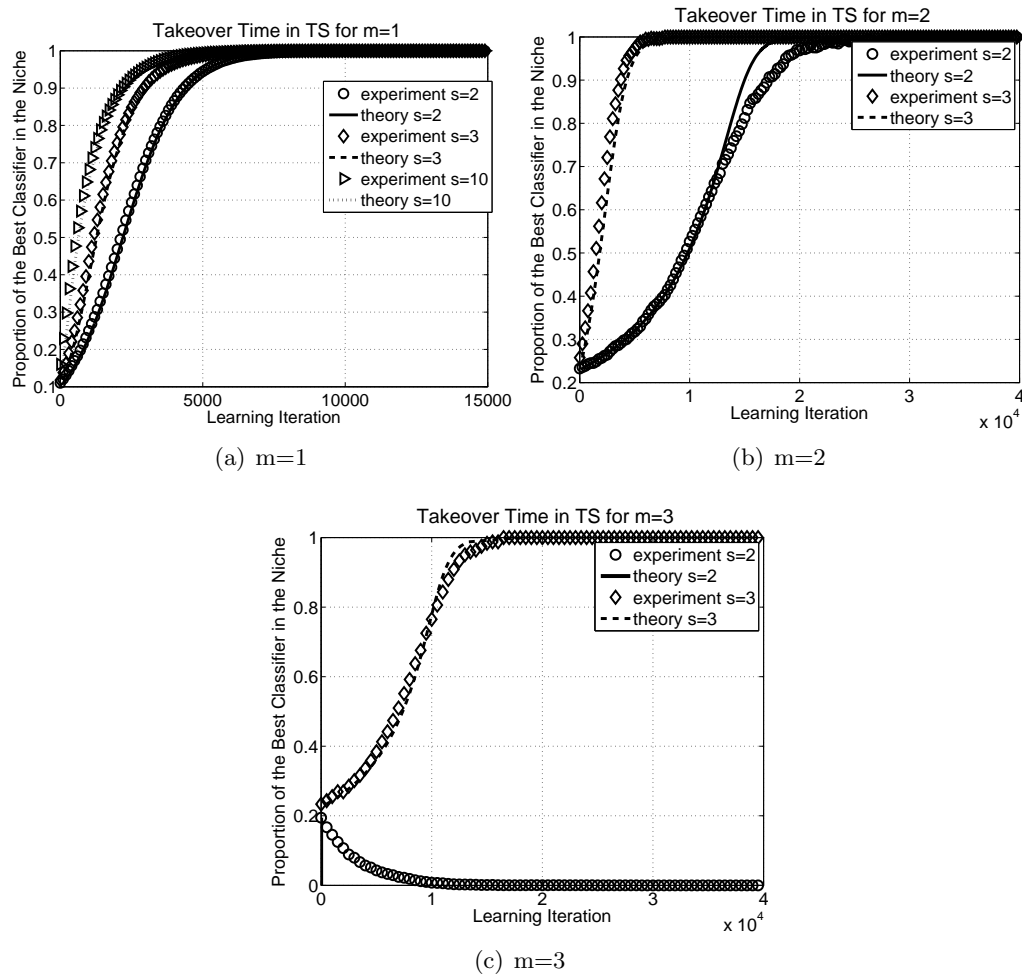


Figure 5.10: Takeover time in tournament selection for (a) $m=1$, (b) $m=2$, and (c) $m=3$.

appealing alternative for domains in which there is a high separation among the fitness of accurate and inaccurate rules. As pointed out by Kharbat et al. (2005), in balanced domains, this can be easily done by tuning the fitness pressure ν . Nonetheless, in imbalanced domains, the error of over-general classifiers decreases with the imbalance ratio (see equation 5.9). In these cases, proportionate selection may promote the existence of over-general classifiers. Thence, tournament selection appears to be the most robust selection scheme for imbalanced domains, provided that s is sized properly. A combination of both schemes seems to be an attractive alternative to deal with new real-world problems with unknown characteristics.

With the study of the takeover time and the conditions of extinction of starved niches, we completed the analysis of the different facets proposed in section 5.3.4. Facetwise models have provided key insights and points of view of the problem. The next section integrates all these models, provides configuration guidelines based on the lessons learned from them, and shows that, following these recommendations, XCS is able to solve highly imbalanced problems that previously eluded solution.

5.9 Lessons Learned from the Models

In this section, we use qualitative arguments to integrate the different models and extract lessons from the whole design decomposition and facetwise analysis. Then, we use the derived facetwise analysis as a tool for designing a set of guidelines that should be satisfied to warrant that XCS will be able to extract key knowledge from rare classes. We experimentally show that, if the system is configured according to these recommendations, XCS is able to solve problems with large imbalance ratios that previously eluded solution. More specifically, we show that XCS with a proper configuration solves the imbalanced multiplexer problem with large imbalance ratios, for which we showed that first generation XCS failed to discover the minority class in section 5.2.2.

5.9.1 Patchquilt Integration of the Facetwise Models

Along this chapter, we have studied the five subproblems, identified by the design decomposition, that may impair XCS from learning from rare classes. Now, we integrate the different models in a general framework and highlight the lessons derived from each particular model and, in general, from their interaction. This integration permits us to (1) identify under which cases XCS will not be able to learn from the minority class and (2) establish configuration recommendations to ensure convergence if possible. To achieve this, we revisit the models from the most restrictive one to the less restrictive one, setting the three steps that need to be guaranteed to ensure convergence and pointing out several configuration guidelines.

1. Niche extinction models (see section 5.8) set the conditions on the maximum imbalance ratio admitted by XCS to create key knowledge from the minority class for proportionate and tournament selection (see equations 5.47 and 5.76). If the requirements are met, takeover time models predict the convergence time inside each niche, that is, the number of learning iterations that an accurate, maximally general representative will need to take over its niche. Satisfying the requirements identified by the extinction time models is a necessary but not sufficient condition. That is, these models indicate that, if the identified restrictions are met, representative classifiers will take over their niche. Therefore, we have to guarantee that, at some point, these representatives are fed into the population.
2. Classifier parameters have to be correctly estimated by one of the methods presented in section 5.4. Otherwise, XCS will not be able to distinguish between over-general and accurate classifiers. We experimentally showed that the Widrow-Hoff rule provided very accurate estimates of the parameters if β was properly set. For this reason, we took this approach in our experiments and proposed an heuristic that automatically sets the value of β ensuring that the error estimate of over-general classifiers is close to the theoretical value (see section 5.6.5).
3. If the above two conditions are satisfied, we can ensure convergence by either (1) sizing the population according to the imbalance ratio or (2) setting θ_{GA} depending on the imbalance ratio according to the models evolved in sections 5.5 and 5.6. It is worth noting that the models work independently of whether covering is able to provide the initial population with schemas of starved niches, as identified in section 5.4.

In the next section, we show that, if the recommendations derived from the models are followed, XCS can solve extremely imbalanced data sets.

5.9.2 Solving Problems with Large Imbalance Ratios

In this section, we take again the imbalance multiplexer problem and show that the lessons extracted from the facetwise analysis enable us to properly configure XCS, letting the system learn from imbalance ratios that previously eluded solution. That is, in section 5.2.2 we showed that XCS was not able to extract the key knowledge from rare classes when $ir > 32$. Now, we run the same experiments and illustrate that, with the better understanding acquired along the facetwise analysis, we can set XCS so that it can solve the multiplexer problem with extremely large imbalance ratios; in particular, we solve the problem for $ir = 1024$.

Before proceeding with the analysis of the results, we first explain how the recommendations have been followed to configure XCS. We took the default configuration as a starting point (see section 5.6.5), but we used the typical deletion scheme of XCS, set $N = 1000$ and probability of crossover $\chi = 0.8$, and employed tournament selection with $\sigma = 0.4$ (that is, 40% of the classifiers in the action set participate in the tournament). This also corresponds to the configuration used in section 5.2.2. Besides, the configuration satisfied the conditions required by the different models, that is:

1. As we used tournament selection, we need to satisfy the condition that

$$1 - m \frac{n_{b,t}}{N} > \left(1 - \frac{n_{b,t}}{n}\right)^s, \quad (5.77)$$

to ensure that the best classifier will take over its niche (note that we change the inequality with respect to equation 5.76 since, now, we require that the best classifier take over its niche). From this equation, we know that $N=1000$ and $m=32$, but $n_{b,t}$ and n are unknown. We assume the worst case, that is, that we only have a best classifier per niche. Thence, $n_{b,t} = 1$ and $n = 1000 - 32 = 968$. Substituting these parameters into the equation, we obtain that $s \geq 29$; this condition is satisfied since 40% of the n classifiers in each niche are selected to be in the tournament. This condition is also satisfied for larger values of $n_{b,t}$. Therefore, this indicates that, if discovered, representatives of starved niches will take over their niches. It is worth noting that similar results can be achieved with proportionate selection.

2. The classifier parameters are estimated according to the Widrow-Hoff rule, but setting β appropriately so that the error of over-general classifiers does not decrease rapidly to zero. For this purpose, we used the heuristic method mentioned in the previous section.
3. Finally, to guarantee that all the niches receive the same number of genetic opportunities, and so, warrant that representatives of starved niches will be created regardless of the imbalance ratio, we set $\theta_{GA} = n \cdot m \cdot ir$ as proposed in section 5.7.

Figure 5.11 plots the results obtained by XCS in the imbalanced 11-bit multiplexer problem with imbalance ratios ranging from $ir = 1$ to $ir = 1024$. More specifically, figure 5.11(a) illustrates the evolution of the proportion of the optimal population %[O] achieved by XCS. That

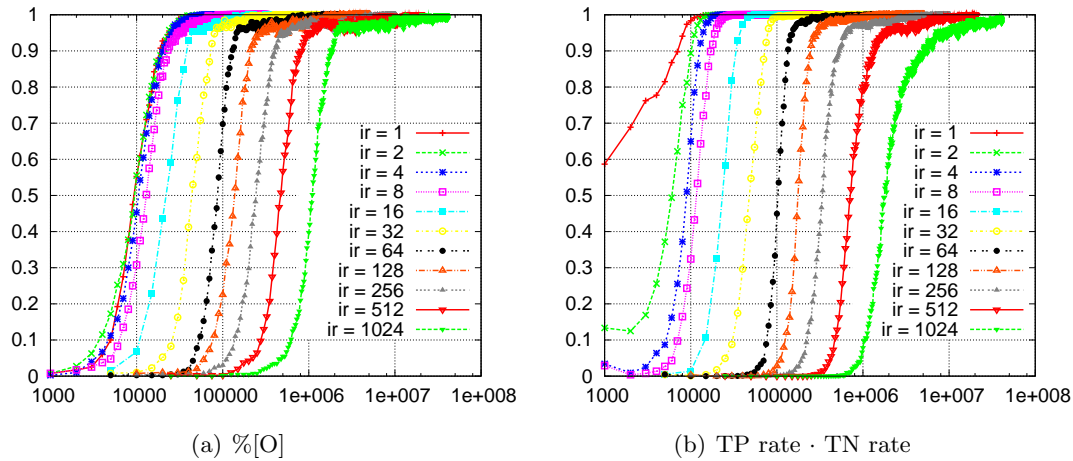


Figure 5.11: Evolution of (a) the proportion of the optimal population and (b) the product of TP rate and TN rate in the 11-bit multiplexer with imbalance ratios ranging from $ir=1$ to $ir=1024$.

is, XCS was expected to evolve 32 optimal classifiers, each one representing a different niche. In this way, we measured the capacity of XCS to generalize and obtain the best representative of each niche at high imbalance ratios. Moreover, figure 5.11(b) depicts the evolution of the product of TN rate and TP rate. Therefore, in addition to measuring whether XCS evolved the optimal population, this figure visualizes whether the instances of the two classes were correctly predicted by the system. Note that we tested extremely imbalanced problems with imbalance ratios up to $ir = 1024$.

Figure 5.11(a) shows that XCS was able to obtain 100% of the optimal population at any imbalance ratio with the same population size. This indicates that class imbalances did not reduce the generalization capabilities of XCS. Similarly, figure 5.11(b) illustrates that the product of TP rate and TN rate raised to 100% for all the tested imbalance ratios. Notice that before the analysis, XCS could only solve the problem for $ir \leq 32$. Hence, the lessons extracted from the design decomposition served to increase our understanding of the system and solve much more complex problems without introducing new mechanisms to the system.

The whole experimental and theoretical analysis performed herein highlights the importance of, previously to designing new approaches to enhance a system whose behavior is only partially understood, really comprehending the underlying problems of the learning architecture. In this way, better approaches that focus on the actual problems of the system can be designed more effectively. Here, we showed that the increased understanding provided by the facetwise analysis enabled first generation XCS to solve problems that seemed to be initially intractable.

5.10 Summary and Conclusions

In this chapter, we analyzed the behavior of XCS in domains that contain rare classes. As XCS learns a set of distributed solutions online, we investigated whether the system may lose, or may

never discover, some sub-solutions whose representative examples are infrequently sampled to the system. XCS learning is driven by the interaction of several components, which introduces complexity to the derivation of models that explain the behavior of the system in its whole. For this reason, we followed the design decomposition approach to study the effect of class imbalances on different components of XCS. That is, we decomposed the problem of learning from imbalanced domains into five subproblems and derived simpler, tractable models that focused on explaining the behavior of concrete parts of the system assuming that the other elements were functioning in an ideal manner. This enabled us to highlight several aspects—mainly related to classifier evaluation, and creation, maintenance and growth of representatives of starved niches—that were critical to guarantee that XCS extract key knowledge from rare classes. In addition, the patchquilt integration of all these models permitted us to draw the *domain of applicability* of XCS in imbalanced domains. We derived critical bounds on the system behavior, identifying the sweet spot where XCS could scalably and efficiently solve problems with class imbalances. Moreover, the study resulted in several recommendations on the system configuration to deal effectively with rare classes. Finally, we showed that all the insights provided by this analysis served to solve new complex problems with high imbalance ratios which previously eluded solution. As example, we empirically demonstrated that XCS was able to solve the 11-bit multiplexer problem with large degree of class imbalance provided that the system was properly configured according to the guidelines indicated by the models.

The importance of the lessons extracted from the whole analysis goes beyond the application of XCS to imbalanced domains. The analysis sets the conditions required to ensure complete solution sustenance—i.e., discovery, maintenance, and growth of all niches of the system—in problems where some niches are activated with less frequency than other niches. This is a common characteristic in real-world classification problems that contain continuous attributes, in which, although not having large imbalance ratios, there may be small regions of instances of one class for which the system needs to evolve starved niches. A deeper discussion about this aspect is postponed to chapter 7, where LCSs are applied to real-world classification problems.

Finally, let us highlight that, as we applied a design decomposition principle, the provided analysis is not restricted to XCS. In fact, we developed a framework in which several subproblems were analyzed separately and simplified models were provided. In all the derived models, we tried to keep the analysis as simple as possible and used intuitive arguments to patch the pieces together. This supplied high flexibility and power to the theoretical framework, which can be adapted with low cost to model other Michigan-style LCSs or other online learning architectures that are based on a competition-collaboration scheme. For this purpose, models that are affected by the architecture change may be revisited and plugged again into the theoretical framework. Other models may still be valid; for example, takeover time models may still be accurate for most of the Michigan-style LCSs. In the next chapter, we illustrate this, and carry over the design decomposition framework developed for XCS to UCS.

Chapter 6

Carrying over the Facetwise Analysis to UCS

In the previous chapter, we decomposed the problem of learning from imbalanced domains in *five elements* that need to be satisfied by any LCSs to efficiently and scalably extract accurate models from rare classes. Then, we centered on XCS and, with little algebra effort, we developed facetwise models for each one of these elements. Although the models were particularly designed for XCS, we claimed that the framework could be applied to other LCSs for two main reasons. The first reason is due to the simplification and abstraction effort taken when deriving the models and the use of qualitative arguments to patch the pieces together. That is, since the analysis was kept as abstract as possible, considering the general learning architecture and avoiding going into too specific details of XCS, some of the models can be applied to other Michigan-style LCSs. The second reason comes implicitly with the design decomposition methodology; that is, since each facet was analyzed considering that the others behave in an ideal manner, changes that only affect one of the facets could be incorporated by rewriting the models of the corresponding facet and plugging the new model into the general framework. This gives an important advantage of facetwise analysis with respect to global models, in which, probably, a little change in the system architecture may require to rewrite the totality of the model.

In this chapter, we take advantage of the flexibility of the framework provided in the previous chapter and show that, with little changes on some of the facets, the behavior of UCS can be easily modeled using the same ideas employed for XCS. We first recall the design decomposition with the list of elements that should be followed by any LCSs to solve class-imbalanced problems. Then, we review each subproblem for UCS. We show that some of the models developed in the previous section, such as the takeover time models, can be directly applied to UCS with no modifications. In other cases, such as the starved niches generation models, new theoretical derivations need to be done and plugged into the general framework. Therefore, we demonstrate the “*plug and play*” capabilities that design decomposition and facetwise analysis provide.

The remainder of this chapter is structured as follows. Section 6.1 reviews the design decomposition presented in the previous section, which identified five main elements or subproblems, and intuitively analyzes whether UCS can solve the five subproblems. Then, each of these five subproblems gets one of the subsequent sections, regardless of whether the original models provided in the previous chapter are still valid for UCS. Therefore, section 6.2 studies whether

UCS can obtain reliable parameter estimates in imbalanced domains, section 6.3 and section 6.4 model the initialization and generation of classifiers of the minority class, section 6.5 studies the effect of occurrence-based reproduction, and section 6.6 revisits the takeover time models. Note that most of these sections are very concise since they use the models derived in the previous chapters. All these models are integrated in section 6.7. Finally, section 6.8 summarizes and concludes the chapter.

6.1 Design Decomposition for UCS

In the previous chapter, we decomposed the problem of learning from imbalanced domains in LCSs in five elements that need to be guaranteed, i.e.,

1. Estimate the classifier parameters correctly.
2. Analyze whether representatives of starved niches can be provided in initialization.
3. Ensure the generation and growth of representatives of starved niches.
4. Adjust the GA application rate.
5. Ensure that representatives of starved niches will take over their niches.

Here, we follow the same decomposition to study the behavior of UCS in imbalanced domains. As proceeds, we first intuitively discuss the differences between XCS and UCS in these types of problems. Then, each of the subsequent sections analyzes of these elements in detail.

Estimate the classifier parameters correctly. Having accurate estimates of the classifier parameters was identified as one of the most important elements that need to be guaranteed in imbalanced domains. That is, the system relies on these estimates to distinguish between over-general and accurate classifiers; therefore, poor parameter estimates may thwart the competition between over-general and accurate classifiers. XCS used a temporal widowed average to update the classifier parameters. We showed that this may result in poor estimates if the size of the window is not set properly. In UCS, the classifier’s accuracy—which is equivalent to the classifier’s error in XCS—is computed as a the true average of the number of examples that have been correctly classified over the total number of examples that the classifier has matched. Intuitively, this seems to indicate that the parameters of over-general classifiers would not oscillate as abruptly as in XCS. A further study of the parameter update procedure is conducted in section 6.3.

Analyze whether representatives of starved niches can be provided in initialization. Once the proper evaluation of classifier parameters is ensured, we are concerned about whether UCS is able to provide representative schemas of starved niches in the beginning of the run. For XCS, we showed that the probability that the covering operator supplies the population with schemas of starved niches decreases exponentially with the imbalance ratio. This is because, in XCS, covering is applied to the match set, creating classifiers that can predict any class. On the other hand, UCS applies covering in the correct set as the class of the input example is also provided at each learning iteration. Thence, the covering operator in UCS only generates classifiers

that predict the class of the sampled input instance. In section 6.4, we analyze whether the new covering scheme in UCS is able to provide the initial population with schemas of the under-sampled class.

Ensure the generation and growth of representatives of starved niches. After initializing the population, the GA is responsible for obtaining high accurate classifiers that represent the different niches. As in XCS, intuition seems to indicate that the occurrence-based reproduction of UCS may favor both over-general classifiers and representatives of nourished niches, which may go in detriment of representatives of starved niches. Section 6.4 develops this aspect in detail.

Adjust the GA application rate. As in XCS, varying the application rate of the genetic algorithm influences the genetic opportunities that the different niches receive. Section 6.5 studies the impact of varying the frequency of application of the GA.

Ensure that representatives of starved niches will take over their niches. Finally, we analyze whether the best classifiers will be able to take over their niches. For this purpose, section 6.6 validates the applicability of the takeover time models to UCS and relates these models to the maximum imbalance ratio up to which the representatives of starved niches will be able to take over their niche.

As proceeds, each of the five elements is analyzed in detail and the derived models are validated with the imbalanced parity problem. Section 6.7 unifies all the models, emphasizing the lessons extracted from all them. Finally, as done for XCS, we show that following the recommendations provided by the models, UCS is able to solve the 11-bit multiplexer problem with large imbalance ratios.

6.2 Estimation of Classifier Parameters

The first element of the design decomposition that needs to be satisfied is that the parameters of classifiers be correctly estimated. In XCS, we detected that the original parameter update procedure may provide poor estimates of the parameters of over-general classifiers in problems with large imbalance ratios. However, note the difference between the parameter update procedure in both systems. XCS computes the quality of a rule by means of a fitness based on the error of the prediction of the rule. This error is updated online by a credit apportionment algorithm that performs a windowed average of the last received rewards. Conversely, as UCS is specialized for supervised learning tasks, the fitness of a classifier is based on its classification accuracy, which is computed as the true average of the number of examples correctly classified with respect to the total number of examples matched by the classifier. Then, the fitness is computed from the relative accuracy of each classifier¹. Therefore, the larger the number of examples matched by the rule, the more accurate the estimation of the classifier's accuracy, and, consequently, the fitness estimate.

To illustrate the differences between the parameter update procedures of XCS and UCS, we ran the same experimentation proposed in section 5.5 but with UCS. Figure 6.1 shows a

¹In all the experiments conducted along the subsequent chapters, we use UCS with fitness sharing.

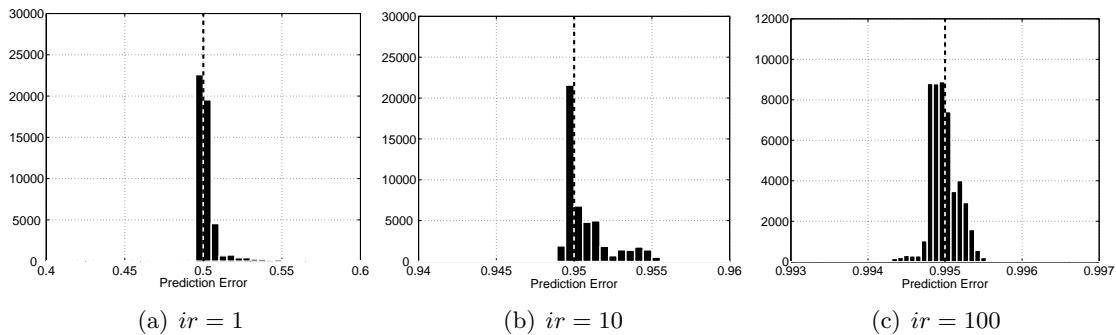


Figure 6.1: Histogram of the error of the most over-general classifier in UCS for $ir = \{1, 10, 100\}$.

histogram of the accuracy estimate of the most over-general classifier along a complete run. The vertical line depicts the theoretical value of the accuracy. As expected, the empirical accuracy estimate of the most over-general classifier matches perfectly with the theoretical one most of the time. Note the difference of these results with respect to those provided by XCS. In XCS, the parameters of over-general classifiers oscillated as they received infrequently negative rewards. In UCS, the parameters of over-general classifiers are stabilized as the classifier receives more updates.

In summary, in this section we discussed and empirically showed that the parameters update procedure of UCS enables the system to obtain reliable estimates. Thence, the remainder of the analysis is conducted assuming accurate parameter estimates.

6.3 Supply of Schemas of Starved Niches in Population Initialization

In this section, we analyze the first element that leads to the creation of representatives of the different niches: population initialization. In the previous chapter, we identified that the covering mechanism of XCS impaired the system from initializing the population with correct schemas of starved niches. This was mainly due to the exploration regime of XCS, which, given an unlabeled input example, analyzed the consequences of each possible class. Therefore, we assumed a covering failure in XCS, and derived the models for the remaining elements considering this covering failure.

Differently from XCS, the supervised learning architecture of UCS only applies covering on the correct set. That is, covering only creates classifiers that predict the class of the sampled example. Therefore, covering will be triggered on the first instances of the each one of the classes, including the rare class, regardless of the imbalance ratio of the learning data set. Consequently, we can calculate the probability that a minority class instance is covered by, at least, one classifier

in the population as

$$P(\text{cover}) = 1 - \left[1 - \left(\frac{2 - \sigma[P]}{2} \right)^\ell \right]^{\frac{N}{ir}}, \quad (6.1)$$

where ℓ is the input length, N is the population size, and $\sigma[P]$ is the specificity of the population. Note that the only difference with respect to the corresponding equation in XCS (see equation 5.11) is that, in UCS, the power of the term in brackets of the right-most expression is N/ir instead of N . This modification is because the number of minority class classifiers provided by covering is directly proportional to the number of instances of the minority class that have been sampled to the system.

Provided that the probability of activating covering is $1 - P(\text{cover})$, and recognizing that $(1 - r/n)^n \approx e^{-r}$, we can derive that the probability of activating covering, having sampled a minority class instance, is

$$P(\text{activate cov. on. min.}) = 1 - P(\text{cover}) \approx e^{-\frac{N}{ir}} \cdot e^{-\frac{\ell\sigma[P]}{2}}, \quad (6.2)$$

which decreases exponentially with the ratio of the population size to the imbalance ratio N/ir and, in a higher degree, with the condition length and the initial specificity. Notice that N/ir decreases linearly with the sampling frequency of the minority class. Therefore, the capabilities of covering to provide accurate schemas of the minority class do not depend directly on the imbalance ratio, but on the initial population specificity.

The analysis performed in this section showed that, differently from XCS, the success of the covering operator in supplying classifiers representing correct schemas of the minority class does not depend on the imbalance ratio. Although these positive results, in the next section we derive the models for creation of new representative classifiers of the minority class under the assumption that the population has not any representative of starved niches, as done for XCS. As we are assuming the most pessimistic situation, we expect that the models predict an upper bound on the time and the population size required by UCS to solve the problem. Note that, although this would not result in a precise model of the population size, it will bound the maximum population size required to solve problems with large imbalance ratios, which still provides critical information about UCS behavior in imbalanced domains.

6.4 Generation of Classifiers in Starved Niches

In this section, we study the conditions that must be satisfied to enable the GA to create representatives of starved niches. Therefore, as done for XCS, we derive models that predict the time until creation and extinction of these representatives. With these models, we write population size bounds to warrant the existence and growth of representatives of starved niches. As proceeds, we first review the assumptions of the model, which are equivalent to those considered for XCS, and then revisit the models for each element.

6.4.1 Assumptions for the Model

The models are developed under the same assumptions considered for XCS, i.e., (i) covering has not provided any representative of starved niches, (ii) mutation is the only operator that guides the genetic search (i.e., we do not consider crossover), (iii) the GA is applied at the end of each learning iteration (i.e., $\theta_{GA} = 0$), and (iv) the system uses random deletion. Note that the first assumption may not be necessarily true in UCS; in fact, the previous section demonstrated that the covering operator could provide the same number of schemas of the minority class regardless of the imbalance ratio. Nonetheless, we consider this assumption and derive an upper bound of the convergence and population size models. After this, we experimentally study the effect of breaking the three last assumptions.

6.4.2 Creation and Deletion of Representatives of Starved Niches

With the assumptions provided above, we are now in position to derive both the time until creation and the time until deletion of representatives of starved niches. For this purpose, we first calculate the probability to obtain the first accurate representative cl_{min} of the starved niche i , which is represented by a schema with length k_m . This probability will be used to compute the creation time.

As proposed in section 5.6.2, we calculate the probabilities of creating cl_{min} when sampling (i) instances of the minority class and (ii) instances of the majority class. Recognizing that the probability of sampling a minority class instance is $1/(1 + ir)$ and the probability of sampling a majority class instance is $ir/(1 + ir)$, we can write that

$$P(cl_{min}) = \frac{1}{1 + ir}P(cl_{min}|\text{min. inst}) + \frac{ir}{1 + ir}P(cl_{min}|\text{maj. inst}). \quad (6.3)$$

Let us first derive $P(cl_{min}|\text{min. inst})$. When an instance of the minority class is sampled, a niche containing classifiers that predict the minority class will be activated. As we assumed that there are no representatives of starved niches in the population, the correct set will only consist of over-general classifiers. Thence, to create a representative of a starved niche, all the k_m bits of the schema that represents the niche must be correctly set to the values of the niche schema; here, we consider the worst case, and assume that all the k_m bits need to be mutated. Thence, the probability of getting the correct schema is $(\frac{\mu}{2})^{k_m}$. Besides, the class of the classifier cannot be changed. Therefore,

$$P(cl_{min}|\text{min. inst}) = \left(\frac{\mu}{2}\right)^{k_m} \cdot (1 - \mu). \quad (6.4)$$

We follow the same procedure to derive the probability of creating cl_{min} when sampling an instance of the majority class, i.e., $P(cl_{min}|\text{maj. inst})$. When a majority class instance is sampled, a nourished niche containing both representatives and over-general classifiers will be activated. Again, we consider the worst case and assume that we need to change all the k_m bits of the schema, i.e., $(\frac{\mu}{2})^{k_m}$. Furthermore, in this case, the class has to be mutated to the minority class, which will happen with probability $\mu/n - 1$, where n is the number of classes. Thence,

$$P(cl_{min}|\text{maj. inst}) = \left(\frac{\mu}{2}\right)^{k_m} \cdot \frac{\mu}{n - 1}. \quad (6.5)$$

Substituting equations 6.4 and 6.5 into equation 6.3, we obtain that

$$P(cl_{min}) = \frac{1}{1+ir} \left(\frac{\mu}{2}\right)^{k_m} \left[(1-\mu) + \frac{\mu \cdot ir}{n-1} \right]. \quad (6.6)$$

From this formula, we can derive the time required to discover the first representatives of starved niches $t_{cl_{min}}$ as

$$t(cl_{min}) = (n-1) \left(\frac{2}{\mu}\right)^{k_m} \left[\frac{1+ir}{(1-\mu)(n-1) + \mu \cdot ir} \right], \quad (6.7)$$

which depends on the imbalance ratio ir , the probability of mutation μ , and the length of the schema k_m . For highly imbalanced domains, we can consider that $(1-\mu)(n-1) \ll \mu \cdot ir$. Thence, $t(cl_{min})$ increases linearly with $\frac{1+ir}{\mu \cdot ir}$, which becomes nearly constant for large values of ir .

After computing the creation time, we now approximate the deletion time of these representatives. As done for XCS, we consider random deletion. Hence, as two classifiers are deleted at each GA application, we obtain that the time until deletion is

$$t(\text{delete cl}) = \frac{N}{2}, \quad (6.8)$$

where N is the population size. In the next section, we use both equations 6.7 and 6.8 to derive population size bounds that guarantee the discovery, maintenance, and growth of representatives of starved niches under the assumption that covering has not provided any of them.

6.4.3 Bounding the Population Size

We now follow the same steps proposed in section 5.6.4 to derive two population size bounds that ensure (i) that XCS will be able to maintain accurate representatives of starved niches and (ii) that these representatives will receive, at least, a genetic opportunity.

The first bound can be derived by requiring that the deletion time of starved niches representatives be larger than their creation time, i.e.,

$$t(\text{delete } cl_{min}) > t(cl_{min}). \quad (6.9)$$

Using formulas 6.7 and 6.8, the expression can be rewritten as

$$N > 2(n-1) \left(\frac{2}{\mu}\right)^{k_m} \left[\frac{1+ir}{(1-\mu)(n-1) + \mu \cdot ir} \right]. \quad (6.10)$$

Again, note that, for large values of ir , the population size increase is guided by the term $\frac{1+ir}{\mu \cdot ir}$, which remains nearly constant. Consequently, at a certain imbalance ratio, the population size needed to discover representatives of the minority class becomes constant.

We now derive the second bound by requiring that the deletion time of representatives of starved niches be greater than the time until these representatives receive a genetic event, that is,

$$t(\text{delete } niche_{min}) > t(\text{GA } niche_{min}). \quad (6.11)$$

As we assumed that $\theta_{GA} = 0$, a starved niche receives a genetic event every time that it is activated. Since the probability of sampling a minority class instance is $1/(1+ir)$, the time to apply a GA on a starved niche is

$$t(\text{GA } niche_{min}) = (1 + ir). \quad (6.12)$$

Replacing equations 6.8 and 6.12 into equation 6.11, we obtain that

$$N > 2(1 + ir), \quad (6.13)$$

which indicates that the population size has to increase linearly with the imbalance ratio to ensure that representatives of starved niches will receive, at least, a genetic opportunity.

In this section, we derived models that explain the creation and growth of representatives in starved niches. As the models were developed under the assumption of a failure of the covering operator to provide schemas of starved niches—which it is not necessarily the case, as argued in section 6.3—, the models represent an upper bound of the population size required by UCS to solve imbalanced problems. Therefore, the models explain that the population size has to increase, at most, linearly with the imbalance ratio to ensure the discovery and growth of accurate classifiers in starved niches. In the next section, we empirically validate the population size bounds with different configurations of the imbalanced parity problem.

6.4.4 Experimental Validation of the Models

To validate the population size models, we first use a configuration of UCS that satisfies the assumption of the models. Then, we empirically analyze the effect of breaking these assumptions.

Experiments Satisfying the Assumptions

Our first concern is to empirically contrast whether the population size bound derived in equation 6.13 predicts an upper-bound of the population size as the imbalance ratio increases. For this purpose, we performed the same experiments as for XCS (see section 5.6.5). We ran UCS on the imbalanced parity problem with $k = \{1, 2, 3, 4\}$, $\ell = 10$, and $ir = \{1, 2, 4, 8, 16, 32, 64, 128\}$, and we used the bisection procedure to obtain the minimum population size required to solve the problem (see section 5.6.5 for more details about the procedure). The results are averages over 50 runs with different random seeds. UCS was configured so that the initial assumptions were satisfied. Thence, crossover was deactivated ($\chi = 0$), random deletion was used, and the GA was applied every time a niche was activated ($\theta_{GA}=0$). The other parameters were set as $acc_0 = 0.999$, $\nu = 10$, $\mu = 0.04$, $\theta_{del} = 20$, $\delta = 0.1$, $\theta_{sub} = ir$, $P_{\#} = 0.6$, $\beta = 0.2$. We used both proportionate and tournament selection for the GA. We ran UCS during $\{10\,000 \cdot ir, 20\,000 \cdot ir, 40\,000 \cdot ir, 80\,000 \cdot ir\}$ iterations for the parity problem with $k = \{1, 2, 3, 4\}$ respectively; thus, given a problem, we ensured that the system received the same number of genetic opportunities for all imbalance ratios. Finally, to prevent having young over-general classifiers with poorly estimated parameters in the final population, we introduced $5\,000 \cdot ir$ iterations with the GA switched off at the end of the learning process. In the remainder of this analysis, this configuration is referred to as the default configuration.

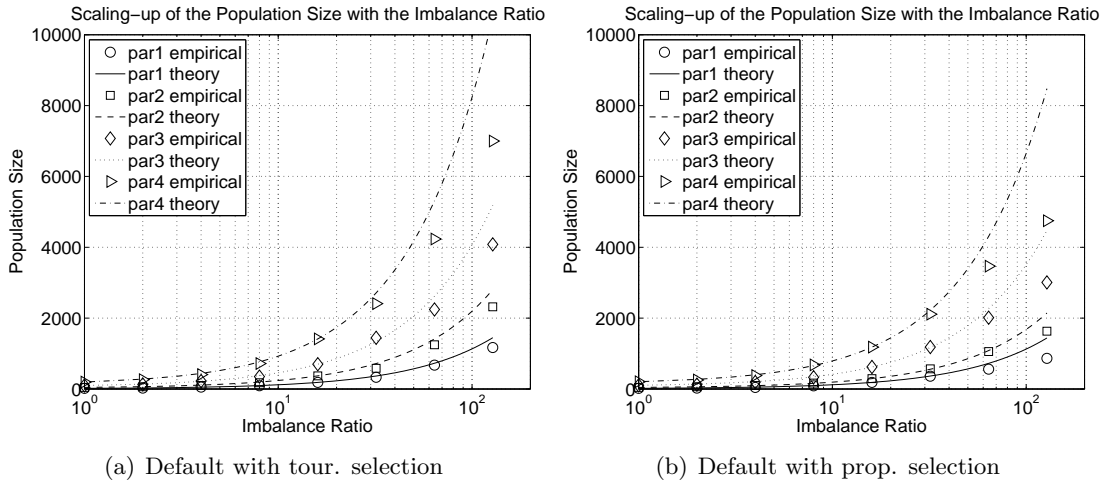


Figure 6.2: Scalability of the population size with the imbalance ratio in the k -parity problem with $k=\{1,2,3,4\}$ and the default configuration with (a) tournament selection and (b) roulette wheel selection. The dots show the empirical results and lines plot linear increases with ir (according to the theory).

Figure 6.2 shows the minimum population size required to solve the parity problem with different building block sizes ($k = \{1, 2, 3, 4\}$) and imbalance ratios from $ir = 1$ to $ir = 128$ for (a) tournament and (b) proportionate selection. For each plot, the points depict the empirical values and the lines show the theoretical bound, that is, they draw a linear increase with the imbalance ratio. Two main conclusions can be extracted from these results. Firstly, note that the theory estimates an upper bound of the population size required by UCS to solve the problem, especially as the imbalance ratio increases. This behavior was already announced in the beginning of this section. The theory was developed with the assumption that the covering operator was not able to provide accurate schemas of starved niches. Nonetheless, section 6.3 showed that the initial supply of schemas of starved niches was independent of the imbalance ratio. For this reason, the theory is approximating an upper bound of the required population size. Secondly, UCS with proportionate selection requires smaller population sizes to solve the parity problems than UCS with tournament selection, especially as the imbalance ratio increases. The Wilcoxon signed-ranks test confirmed that this difference was significant at $\alpha = 0.05$. This may be due to the fact that proportionate selection can produce a stronger pressure toward fit classifier than tournament selection in this particular problem. Nevertheless, we leave further discussion about the selection schemes to section 6.6.

In summary, the experimental analysis pointed out that the theory is an accurate upper bound of the population size of UCS for imbalanced domains when the system is configured so that the underlying assumptions of the model are met. In the next section, we investigate whether this population size bound is still valid when the different assumptions are not satisfied.

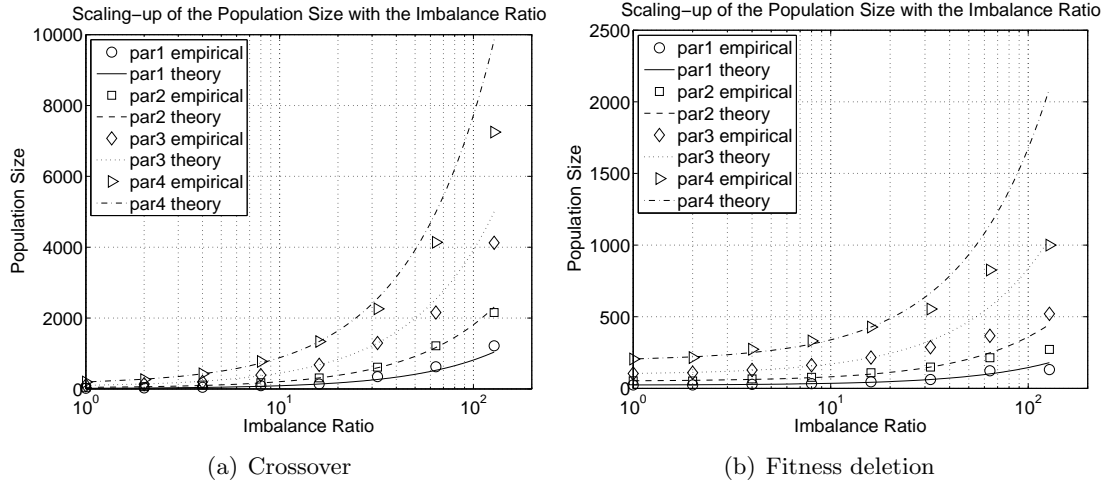


Figure 6.3: Scalability of the population size with the imbalance ratio in the k -parity problem with $k=\{1,2,3,4\}$ and different UCS's configurations that do not satisfy the initial model assumptions: (a) using 2-point crossover and (b) using the correct set size deletion scheme. The dots shows the empirical results and lines plot linear increases with ir (according to the theory).

Impact of Breaking the Assumptions

In this section, we experimentally analyze the impact of breaking two of the initial assumptions. That is, we introduce crossover and the typical deletion scheme of UCS. The impact of breaking the third assumption, i.e., varying the frequency of application of the GA, is further studied in the next section. Figure 6.3 provides the results of running UCS on the same configuration of the parity problem used in the previous subsection, but using (a) two-point crossover, with $\chi = 0.8$ and (b) the typical UCS's deletion scheme, setting $\theta_{del} = 20$ and $\delta = 0.1$. In both cases we used tournament selection.

Several conclusions can be drawn from the comparison of these results with those obtained in the previous section. Firstly, notice that, in both cases, the theory still predicts an upper bound of the minimum population size required to solve the problem, although the initial assumptions are not satisfied. The population size required by UCS when using crossover (see figure 6.3(a)) is equivalent to the population size needed by UCS with the default configuration (see figure 6.2(a)) according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. This indicates not only that the models are still valid when using crossover, but also that the population sizes demanded solving the different configurations of the problem are statistically equivalent to the ones required by UCS without crossover. On the other hand, the population sizes needed for UCS with the usual deletion scheme are statistically smaller than those required by UCS with random deletion, since the deletion scheme protects classifiers that belong to starved niches.

The overall study provided along this section showed that the theory approximates the experiments accurately, even though two of the initial assumptions of the model are not satisfied. In the next section, we investigate the effect of breaking the last assumption; that is, we analyze the effect of varying the frequency of application of the GA.

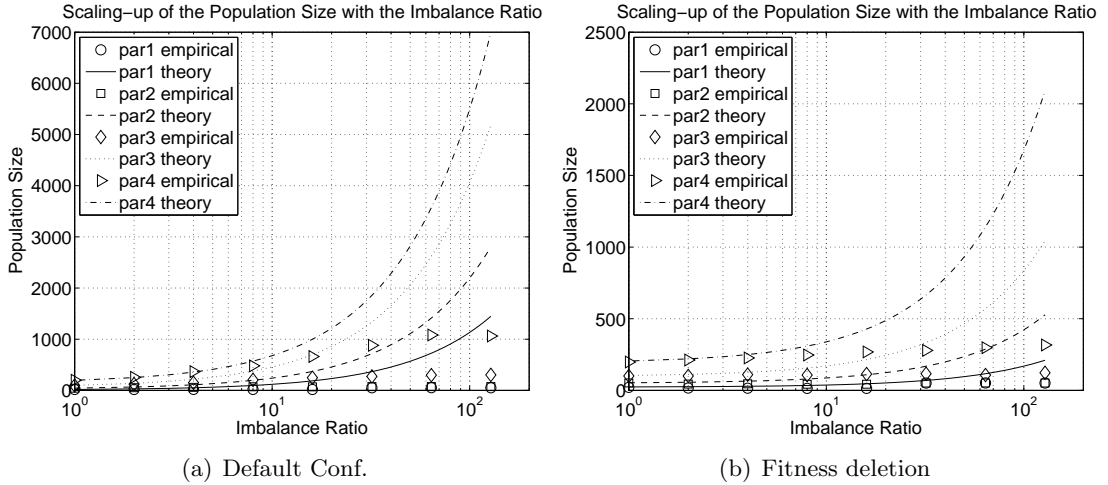


Figure 6.4: Scalability of the population size with the imbalance ratio in the k -parity problem with $k=\{1,2,3,4\}$ and different UCS’s configurations with $\theta_{GA} = n \cdot m \cdot ir$. The points indicate the empirical values of the minimum population size required by UCS. The lines depict the theoretical increase calculated with the previous models, which assumed $\theta_{GA} = 0$.

6.5 Occurrence-based Reproduction

The models developed so far have assumed that any niche received a genetic event every time that it was activated, i.e., $\theta_{GA} = 0$. Due to this occurrence-based reproduction, nourished niches receive a larger number of genetic events than starved niches. Besides, the reproductive opportunities of over-general classifiers with respect to the reproductive opportunities of representatives of starved niches increase linearly with the imbalance ratio. To counterbalance this effect in XCS, section 5.7 showed that if θ_{GA} was set according to the imbalance ratio, i.e.,

$$\theta_{GA} \approx n \cdot m \cdot (1 + ir), \quad (6.14)$$

both starved and nourished niches would receive, approximately, the same number of genetic opportunities. The models developed for XCS are still applicable to UCS, since they are based on the occurrence-based reproduction, which is shared in both systems. That is, the key difference between the exploration methodology of both systems is that XCS explores the consequences of all possible actions, while UCS only explores the class of the input instance. Nevertheless, in both cases, niches are only activated when an instance that is matched by the niche schema is sampled. Therefore, in both LCSs, niches that represent instances of the minority class are activated with a lower frequency than niches that represent instances of the majority class. Due to this similarity, we use the models developed for XCS to explain the occurrence-based reproduction in UCS.

To validate that the conclusions extracted from XCS models are still valid for UCS, we ran the same experiments with the parity problem proposed in section 5.7. That is, we ran UCS on the parity problem with $\ell = 10$, $k = \{1, 2, 3, 4\}$, and $ir = \{1, 2, 4, 8, 16, 32, 64, 128\}$. Figure 6.4(a) shows the minimum population with which UCS with the default configuration could

solve the parity problem. In this picture, the empirical values are plotted with points. To analyze the differences introduced by adjusting θ_{GA} according to the theory, the lines depict the population size increase predicted by the theoretical model calculated for the same configurations but with $\theta_{GA} = 0$ (see figure 6.2(a)). As happened with XCS, the empirical results obtained with UCS show that the population size remained nearly constant for all the imbalance ratios. There was only a slightly small increase for $ir > 32$. Figure 6.4(b) provides the same results for UCS with the typical deletion scheme instead of random deletion. The typical deletion scheme protects young classifiers and produces more pressure toward deletion of over-general, inaccurate classifiers. The experimental results show that, with the enhanced deletion scheme, the population size remained constant as the imbalance ratio increased, even for the largest imbalance ratios.

6.6 Takeover Time of Accurate Classifiers in Starved Niches

With the theory and experiments provided so far, we have shown that UCS is able to create representatives of starved niches, and that starved niches receive, at least, a genetic event before removing their representatives. Then, the last element that has to be analyzed according to our design decomposition is whether the best representatives of the different niches will be able to take over their niche when they are in competition with over-general classifiers. Therefore, we model the competition between accurate classifiers and over-general classifiers, especially focusing on the problems caused by rare classes. That is, in imbalance domains, the accuracy of over-general classifiers predicting the majority class may be high since the minority class is under-sampled. This, combined with the occurrence-based reproduction, may promote the existence of over-general classifiers in the population in detriment of accurate representatives of starved niches. In this context, the purpose of this section is two-fold: (i) develop models that predict the takeover time of the best representative of a niche for UCS and (ii) derive the conditions under which the best representative of a starved niche will not be able to take over its niche.

Instead of developing new theory, in this section we consider the same the takeover time models that were derived for XCS in section 5.8. That is, in the takeover time models developed for XCS, we considered a system that evolved a distributed set of niches where each niche contained a representative that was maximally accurate, which we addressed as cl_b ; besides, there was an over-general classifier that matched all niches, which was referred to as cl_o . The quality of these classifiers was denoted by a the accuracy parameter k associated with each classifier, i.e., κ_b and κ_o . Notice that UCS exactly follows the same schema. Therefore, the takeover time models can be directly applied to UCS. The only difference between XCS and UCS is that the accuracy of each classifier is computed differently. For this reason, the conditions under which the best classifier will not be able to take over its niche may vary in both systems. The next section computes these conditions.

6.6.1 Conditions for Starved Niches Extinction under Proportionate Selection

To compute the conditions of niche extinction under proportionate selection, we depart from equation 5.47, that is,

$$P_t < \frac{\rho}{m\rho - 1}, \quad (6.15)$$

to derive under which conditions the best classifier will not take over its niche. m is the number of niches and ρ is the ratio of the accuracy of the over-general classifier to the accuracy of the best representative of the niche, i.e., $\rho = \frac{\kappa_o}{\kappa_b}$. As demonstrated in section 5.47, this inequality only holds for $m > 2$ and

$$1/m < \rho \leq 1. \quad (6.16)$$

In UCS, κ is computed from the raw accuracy acc of the classifier. In imbalanced domains, the most over-general classifier that predicts the majority class will receive ir examples of the majority class for each example of the minority class. Therefore, the raw accuracy of the most over-general classifier is

$$acc_o = \frac{ir}{1 + ir}, \quad (6.17)$$

where $acc_o < acc_0$; that is, the accuracy of the over-general classifier is less than the threshold beyond which UCS considers that a classifier is accurate. The accuracy of the best representative is $acc_b = 1$. From this, we can compute κ_b and κ_o using equation 4.2 of the fitness-sharing scheme as

$$\kappa_o = \alpha \left(\frac{acc_0}{acc_o} \right)^\nu, \quad (6.18)$$

and

$$\kappa_b = 1. \quad (6.19)$$

Replacing equations 6.17, 6.18 and 6.19 into equation 6.16 we obtain that, for proportionate selection, the best classifier will not be able to take over its niche if

$$\frac{1}{m} < \alpha \left(\frac{acc_0 \cdot (ir + 1)}{ir} \right)^\nu < 1 \quad (6.20)$$

where $\frac{ir}{1+ir} < acc_0$; besides, provided that $0 < \alpha < 1$ (usually 0.1) and that $\nu \geq 1$, the right-most inequality is typically satisfied. The left-most inequality can be expressed as

$$\frac{1 + ir}{ir} > \frac{1}{acc_0} \left(\frac{1}{m\alpha} \right)^{\frac{1}{\nu}}. \quad (6.21)$$

Recognizing that the left-most term is the inverse of the raw accuracy of the over-general classifier cl_o , we can derive that the best classifier will not be able to take over its niche if

$$acc_o < acc_0(\alpha m)^{\frac{1}{\nu}}. \quad (6.22)$$

Note that the right-most expression depends on the threshold acc_0 , but especially in α and the number of niches m . As the number of niches depends on the problem, the user can tune the imbalance acceptance of UCS by adjusting the α parameter. That is, lower values of α produce a decrease in the right-most expression. Given the accuracy of the most over-general classifier, which depends directly on ir , we can tune α so that the equation is not satisfied, and thus, the best classifier takes over its niche.

6.6.2 Conditions for Starved Niches Extinction under Tournament Selection

We now perform the same analysis for tournament selection. Tournament selection randomly chooses a set of classifiers from the population and selects the one with highest fitness. As the fitness of the best classifier is greater than the fitness of the over-general classifier, if the best classifier participates in a tournament, it will be selected. For proportionate selection, the condition for the extinction of starved niches depended on ρ ; for this reason, the condition varied from the one computed for XCS. For tournament selection, the condition for the extinction of starved niches only depends on the selection pressure s , the number of niches m , the size of the niche n , the size of the population N , and the number of the representatives in the niche n_b . For this reason, the same condition derived for XCS is still valid for UCS. Thus, the best representative will not be able to take over its niche if

$$1 - m \frac{n_{b,t}}{N} < \left(1 - \frac{n_{b,t}}{n}\right)^s. \quad (6.23)$$

where s is the tournament size, N is the population size, m is the number of niches, n is the number of classifiers in the niche, and $n_{b,t}$ is the numerosity of the best classifier.

In this section, we argued why the takeover time models derived for XCS are still valid for UCS, and have used the takeover time equations to develop the conditions for the extinction of starved niches. In the following section, we put all the pieces together and provide recommendations for UCS configuration in imbalanced domains. Finally, we show that, following the guidelines, UCS, as XCS, is able to solve the 11-bit multiplexer problem with large imbalance ratios.

6.7 Reassembling the Theoretical Framework: UCS in Imbalanced Domains

With the different models and qualitative arguments provided along this chapter, this section follows the same steps as done for XCS to unify all the different models, analyze the interaction among them, and give guidelines on how the system should be configured to guarantee the discovery of the minority class. Then, we show that applying the lessons learned from the theoretical study enables UCS to solve problems with large imbalance ratios.

6.7.1 Patchquilt Integration: from XCS to UCS

In this section, we consider the theoretical framework derived for XCS and study how the new models of UCS can be plugged into the framework. We review the models in the same order

as proposed in section 5.9, that is, from the most restrictive one to the less restrictive one, and compare the differences with respect to the models derived from XCS.

1. The takeover time models set the maximum imbalance ratio beyond which UCS will not be able to discover the minority class. The takeover time models derived for XCS are still valid, and the specific conditions under which representatives of starved niches will be deleted from the population for proportionate and tournament selection have been calculated in equations 6.22 and 6.23 respectively. Satisfying the requirements identified by these models is a necessary but not sufficient condition.
2. The parameter update procedure in UCS is fairly robust, providing accurate approximations of the real values of over-general parameters. Thence, differently from XCS, in UCS it is not necessary to tune the parameter update procedure according to the imbalance ratio.
3. Once the takeover time requirements are met, we have to ensure that accurate representatives of starved niches will be fed into the population. For this purpose, we can either (i) increase the population size linearly with ir —at maximum—or (ii) set θ_{GA} according to ir . Note that the main difference with respect to XCS is that, in UCS, the population size model provides an upper bound instead of predicting the actual increase.

UCS appears to be slightly more robust to class imbalances than XCS since the parameter update procedure is not as sensitive as the XCS’s one and, as experimentally shown, the population size increases slightly slower than XCS’s one. In the next section, we show that, if the recommendations derived from the models are followed, UCS can solve extremely imbalanced data sets.

6.7.2 Solving Highly Imbalanced Domains with UCS

Having revised the information provided by the different methods and established the framework of UCS’s learning from class imbalances, we use this information to tune UCS so that it can solve highly imbalanced domains. For this purpose, we ran UCS on the imbalance 11-bit multiplexer problem with $ir = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$ (see appendix A.4.1 for a description of the problem). We used the default configuration provided in this section with proportionate selection and the following exceptions: (i) crossover was activated with $\chi = 0.8$, (ii) the typical deletion scheme of UCS was employed and (iii) θ_{GA} was set to $n \cdot m \cdot ir$. We set a population size of $N=1,000$, and we used tournament selection. Note that, with this configuration, the requirements of the three items enumerated in the previous section are satisfied:

1. As we used tournament selection, we need to satisfy the condition of equation 6.23. In the previous chapter, we already showed that the proposed configuration satisfied this condition (notice that the condition imposed by tournament selection is equal in both XCS and UCS).
2. Parameters are correctly estimated by the update procedure, especially as the experience of the classifier increases.

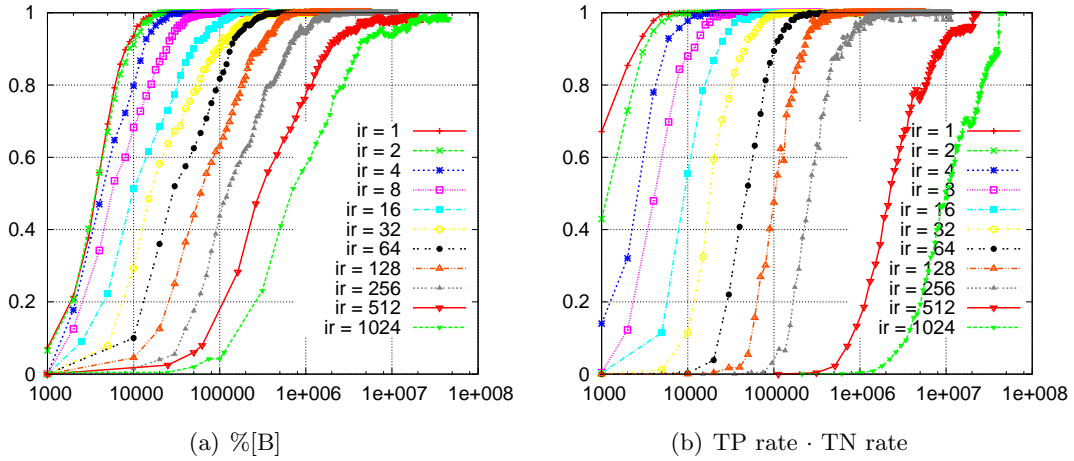


Figure 6.5: Evolution of (a) the proportion of the optimal population and (b) the geometric mean of TP rate and TN rate in the 11-bit multiplexer with $ir = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$.

3. As we set $\theta_{GA} = n \cdot m \cdot ir$, we ensure that starved and nourished niches will have, approximately, the same number of genetic opportunities. Since we take this approach, we maintain the same population size for all the runs.

Figure 6.5 plots (a) the evolution of the proportion of the optimal population size (%[O]) achieved by XCS and (b) the evolution of the geometric mean of TN rate and TP rate in the 11-bit imbalanced multiplexer problems from $ir = 1$ to $ir = 1024$. Note that, for $ir = 1024$, the system only received an instance of the minority class every each 1024 instances of the majority class. The results show that, even under these large imbalance ratios, UCS is able to extract accurate knowledge from the under-sampled class. Figure 6.5(a) shows that UCS is able to obtain 100% of the optimal population for all the runs. Besides, figure 6.5(b) indicates that the system achieves 100% performance measured as the geometric mean of TP rate and TN rate. Notice that, for $ir = 1024$, the performance reaches 100% after activating the condensation runs, which explains that the performance curve increases abruptly from 94% to 100% in the last few iterations. This is because the frequent activation of nourished niches and over-general classifiers of the majority class leads to the creation of some over-general classifiers of the majority class with poorly estimated parameters. As these over-general classifiers match a negative example very infrequently, the system needs several learning iterations to adjust their parameters, realize that they are not accuracy, and remove them from the system. When condensation is activated, as crossover and mutation are deactivated, these classifiers are correctly evaluated and removed from the population. In any case, these results evidence that UCS is able to classify correctly all the input instances, regardless of whether they belong to the minority class or not.

6.8 Summary and Conclusions

In this section, we carried over the facetwise analysis of XCS to UCS. Following the design decomposition provided for LCSs in general, we examined the behavior of UCS in imbalanced domains. Similar conclusions than those extracted for XCS were reached for UCS. That is, the models showed that, to ensure the growth and takeover of representatives of starved niches, either (1) the population size needs to increase linearly with the imbalance ratio or (2) the frequency of application of the GA has to decrease linearly with the imbalance ratio. Besides, two key differences with respect to XCS were found. Firstly, the parameter update procedure of UCS was shown to provide accurate estimates of classifier parameters without requiring any especial configuration. Secondly, theory indicated that the covering operator is able to supply the initial population with schemas of the minority class regardless of the imbalance ratio. Consequently, the population size bounds derived subsequently predict an upper bound, instead of an exact bound, of the scalability of the population size with the imbalance ratio.

Finally, let us point out two important conclusions. The first conclusion is related to the analysis methodology, that is, the design decomposition and facetwise analysis principle. Note that, at the beginning of the previous chapter, we decomposed the complex problem of learning from imbalanced domains in five critical *elements* that need to be satisfied to efficiently deal with rare classes. Then, for each one of the elements, we developed low cost models that explained the corresponding facet, assuming that the others behave in an ideal manner. This approach has two key advantages with respect to creating complex models that try to capture all the interactions in the components of the whole system. The first advantage is that the algebra effort is reduced with respect to that required in global models since each element is analyzed separately, and the interactions with other elements are not considered. At first glance, one may think that this approach also results in models that can explain less than global models which include complex interactions among different elements. Nonetheless, as shown along the two previous chapters, this may not be the case. That is, facetwise models permit focusing on the actual problems of each element, some of which could be hidden in more complex models. Then, the patchquilt integration enables to draw a domain of competence of the systems, indicating the sweet spot in which the system actually scales. The second advantage is that design decomposition enables us to easily transport models from one system to another. In the present chapter, we used parts of the theory developed for XCS, merged this theory together with new models particularly developed for UCS, and put the pieces together, obtaining a framework that explains how UCS behaves in imbalanced domains.

The second conclusion is about the excellence of UCS—and XCS as well—in imbalanced domains. The experiments provided in this section culminated the whole study of the behavior of both LCSs in domains that contain rare, under-sampled classes. In summary, we showed that, as XCS, UCS is a competitive machine learning technique able to deal with large imbalance ratios. We showed this competitiveness in a set of artificial problems which were defined with binary attributes. In the next section, we move to real-world problems which contain continuous values. We will discuss how the theory adapts to these cases and will test both LCSs on a collection of real-world imbalanced classification problems.

Chapter 7

XCS and UCS for Mining Imbalanced Real-World Problems

In the previous two chapters, we have carefully analyzed the behavior of XCS and UCS in domains that contain class imbalances. We decomposed the problem of learning from imbalanced domains in several elements or subproblems and derived facetwise models for each element. This resulted in a better understanding of how the two LCSs work and in the definition of several guidelines or recommendations that need to be satisfied to warrant that the two LCSs are able to learn from rare classes. All these models and recommendations depended on the imbalance ratio ir . Throughout all the theory development, we assumed that the imbalance ratio of the training data set was equivalent to the ratio of the frequency of activation of nourished to the one of starved niches. The artificial problems used to contrast the models met this assumption. Nevertheless, the number of niches and their frequency of activation is not known in real-world problems. Therefore, there is a gap between the theory and its application to effectively solve real-world problems.

The purpose of this chapter is three fold. Firstly, we aim at connecting the dots between theory and application in imbalanced real-world domains. We study in more detail the structure of real-world problems and provide some heuristic procedures, which are based on the information gathered during the online evolution of the two LCSs, to estimate the imbalance ratio between niches; this estimate is used to self-adapt the parameters of the two LCSs according to the recommendations derived from the theory. We show the effectiveness of these procedures in the imbalanced 11-bit multiplexer problem. The second objective is to confirm that both LCSs are really valuable machine learning techniques for supervised learning, and especially, for extracting classification models from imbalanced domains. For this purpose, we compare the performance of XCS and UCS with the one achieved by three of the most influential machine learning techniques (Wu et al., 2007). The third objective is to incorporate re-sampling methods into the comparison, since these types of techniques have been identified—and widely used in the machine learning community—as one of the best alternatives to improve the accuracy of different learning methods in imbalanced domains. For this reason, we include some of the most-used re-sampling techniques into the comparison and empirically analyze how the different learners are influenced by these re-sampling techniques.

The remainder of this chapter is structured as follows. Section 7.1 points out new character-

istics that can be found in real-world problems and how the theory can be adapted to these new characteristics. With the new identified challenges, section 7.2 proposes a heuristic method to self-adapt the configuration parameters of XCS and UCS that are sensible to class imbalances and shows that, with this heuristic procedure, both XCS and UCS can solve the imbalanced 11-bit multiplexer problem although they are not properly configured in the beginning of the run. Section 7.3 compares XCS and UCS with three highly-competent learners, showing the competitiveness of the two LCSs. The study of learning from imbalanced domains is complemented with the introduction of re-sampling techniques. That is, section 7.4 presents some of the most-used re-sampling techniques and illustrates how they work in a case study. These techniques are introduced in the comparison of the five learners in section 7.5. Section 7.6 discusses the overall results and points out some future work lines. Finally, section 7.7 summarizes and concludes this chapter.

7.1 LCSs in Imbalanced Real-World Problems: What Makes the Difference?

In this section, we study how the theory—which has been validated with artificial problems with known characteristics—can be applied to imbalanced real-world problems whose characteristics are unknown and can barely be estimated. As proceeds, we deal with two aspects that need to be solved to adapt the theory to real-world problems. Firstly, as a reminder of the concepts presented in chapter 3, we briefly reintroduce a rule representation for XCS and UCS that is able to deal with data that contain continuous attributes; then, we revise the concept of niche under this new representation. Lastly, we discuss which information is lacking in real-world problems to apply the theory.

7.1.1 XCS and UCS Enhancements to Deal with Continuous Data

Thus far, all the artificial problems used in the previous chapters were defined with binary strings. To solve these problems, we used the original rule representation defined by Wilson (1995), in which each variable of a rule takes a value of the ternary alphabet $\{0,1,\#\}$ (see chapter 3). Nonetheless, real-world problems have new types of attributes such as continuous attributes and ordered nominal attributes. To cope with these new types of data, the system was provided with an interval-based representation in which each variable of a rule is coded with an interval which determines the range of values that the corresponding input attribute can take (Wilson, 2001; Stone and Bull, 2003). Therefore, a rule is a conjunction of feasible intervals, and a new example e matches the a rule if each attribute e_i is included in the interval of the corresponding variable of the rule. For more information about this rule representation the reader is referred to chapter 3.

The introduction of this new representation makes the definition of problem niche and representative of a niche a little fuzzy. According to the definitions given in chapter 5, a niche is a subproblem where a maximally general sub-solution applies. This niche is defined by a schema, and a representative of a niche is any classifier whose condition specifies all the relevant bits of the schema.

These ideas still apply—not rigorously, but intuitively—to real-world problems. In the

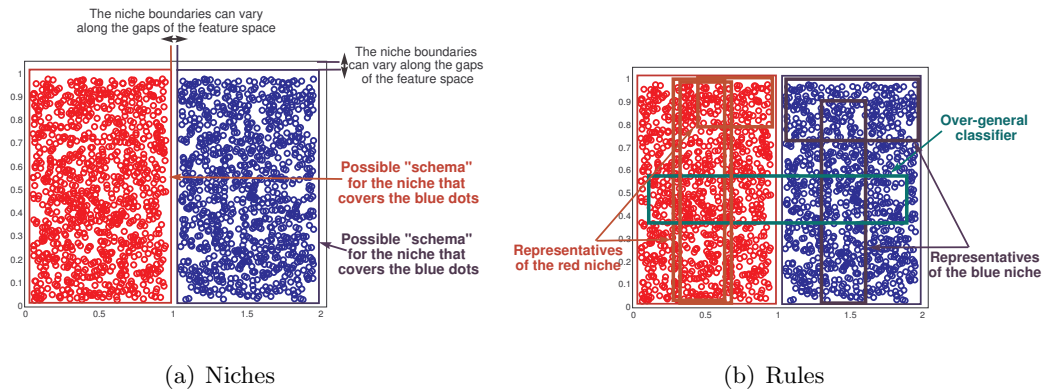


Figure 7.1: Example of a domain with two niches (a) and examples of possible representatives of the two niches and over-general classifiers (b) in a two-dimensional problem with continuous attributes

interval-based representation, a niche can be defined as a hyper rectangle in the feature space in which a maximally general and accurate solution applies. This hyper rectangle expresses the schema of the niche. Consequently, a representative of this niche is any classifier whose condition draws a hyper rectangle included in the hyper rectangle of the niche. Moreover, an over-general classifier is a classifier that defines a hyper rectangle that covers examples of different classes.

To further explain the consequences of this redefinition, figure 7.1(a) shows a very simple two-class domain where there are two niches, and figure 7.1(b) illustrates some examples of representative classifiers of each niche and over-general classifiers. This simple example shows two important aspects that must be highlighted since they make the difference with the definitions of niche and representatives given in the previous chapters. Firstly, notice that there may be several hyper rectangles that represent the niche. That is, by slightly varying one of the sides of any of the hyper rectangles in figure 7.1(a), with the condition that all the instances of the corresponding class are covered and that no instance of another class is matched, we obtain another hyper rectangle that can represent the niche as well. Therefore, the definition of niche schema is not deterministic. Secondly, there may be different accurate representatives of the niches whose condition is partially overlapped. This aspect is not exclusive of continuous-valued problems; in binary problems, we could find some overlapped representatives. Nonetheless, in continuous-valued problems, as the interval-based representation can define any possible hyper rectangle, the number of potential overlapping representatives increases abruptly. In general, it can exist an infinite number of representatives, equally general¹, that are highly overlapped. For example, in figure 7.1(b) there are two representatives of the red niche that are equally general and highly overlapped.

Although these differences, the ideas derived from the facetwise analysis are still valid in this new scenario. That is, the same mechanisms of the evolutionary learners apply: the population is initialized by the covering operator, and the evolutionary pressures drive the search toward obtaining representatives of different niches. These best representatives should be able

¹In the interval-based representation, the generality can be computed as the volume of the hyper rectangle defined by the condition.

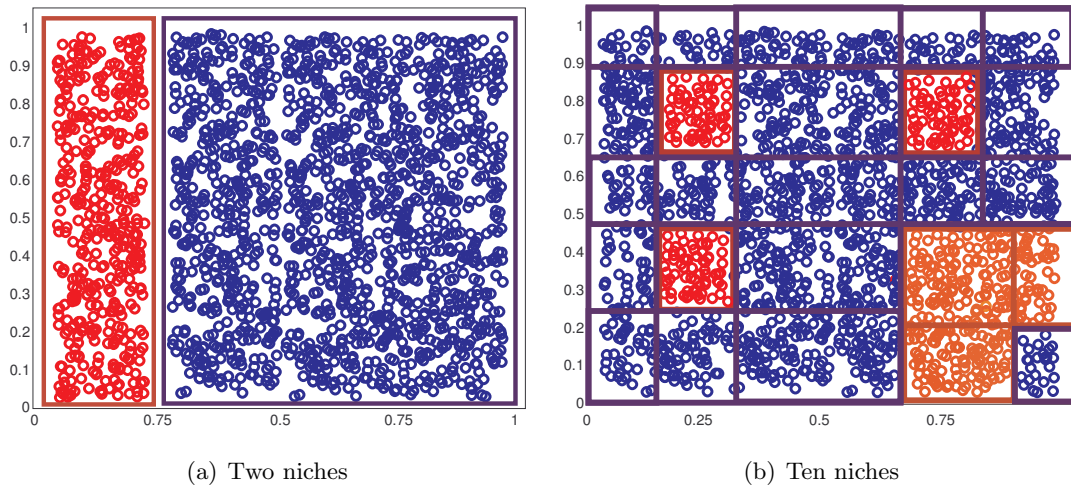


Figure 7.2: Example of two domains with the same imbalance ratio in the training data set but different niche imbalance ratio.

to take over their niches and remove competing over-general, less accurate classifiers. The main difference with the binary case is that, due to the fuzziness inherent in the definition of niche schema, there may be several highly overlapped representatives of the niches which will share the niche resources instead of having a single representative per niche. Therefore, an effort needs to be made to connect this new situation to the theory. The implications of this difference are analyzed in more detail in the next subsection.

7.1.2 What Do we Need to Apply the Theory?

Having redefined the concepts of niche and representative in imbalanced domains, now we are in position to examine which information, handy in the artificial problems used in the two previous chapters, is not available in real-world data sets. That is, the different models developed in the previous chapters were translated into a set of recommendations that suggested to configure different parameters of XCS and UCS depending on the imbalance ratio ir . We assumed that the class-imbalance ratio of the training data set reflected the ratio of the frequency of activation of nourished niches to the frequency of activation of starved niches, which we refer to as the *niche imbalance ratio* in the rest of this chapter. This condition was satisfied in the tested artificial problems, since the number of starved niches was equal to the number of nourished niches, all nourished niches had the same activation frequency, and all starved niches had the same activation frequency which, in turn, was smaller than that of nourished niches.

Nonetheless, this is not the case in real-world domains since, the formation of niches depends not only on the class-imbalance ratio but also on the distribution of the training examples in the feature space. To illustrate this, figure 7.2 shows two domains with the same imbalance ratio. Notice that, in these examples, the niche imbalance ratio is not directly determined by the class-imbalance ratio of the training data set. That is, the domain in figure 7.2(a) contains one niche of the minority class, and the domain in figure 7.2(b) consists of five niches of the

minority class—two of them overlapped—with a lower number of instances per niche. Therefore, the domain in figure 7.2(b) is more difficult to learn than the domain in figure 7.2(a), since both LCSs have to discover a larger number of smaller niches. Besides, notice that, in figure 7.2(b), there exist niches with different frequencies of activation², and that, in this case, the most starved niche corresponds to a niche of the majority class—that is, the bottom right most niche.

Two important conclusions can be extracted from this elemental example. Firstly, that the imbalance ratio of the training data set may provide a misleading information about the real niche imbalance; therefore, we need to develop new procedures to obtain more accurate estimates of the niche imbalance ratio. Secondly, that the *niche imbalance problem* may be present in the majority of real-world domains, and that this is related to (1) the *knowledge representation* and (2) the *geometrical distribution* of the training examples in the feature space.

To further illustrate this last point, let us suppose that XCS or UCS are used to extract an interval-based rule set from the domain represented in figure 7.3, which is a completely balanced data set with oblique boundaries. The same figure shows some of the possible niches and representatives of the blue class. The two LCSs may be expected to have no problems to learn this domain since the training data set is completely balanced. Nevertheless, note that the interval-based representation needs to evolve some representatives whose conditions define small hyper rectangles to approximate the class boundary accurately; these representatives belong to *starved niches*. On the other hand, the interval-based representation also enables the existence of representatives with larger conditions—which belong to *nourished niches*—that match examples that are far away from the class boundary. Notice that, in this particular example, this problem is due to the combination of geometrical complexity and expressiveness—or shape—of the rule representation, but not to the class-imbalance ratio. If we had conditions that defined triangles in the solution space, this domain could be predicted with only two rules.

In fact, a similar problem has been addressed by the machine learning community, in the context of offline learning, under the label of the problem with *small disjuncts* (Holte et al., 1989). That is, a disjunct is the analogous definition of niche in an offline system, and the problem of small disjuncts refers to the problem of extracting accurate models of infrequent or starved niches. As discussed in chapter 5, approaches designed to deal with this problem in offline learning can be barely carried over to online learning since, in the latter one, instances are made available in data streams, and so, no information about the class distribution is known a priori. For sake of notation, in the remainder of this chapter, we will indistinctively use the two terms to refer to the described problem.

In summary, the problem of learning from imbalanced domains has been broadened due to the presence of continuous attributes; note that, now, the effects of class imbalances can be present in any real-world problem. Thence, we reformulate the problem as follows. As in the binary case, we are concerned about the competition among starved niches, nourished niches, and over-general classifiers. Notwithstanding, the imbalance ratio gives now little information about the distribution of niches around the feature space. In this context, the general purpose would be to estimate the number of niches of the system and the frequency of all these niches to get an accurate estimate of the niche imbalance ratio and tune the configuration of the two LCSs based on this estimate. In fact, if we could be able to perform this complex task, we would

²The frequency of activation is directly related to the number of instances that are included in the hyper rectangle defined by the niche.

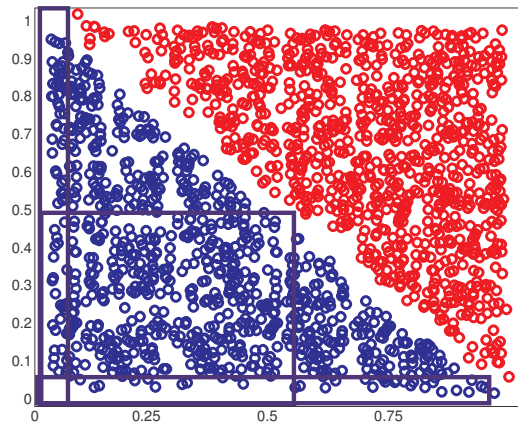


Figure 7.3: Example of a domain with oblique boundaries. Several interval-based rules are required to define the class boundary precisely.

have solved the learning problem itself, removing the necessity of applying a machine learning technique. Here, we relax the goal of obtaining the information of all niches and define that the niche imbalance ratio equals to the ratio between the frequency of the most nourished niche and the frequency of the most starved niche that lay together in the solution space. Thus, we only need to estimate the frequency of two niches of our problem to obtain an estimate that represents the upper bound of the niche imbalance ratio. Even though this simplification is done, the problem of estimating the niche frequency is not trivial. In the following section, we propose a mechanism to estimate the niche imbalance ratio.

7.2 Self-Adaptation to Particular Unknown Domains

This section presents a heuristic approach to determine the frequency of application of starved and nourished niches and self-configure XCS and UCS based on this information. This approach enables us to properly estimate the niche imbalance ratio and so to self-configure both LCSs according to this information. Before applying XCS and UCS to real-world problems, we show that this self-configuration procedure enables both XCS and UCS to solve the imbalanced 11-bit multiplexer problem with large imbalance ratios without being previously configured according to the imbalance ratio.

7.2.1 Online Adaptation Algorithms

To estimate the niche imbalance ratio ir_n and self-adapt the LCSs based on this estimate, we propose to use the information that intrinsically resides in over-general classifiers. Over-general classifiers cover several niches that are close in the feature space. By computing the number of examples covered per class of an over-general classifier, we can estimate the imbalance ratio between these niches. Note that this strategy permits not only detecting the presence of starved niches, but also calculating an estimate of the imbalance ratio between these starved niches and

Algorithm 7.2.1: Pseudo code for the *online adaptation algorithm* in XCS.

```

1 Algorithm: OnlineAdaptationXCS ( cl is classifier )
   Data: cl is a classifier after updating its parameters.
   Result: Modify  $\beta$  and  $\theta_{GA}$  if necessary.
2 if cl is overgeneral then
3    $ir_n := \frac{exp_{maj}(cl)}{exp_{min}(cl)}$ 
4   if ( $ir_n < \frac{2R_{max}}{\epsilon_0} \wedge exp_{cl} > \theta_{ir} \wedge num_{cl} > \overline{num}_{[P]}$ ) then
5     | Adapt  $\beta$  and Adapt  $\theta_{GA}$  based on  $ir_n$ 
6   end
7 end

```

Algorithm 7.2.2: Pseudo code for the on-line adaptation of β .

```

1 Algorithm: Adapt  $\beta$  (  $ir_n$  is double )
   Data:  $ir_n$  is the niche imbalance ratio
    $\zeta$  is a discount factor ( $0 < \zeta < 1$ )
   Result: New value of  $\beta$ .
2  $\epsilon_{th} = 2 \cdot R_{max} \frac{ir}{(1+ir)^2}$ 
3 Obtain  $\epsilon_{ir_\beta}$  with the current value of  $\beta$ 
4 while  $\epsilon_{ir_\beta} < p_{th}$  do
5   |  $\beta = \beta \cdot \zeta$ 
6   | Obtain  $\epsilon_{ir_\beta}$  with the current value of  $\beta$ 
7 end

```

their neighbors.

We first present the algorithm for online self-adaptation of XCS and later translate this algorithm to the particular case of UCS. Algorithm 7.2.1 provides the pseudo code of the main procedure, and algorithm 7.2.2 supplies the code for the subroutine that specifically tunes the β parameter for the Widrow-Hoff rule. The algorithm works as follows. Algorithm 7.2.1 is applied to each classifier after updating its parameters. It first checks whether the classifier is over-general or not. For this purpose, we extended the parameters of a classifier to compute the experience per class. Then, a classifier is over-general if it is experienced in more than one class. Next, the imbalance ratio between the niches in which this classifier participates is estimated as follows. We select the class with maximum experience $exp_{maj}(cl)$ and the class with minimum experience $exp_{min}(cl)$ (we require that $exp_{min}(cl) > 0$), and return the ratio of these two values as an estimate of the niche imbalance ratio. Then, we use this information to self-adapt the configuration of XCS only if (i) the niche imbalance ratio is less than the maximum imbalance ratio identified in equation 5.9, (ii) the classifier is experienced enough ($exp_{cl} > \theta_{ir}$, where θ_{ir} is a configuration parameter), and (iii) the classifier is strong in the population, i.e., if its numerosity is greater than the average numerosity of the classifiers in the population.

If we are using the Widrow-Hoff rule, we first adapt β . Algorithm 7.2.2 provides the implementation details of this procedure. The goal of the algorithm is to adjust the value of β

Algorithm 7.2.3: Pseudo code for the *online adaptation algorithm* in UCS.

```

1 Algorithm: OnlineAdaptationUCS ( cl is classifier )
   Data: cl is a classifier after updating its parameters.
   Result: Modify  $\theta_{GA}$  if necessary.
2 if cl is overgeneral then
3    $ir_n := \frac{exp_{maj}(cl)}{exp_{min}(cl)}$ 
4   if (  $ir_n < acc_0 \wedge exp > \theta_{ir} \wedge num_{cl} > \overline{num}_{[P]}$  ) then
5     | Adapt  $\beta$  and Adapt  $\theta_{GA}$  based on  $ir_n$ 
6   end
7 end

```

so that the estimate error of a classifier approaches its theoretical value, which is computed in equation 5.5. Thus, the procedure first computes this theoretical value, ϵ_{th} . Then, it calculates the real value of the error for the given β . To do this, the algorithm assumes the worst case: that an instance of the minority class is sampled, and then, ir_{cl} instances of the majority class are received. If the real value of the error is lower than the theoretical one, β is decreased according to a discount factor ζ . This process is repeated until a β for which the theoretical and the real value of the error are approximately the same is found. Finally, the algorithm adapts θ_{GA} by setting $\theta_{GA} = ir$.

In algorithm 7.2.3, this procedure is extended to UCS. The algorithm works similarly to the one designed for XCS with two main differences. The first difference is that, in UCS, the update parameter procedure does not need to be adapted. The second difference is that the condition to update θ_{GA} depends on whether $ir_n > acc_0$. The remaining part of the algorithm is the same. In the next section, we show that these self-adaptation mechanisms enable XCS and UCS to self-configure according to the estimated niche imbalance ratio and solve the imbalanced 11-bit multiplexer with large imbalance ratios.

7.2.2 Experiments

In this section, we empirically analyze whether the two heuristic procedures can provide accurate estimates for XCS and UCS. For this purpose, we ran XCS and UCS on the imbalanced 11-bit multiplexer problem with $ir = \{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$, that is, the same experiments with the imbalanced multiplexer problem performed in chapters 5 and 6. We used the same configuration proposed in these two chapters for XCS and UCS, but with the following exceptions. We set $\theta_{GA} = 0$ for both systems and fixed $\beta = 0.2$ for XCS. That is, we did not configure the systems according to the imbalance ratio and the recommendations derived from the theory. Therefore, we expected the heuristic procedures to discover the niche imbalance ratio of each problem and to use it to self-adapt both systems. All the results provided as follows are averages over 25 runs with different random seeds.

Figure 7.4 plots the results obtained by XCS and UCS in the imbalanced 11-bit multiplexer problem with imbalance ratios ranging from $ir = 1$ to $ir = 1024$. More specifically, figures 7.4(a) and 7.4(c) plot the proportion of the optimal population achieved by XCS and UCS, and figures 7.4(b) and 7.4(d) depict the evolution of the performance, measured as the product of TN rate

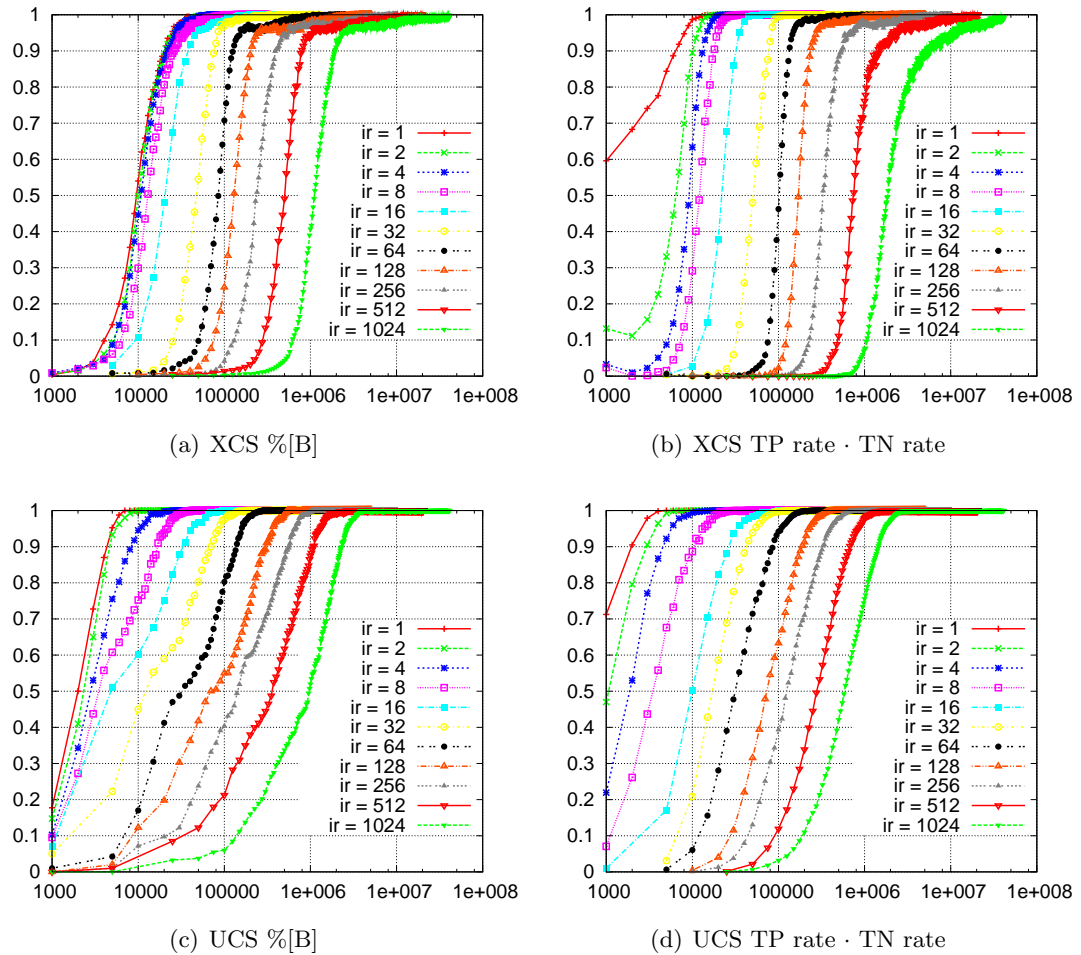


Figure 7.4: Evolution of (a,c) the proportion of the optimal population and (b,d) the geometric mean of TP rate and TN rate of XCS and UCS, respectively, in the 11-bit multiplexer with $ir=\{1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024\}$.

and TP rate, of XCS and UCS respectively. These results can be compared with those obtained by XCS and UCS when they were properly configured according to the imbalance ratio and the recommendations derived from the theory (see figures 5.11 and 6.5 respectively).

The results show that both XCS and UCS could achieve 100% optimal population and 100% performance for all the imbalance ratios. Therefore, this confirmed that the heuristic procedure was able to tune the configuration of both LCSs correctly. Furthermore, these results also permitted establishing a comparison of the performance of XCS and UCS. UCS achieved 100% of the optimal population slightly quicker than XCS, especially in the larger imbalance ratios. Note that, although XCS's curves were steeper at the beginning of the run, the system suffered more than UCS to discover the last optimal classifiers in the population. These behavior was also observed in the performance curves for high imbalance ratios. Nonetheless, it is worth noticing that both approaches could completely learn all the optimal population and classify all

the input instances correctly with similar training time. This indicates that the application of these systems in real-world imbalanced domains with unknown characteristics holds promise. In the next section, we deal with real-world problems and analyze the capabilities of both online learning architectures.

7.3 LCSs in Imbalanced Real-World Domains

After analyzing the two LCSs on artificial problems, we now apply both systems to a collection of real-world imbalanced problems and examine their behavior. The understanding of LCSs behavior—and the behavior of any learner in general—on real-world problems is really complicated since these problems may have different sources of complexity which can hardly be identified; the interaction of all these complexities may limit the maximum performance that a given learner can achieve (Ho and Basu, 2002). Notice the difference between real-world problems and the artificial problems that have been used through all the previous study, where we could control the different sources of complexity.

Thence, we need to take another approach to evaluate the competence of XCS and UCS in real-world problems. That is, information about the optimal population is no longer available, and providing the training or test accuracy of the two learners may be not enough to conclude whether the two systems are competitive for mining real-world domains. To measure the competence of both LCSs on imbalanced real-world domains, we compare the performance of XCS and UCS with three of the most influential machine learning systems (Wu et al., 2007). Therefore, the aim of this section is to analyze whether XCS and UCS are competitive with these highly recognized learning methods. It is worth noticing that XCS and UCS perform online learning—i.e., they process data streams—, whereas the three order methods learn offline. Thus, XCS and UCS provide an added value with respect the three other techniques. As proceeds, we first present the methodology, and then, we compare XCS and UCS with the other learners.

7.3.1 Comparison Methodology

Before proceeding with the analysis of the experimental results, we first describe the test problems used in the comparison, and the details about the metrics used to evaluate the learners and the statistical tests employed to aid the process of conclusion extraction.

We used a collection of 25 real-world problems with different characteristics and imbalance ratios, which were constructed as follows. We selected the following twelve problems: *balance-scale*, *bupa*, *glass*, *heart disease*, *pima indian diabetes*, *tao*, *thyroid disease*, *waveform*, *Wisconsin breast-cancer database*, *Wisconsin diagnostic breast cancer*, *wine recognition data*, and *Wisconsin prognostic breast cancer*. All the real-world problems were obtained from the UCI repository (Asuncion and Newman, 2007), except for *tao*, which was selected from a local repository (Bernadó-Mansilla et al., 2002). To force higher imbalance ratios and increase the test bed, we discriminated each class against all the other classes in each data set, considering each discrimination as a new problem. Thus, n two-class problems were created from a problem with n classes ($n > 2$), resulting in a test bed that consisted of 25 two-class real-world problems. Table 7.1 gathers the most relevant features of the problems. Note that the imbalance ratio between niches ir_n can be much higher than the imbalance ratio of the learning data set reported in the

Table 7.1: Description of the data sets properties. The columns describe the data set identifier (Id.), the original name of the data set (Data set), the number of problem instances (#Ins.), the number of attributes (#At.), the proportion of minority class instances (%Min.), the proportion of majority class instances (%Maj.), and the imbalance ratio (ir).

Id.	Data set	#Ins.	#At.	%Min.	%Maj.	ir
<i>bald1</i>	balance-scale disc. 1	625	4	7.84%	92.16%	11.76
<i>bald2</i>	balance-scale disc. 2	625	4	46.08%	53.92%	1.17
<i>bald3</i>	balance-scale disc. 3	625	4	46.08%	53.92%	1.17
<i>bpa</i>	bupa	345	6	42.03%	57.97%	1.38
<i>glsd1</i>	glass disc. 1	214	9	4.21%	95.79%	22.75
<i>glsd2</i>	glass disc. 2	214	9	6.07%	93.93%	15.47
<i>glsd3</i>	glass disc. 3	214	9	7.94%	92.06%	11.59
<i>glsd4</i>	glass disc. 4	214	9	13.55%	86.45%	6.38
<i>glsd5</i>	glass disc. 5	214	9	32.71%	67.29%	2.06
<i>glsd6</i>	glass disc. 6	214	9	35.51%	64.49%	1.82
<i>h-s</i>	heart-disease	270	13	44.44%	55.56%	1.25
<i>pim</i>	pima-inidan	768	8	34.90%	65.10%	1.87
<i>tao</i>	tao-grid	1888	2	50.00%	50.00%	1.00
<i>thyd1</i>	thyroid disc. 1	215	5	13.95%	86.05%	6.17
<i>thyd2</i>	thyroid disc. 2	215	5	16.28%	83.72%	5.14
<i>thyd3</i>	thyroid disc. 3	215	5	30.23%	69.77%	2.31
<i>wavd1</i>	waveform disc. 1	5000	40	33.06%	66.94%	2.02
<i>wavd2</i>	waveform disc. 2	5000	40	33.84%	66.16%	1.96
<i>wavd3</i>	waveform disc. 3	5000	40	33.10%	66.90%	2.02
<i>wbcd</i>	Wis. breast cancer	699	9	34.48%	65.52%	1.90
<i>wdbc</i>	Wis. diag. breast cancer	569	30	37.26%	62.74%	1.68
<i>wined1</i>	wine disc. 1	178	13	26.97%	73.03%	2.71
<i>wined2</i>	wine disc. 2	178	13	33.15%	66.85%	2.02
<i>wined3</i>	wine disc. 3	178	13	39.89%	60.11%	1.51
<i>wdbc</i>	wine disc. 4	198	33	23.74%	76.26%	3.21

table.

The performance was measured with the product of TP rate and TN rate. Ten-fold cross validation (Dietterich, 1998) was used to estimate the product of TP rate and TN rate. The results obtained with the different techniques were statistically compared with the following procedure. We first used the multiple-comparison Friedman’s test (Friedman, 1937, 1940) to test the null hypothesis that all the learning methods performed the same on average. If the null hypothesis was rejected, the Nemenyi test (Nemenyi, 1963) was employed to identify groups of learners with statistically equivalent results. Moreover, as we were interested in analyzing the differences in particular problems, the performance of each pair of learning algorithms on each problem was compared using the Wilcoxon signed-ranks test (Wilcoxon, 1945). We acknowledge in advance that pairwise comparisons increment the risk of rejecting null hypotheses that are actually true.

In our experiments, we assume this risk with the aim of providing further information about the excellence of each learning algorithm in particular problems. For more information about the used tests, the reader is referred to appendix B.

Both LCSs were compared with three of the most competent learners: C4.5 (Quinlan, 1995), SMO (Platt, 1998), and IBk (Aha et al., 1991). C4.5 is a decision tree derived from the ID3 algorithm (Quinlan, 1979). SMO is a support vector machine (Vapnik, 1995) that implements the *Sequential Minimal Optimization* algorithm. IBk is a nearest neighbor algorithm. All these machine learning methods were run using WEKA (Witten and Frank, 2005), and the recommended default configuration was used. We selected the model for SMO as follows. We ran SMO with polynomial kernels of order 1, 5, and 10, and with Gaussian kernels. Then, we ranked the results obtained with the four configurations and chose the model that maximized the average rank: SMO with lineal kernels. In this way, we avoided using particular configurations for each problem. We followed the same process with IBk, which was ran for $k = \{1, 3, 5, 7\}$; here, we provide the results with $k=5$. XCS and UCS were configured as previously specified, except for $N=6400$, and the two parameters that refer to the interval-based representation, i.e., $r_0=0.6$, and $m_0=0.1$. Finally, we did not introduce asymmetric cost functions in any system, although the majority of them permitted it. In this way, we aimed at analyzing the intrinsic capabilities of each method to deal with class imbalances.

7.3.2 Results

After defining the experimental methodology, we now analyze the results obtained with the different learning methods. Table 7.2 summarizes the performance of the different learners on the 25 data sets. The last three rows provide the average accuracy, the average rank, and the position of each learner in the ranking. The ranks were calculated as follows. For each data set, we ranked the learning algorithms according to their performance; the learner with the highest accuracy held the first position, whilst the learner with the lowest accuracy held the last position of the ranking. If a group of learners had the same performance, we assigned the average rank of the group to each one of the learners in the group.

The results provided in this table allowed for two types of analyses. Firstly, the results indicated which problems were more complex, in general, for all the learning systems. All learners presented poor performance in the problems *baldd1*, *bpa*, *glsd1*, *glsd3*, *pim*, and *wpsc*. Examining the measure of performance, we observed that all the learners had a low TP rate, which indicated that the minority class was not well defined in these problems. Most of these data sets were highly imbalanced; so, the imbalance ratio turned up to be an important factor that hindered the performance of the tested learners. Nonetheless, the problems *bpa* and *pim* were almost balanced, so there might be other complexity factors affecting the learning performance such as small disjuncts.

Moreover, the experimental results also allowed for a statistical comparison of the performance of the different learners. Firstly, let us note that XCS and UCS were the two best ranked techniques. That means that the two LCSs were among the best performers in most of the problems. To analyze whether this improvement was statistically significant, we used multiple-comparison tests to check the null hypothesis that all the learners performed the same on average. The Friedman multiple-comparison test did not permit rejecting the null hypothesis with $p = 0.2519$. Consequently, post-hoc tests could not be applied since no significant

Table 7.2: Comparison of C4.5, SMO, IBk, XCS, and UCS on the 25 real-world problems. Each cells depicts the average value of the product of TP rate and TN rate and the standard deviation. *Avg* gives the performance average of each method over the 25 data sets. The two last rows show the average rank of each learning algorithm (*Rank*) and its position in the ranking (*Pos*).

	C4.5	SMO	IB5	XCS	UCS
<i>bald1</i>	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
<i>bald2</i>	69.30 ± 6.83	84.03 ± 7.30	81.16 ± 5.54	71.14 ± 5.02	69.75 ± 8.19
<i>bald3</i>	71.20 ± 6.04	85.81 ± 8.40	82.11 ± 8.67	69.98 ± 7.23	73.61 ± 6.66
<i>bpa</i>	33.08 ± 14.09	0.00 ± 0.00	32.40 ± 9.44	47.58 ± 10.92	47.59 ± 11.22
<i>glsd1</i>	79.50 ± 42.16	0.00 ± 0.00	69.32 ± 48.30	20.00 ± 42.16	59.00 ± 50.87
<i>glsd2</i>	34.50 ± 47.43	15.00 ± 33.75	24.13 ± 35.36	59.00 ± 45.02	74.00 ± 41.89
<i>glsd3</i>	28.97 ± 42.16	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.49 ± 25.17
<i>glsd4</i>	73.55 ± 32.63	80.03 ± 24.33	77.07 ± 24.98	80.03 ± 24.33	83.54 ± 19.53
<i>glsd5</i>	66.52 ± 16.77	9.50 ± 9.42	62.26 ± 21.14	68.67 ± 18.71	65.63 ± 21.46
<i>glsd6</i>	52.54 ± 15.13	0.00 ± 0.00	61.74 ± 18.23	60.53 ± 11.21	57.06 ± 14.20
<i>h-s</i>	63.33 ± 13.29	68.83 ± 8.87	64.40 ± 14.65	59.89 ± 15.59	55.00 ± 13.61
<i>pim</i>	43.87 ± 13.27	48.31 ± 5.60	46.91 ± 4.84	45.85 ± 6.37	47.82 ± 6.60
<i>tao</i>	90.98 ± 2.14	70.59 ± 6.45	94.25 ± 2.10	82.89 ± 5.42	78.81 ± 7.18
<i>thyd1</i>	87.61 ± 16.10	76.67 ± 22.50	76.67 ± 22.50	78.36 ± 22.01	92.25 ± 13.66
<i>thyd2</i>	93.24 ± 12.45	54.17 ± 24.92	77.90 ± 21.40	82.50 ± 24.98	93.06 ± 12.09
<i>thyd3</i>	87.65 ± 10.34	33.81 ± 21.35	81.12 ± 16.16	89.84 ± 11.75	88.08 ± 14.89
<i>wavd1</i>	67.79 ± 4.06	78.68 ± 4.27	72.28 ± 3.97	80.44 ± 2.97	76.33 ± 2.10
<i>wavd2</i>	62.54 ± 3.89	72.30 ± 2.71	67.49 ± 1.75	73.48 ± 2.88	71.49 ± 3.83
<i>wavd3</i>	68.60 ± 2.38	79.57 ± 2.04	74.14 ± 2.86	81.01 ± 3.99	76.60 ± 4.14
<i>wbcd</i>	89.12 ± 3.42	92.70 ± 5.32	92.72 ± 5.36	92.31 ± 5.50	94.06 ± 4.23
<i>wdbc</i>	88.79 ± 5.09	94.28 ± 3.28	93.47 ± 3.64	90.27 ± 4.61	89.68 ± 5.61
<i>wined1</i>	85.15 ± 16.63	98.46 ± 3.24	94.98 ± 8.29	99.23 ± 2.43	99.23 ± 2.43
<i>wined2</i>	91.81 ± 8.05	97.50 ± 5.62	97.50 ± 4.03	99.17 ± 2.64	91.88 ± 10.02
<i>wined3</i>	87.62 ± 11.70	97.14 ± 6.02	87.94 ± 12.53	93.38 ± 7.15	85.33 ± 9.55
<i>wpbc</i>	33.55 ± 12.87	9.37 ± 16.98	28.98 ± 16.49	20.33 ± 16.38	17.17 ± 21.63
Avg	66.03	53.87	65.64	65.83	68.26
Rank	3.46	3.14	3.08	2.52	2.80
Pos	5	4	3	1	2

differences among the learners were found (Demšar, 2006). This conclusion is not surprising since compared XCS and UCS with three of the most competent machine learning techniques. Nonetheless, note that these results highlight the robustness of XCS and UCS. That is, XCS and UCS were not only as competitive as three of the most competent machine learning techniques in the used test bed, but they also were the best ranked methods of the comparison.

To extend the statistical analysis to each particular problem, we applied statistical pairwise comparisons according to a Wilcoxon signed-ranks test at 0.95 confidence level. Table 7.3 shows the results. The • and ◦ symbols denote a significant degradation/improvement of the given learning algorithm with respect to another in a particular data set. The overall degradation/improvement comparison (see the row labeled *Score*) permitted ranking the quality of the five learners. Under this criterion, XCS appeared as the most robust method with a ratio of

Table 7.3: Comparison of C4.5, SMO, IBk, XCS, and UCS on the 25 real-world problems. For a given problem, the ● and ○ symbols indicate that the learning algorithm of the column performed significantly worse/better than another algorithm at 0.95 confidence level (pairwise Wilcoxon signed-ranks test). *Score* counts the number of times that a method performed worse-better, and *Score_{ir>5}* does the same but only for the highest imbalanced problems (*ir* > 5).

	C4.5	SMO	IBk	XCS	UCS
<i>bald1</i>					
<i>bald2</i>	●●	○○○	○○○	●●	●●
<i>bald3</i>	●●	○○○	○○○	●●	●●
<i>bpa</i>	●●○	●●●●	●●○	○○○	○○○
<i>glsd1</i>	○○	●●●	○	●	○
<i>glsd2</i>		●●	●	○	○○
<i>glsd3</i>					
<i>glsd4</i>					
<i>glsd5</i>	○	●●●●	○	○	○
<i>glsd6</i>	○	●●●●	○	○	○
<i>h-s</i>		○	○		●●
<i>pim</i>					
<i>tao</i>	●○○○	●●●●	○○○○	●●○○	●●●○
<i>thyd1</i>					
<i>thyd2</i>	○	●●●●	○	○	○
<i>thyd3</i>	○	●●●●	○	○	○
<i>wavd1</i>	●●●●	○○	●●●○	○○○	●○○
<i>wavd2</i>	●●●●	○○	●●●○	○○	○○
<i>wavd3</i>	●●●●	○○	●●○	○○○	●○
<i>wbcd</i>	●●●	○	○		○
<i>wdbc</i>	●	○○	○	●	●
<i>wined1</i>	●●●	○		○	○
<i>wined2</i>					
<i>wined3</i>		○		○	●●
<i>wpbc</i>					
Score	26-10	29-18	11-22	8-20	14-18
Score_{ir>5}	0-3	9-0	1-2	1-2	0-4

degradation/improvement of 8/20, followed closely by IBk and UCS. Both LCSs presented the poorest results with respect to the other learners in the *bald2*, *bald3*, and *tao* problems, which have a low imbalance ratio. In (Bernadó-Mansilla and Ho, 2005), the hyper rectangle codification used by XCS and UCS was shown to be inappropriate when the boundary between classes in the learning data set was curved. This is the case of the *tao* problem (Bernadó-Mansilla et al., 2002). We hypothesize that *bald2* and *bald3* are also characterized by curved boundaries, which would explain the degradation in performance of both LCSs. This hypothesis is also supported by the results obtained with IBk, which improved XCS and UCS in the three aforementioned problems. IBk is not affected by curved boundaries since it decides the output as the majority class of the *k* nearest neighbors.

The two last methods in the ranking were C4.5 and SMO. The surprisingly poor rank of C4.5 was mainly caused by the results obtained in the problems *wavd1*, *wavd2*, and *wavd3*,

in which C4.5 was outperformed by all the other learners. These results were not correlated with the imbalance ratio, so there may be other types of complexity that made C4.5 perform poorly in these problems. Finally, SMO was the last ranked method. It showed a tendency to over-generalize toward the majority class in problems with moderate and high class imbalances such as *glsd1*, *glsd3*, and *glsd6*, in which the TP rate was zero. The same behavior was shown in problems with low imbalance ratios such as the *bpa* problem, which we identified as a difficult problem may be due to the tendency of the learners to create small disjuncts to create accurate models of these problems. However, we can also find significant improvements with respect to other learners in the problems: *baldd2*, *baldd3* and *wdbc*. Thus, these results indicate that SMO performs competitively in a restricted set of problems, but it is affected by some complexities among which we may find the imbalance ratio.

Finally, let us compare the learners in terms of imbalance robustness. To do this, we consider the data sets with the highest imbalance ratio: *glsd1*, *glsd2*, *baldd1*, *glsd3*, *glsd4*, *thyd1*, and *thyd2*, which have imbalance ratios ranging from $ir=5$ to $ir=23$. In these problems, UCS appeared to be the best learner, with a degradation/improvement ratio of 0/4, followed closely by C4.5. These results agree with several papers which indicate that C4.5 can deal with high amounts of class imbalance (Japkowicz and Stephen, 2002; Batista et al., 2004). IBk and XCS were the two next methods in the ranking. IBk might suffer from *small disjuncts*, since minority class regions are surrounded by many instances of the majority class, concentrating a high amount of the test error around the small disjuncts. XCS also turned up to be more sensitive to class imbalances than UCS and C4.5. Lastly, SMO performed poorly in the most imbalanced data sets. As mentioned above, we tried other orders of polynomial kernels, as well as a Gaussian kernel, but no significant improvement was found.

In this section, we have shown the competitiveness of both XCS and UCS with respect to three of the most influential machine learning techniques in imbalanced data sets. Throughout all the comparison, we have considered the intrinsic capabilities of the learners to deal with rare classes without introducing further mechanisms to promote the discovery of the knowledge that resides in the minority class. In the following sections, we consider these types of mechanisms. As the comparison contains learning algorithms with different characteristics, we use re-sampling techniques since they are independent of the final learner. As proceeds, we first explain the re-sampling methods considered in the analysis, and further study whether they improve the accuracy of the models of the minority class when they are combined with the five learning methods used in this section.

7.4 Re-sampling Techniques

Re-sampling techniques have become one of the most used approaches to boost the capabilities of machine learning techniques to discover the knowledge that resides in rare classes. These types of methods are pre-processing methods that re-balance the proportion of examples of the minority class in the training data set by either over-sampling the minority class or under-sampling the majority class. During the last few years, several approaches have been developed in this field. Herein, we adopt three of the most famous algorithms: random over-sampling (Ling and Li, 1998), under-sampling based on Tomek links (Batista et al., 2004), and synthetic minority over-sampling technique (SMOTE) (Chawla et al., 2002). We selected these three

Algorithm 7.4.1: Pseudo code for the Tomek Links algorithm.

```

1 Algorithm: TomekLinks ( d is Dataset )
   Data: d is the training data set
   Result: Collection of Tomek links represented as pairs of examples
2 var
3   | setTomek is PairsExamples
4   | exMin, exMaj, ex is TrainingExample
5   | dst is double
6 end
7 forall example of the majority class exMaj in d do
8   | forall example of the minority class exMin in d do
9     | dst = dist(exMin, exMaj)
10    | if  $\neg \exists ex \in d | \text{dist}(ex, exMin) < dst \vee \text{dist}(ex, exMaj) < dst$  then
11      |   | cjtTomek := addLink (cjtTomek, <exMin, exMaj>)
12      |   end
13    | end
14 end

```

algorithms since they have been empirically shown to be some of the most competitive re-sampling techniques (Chawla et al., 2002; Batista et al., 2004). Moreover, we introduce a modified version of SMOTE that incorporates a data cleaning process, which we address as *cluster SMOTE* (cSMOTE) (Orriols-Puig and Bernadó-Mansilla, 2008b). As proceeds, each one of these approaches is explained in detail, and their behavior is illustrated in a case study; in the next section, the performance of each one of the re-sampling techniques, in combination with the five learners, is empirically examined.

7.4.1 Random Over-sampling

The first re-sampling technique considered in the comparison is random over-sampling. This is a very simple approach that proposes to over-sample the rare classes in the training data set so as to match the size of the majority class. Although the simplicity of this approach, several authors have demonstrated that it improves the performance of highly-known learners in pattern recognition tasks. Japkowicz and Stephen (2000) showed that random over-sampling, combined with a multi-layer perceptron classifier (Rumelhart et al., 1986), was a very effective method to deal with class imbalances. Later, Japkowicz and Stephen (2002) extended this conclusion to the C5.0 decision tree. Moreover, Batista et al. (2004) experimentally showed that random over-sampling resulted in one of the best improvements in comparison with eleven more sophisticated re-sampling techniques. Due to these excellent results, we included random over-sampling in our experiments.

7.4.2 Under-sampling based on Tomek Links

Under-sampling with Tomek links (Batista et al., 2004) is an under-sampling technique that uses the concept of *Tomek link* (Tomek, 1976) to remove examples of the majority class from the training data set. As follows, we first present the concept of Tomek link and then explain how the re-sampling technique works.

The goal of the Tomek links procedure is to find pairs of examples with different class, but that are geometrically close in the feature space. Algorithm 7.4.1 provides the pseudo code of the algorithm that finds all the Tomek links. That is, a *tomek link* is a pair of examples $\langle E_i, E_j \rangle$ that belong to different classes and for which there does not exist any other example E_k so that $dist(E_i, E_k) < dist(E_i, E_j)$ or $dist(E_j, E_k) < dist(E_i, E_j)$, where $dist$ is a function that computes the distance between two examples.

Therefore, the examples that do not form any Tomek link are not in the decision boundary, and thus, the information that they provide is not as interesting as the information that resides in the examples that form Tomek links. Batista et al. (2004) used this idea and proposed an under-sampling technique that removed examples of the majority class that did not belong to any Tomek link. This technique showed to provide competitive results, especially when combined with other over-sampling methodologies. For this reason, we incorporate this re-sampling algorithm in our analysis.

7.4.3 SMOTE

The *synthetic minority over-sampling technique* (SMOTE), originally designed by Chawla et al. (2002), is one of the most influential re-sampling techniques. SMOTE is an over-sampling technique that creates new minority class instances by means of performing different operations on the minority class instances of the training data set. Therefore, the application of this technique results in a new data set where the presence of the minority class is increased by the creation of new “synthetic” examples of the minority class. As follows, the details of the algorithm are given.

Algorithm 7.4.1 provides the pseudo code for the SMOTE algorithm, which works as follows. For each example of the minority class e_i , the procedure searches for the k nearest neighbors of e_i that also belong to the minority class. Then, it creates N examples of the minority class along the line segments joining any of the k minority class nearest neighbors (the value of N depends on the desired degree of over-sampling). To achieve this, for each new example of the minority class that has to be generated, the algorithm randomly selects one of the k nearest neighbors e_r and creates a new instance in which each attribute is a randomly generated on the segment that joins e_i and e_r .

Chawla et al. (2002) empirically demonstrated the competitiveness of SMOTE with respect to other re-sampling techniques. Subsequent to this publication, several authors proposed new approaches, based on SMOTE, to generate synthetic data. For example, Chawla et al. (2003) combined SMOTE with a boosting technique to improve the detection of rare classes. Later, Han et al. (2005) designed a new approach which mainly used the SMOTE algorithm, but trying to re-sample only those instances that lay closely to the decision boundary. All these new approaches supposed little modifications of the initial idea of creating synthetic data of the

Algorithm 7.4.2: Pseudo code for the SMOTE algorithm.

```

1 Algorithm: SMOTE ( d is Dataset, N is integer, k is integer ) return (dOut is
   Dataset)
   Data: N is the proportion of over-sampling
           k is the number of neighbors considered to create new instances
   Result: dOut is the new re-sampled data set
2 var
3   | numNewNeigh, numMin, i is integer
4   | currEx, newEx, selectedNeigh is Example
5   | att is Attribute
6   | dOut, neighbors is Dataset
7 end
8 numMin := number of instances of the minority class in dOut
9 i := 0
10 dout := d
11 while i < numMin do
12   | currEx := get the ith example of the minority class
13   | neighbors := get the k nearest neighbors of the minority class closer to currEx
14   | numNewNeigh := N
15   | while numNewNeigh > 0 do
16     | selectedNeigh := randomly get a neighbor from neighbors
17     | /* create a new example of the minority class */
18     | forall attribute att do
19       | | newEx[att] := currEx[att] + (currEx[att] - selectedNeigh[att])·rand(0,1)
20     | end
21     | numNewNeigh := numNewNeigh - 1
22     | dout := addExample(dout, newEx)
23   | end
24   | i := i + 1
25 end
26 return dout

```

minority class. In the next section, we propose another modification of the SMOTE algorithm that combines these ideas with a data cleaning phase.

7.4.4 cSMOTE

We now introduce *cluster SMOTE* (cSMOTE), a re-sampling technique based on SMOTE. cSMOTE introduces two main modifications to the SMOTE algorithm, which consist in:

- including a phase to clean instances that are considered noise; and
- disabling the creation of new minority class instances beyond the boundaries of a virtual cluster calculated for each minority class instance.

Algorithm 7.4.3: Pseudo code for the cSMOTE algorithm.

```

1 Algorithm: cSMOTE ( d is Dataset, k is integer ) return (dOut is Dataset)
   Data: k is the number of neighbors considered to create new instances
   Result: dOut is the new re-sampled data set.
2 var
3   | numFavor, numAgainst is integer
4   | currEx, newEx, selectedNeigh is Example
5   | att is Attribute
6   | dOut, neighbors is Dataset
7 end
8 dOut = empty data set
9 forall example currEx in d do
10  | neighbors := et the k nearest neighbors closer to currEx
11  | numFavor := number of neighbors of the same class as currEx
12  | numAgainst := number of neighbors of different class than currEx
13  | if numFavor = 0 then
14  |   | Do not insert currEx into dOut
15  | else
16  |   | if currEx belongs to the majority class then
17  |   |   | dOut := addExample(dOut, currEx)
18  |   | else
19  |   |   | dOut := addExample(dOut, currEx)
20  |   |   | while numAgainst > 0 do
21  |   |   |   | selectedNeigh := select a neighbor from the same class as currEx
22  |   |   |   | /* create a new example of the minority class */
23  |   |   |   | forall attribute att do
24  |   |   |   |   | newEx[att] := currEx[att] + (currEx[att] - selectedNeigh[att]) · rand(0,1)
25  |   |   |   | end
26  |   |   |   | dOut := addExample(dOut, newEx)
27  |   |   |   | numAgainst := numAgainst - 1
28  |   |   | end
29  |   | end
30 end
31 return dOut

```

As follows, we explain each one of these two modifications in detail.

Algorithm 7.4.3 provides the pseudo code for cSMOTE. The algorithm is guided by a main loop over all the examples of the data set, independent of whether they belong to the minority class or to the majority class. For each example e_i , the algorithm selects the k nearest neighbors regardless of their class; besides, it counts the number of these k neighbors that belong to the same class as e_i ($numFavor$) and the number of them that belong to another class ($numAgainst$). Depending on these variables and the selected example e_i , the next steps are taken:

1. If, given e_i , there does not exist any other example among its k nearest neighbors that belongs to the same class, e_i is considered as a noisy instance and it is not included in the final data set.
2. If e_i belongs to the majority class and, at least, there exists another example of the majority class among its k nearest neighbors, e_i is copied to the final data set.
3. If e_i belongs to the minority class, we create as many new examples as the number of nearest neighbors of the majority class that e_i has (i.e., new *numAgainst* instances of the minority class are created). The underlying idea of this re-sampling strategy is the fact that, in general, the number of nearest neighbors of another class would be higher as the minority class instance approaches the class boundary. Thence, this technique aims at introducing more examples of the minority class around the class boundary, but not increasing the number of them in regions of the feature space that are far from this boundary.

The generation of new examples of the minority class is also modified with respect to that in SMOTE. cSMOTE does not allow the creation of examples of the minority class beyond the virtual cluster to which the original example belongs. This virtual cluster is defined as follows. The center of the cluster is determined by the example E_i . Then, the cluster is defined by a sphere with radius equal to the distance to the example E_j , where E_j is the farthest neighbor of E_i for which it does not exist any other instance E_k of another class such as $dist(E_i, E_k) < dist(E_i, E_j)$. Thus, as the new instances are created in the segment defined between two instances of the minority class that belong to this cluster, they would never go beyond the virtual cluster. The behavior of cSMOTE and of the other three re-sampling techniques as well is exemplified in the following subsection.

7.4.5 What Do Re-sampling Techniques Do? A Case Study

Before proceeding with the comparison of the four re-sampling techniques—combined with each one of the five learning methods—on the collection of real-world problems, we first illustrate how these techniques work on a two-dimensional artificial problem. Thence, the purpose of this section is not to extract general conclusions about the re-sampling techniques behavior, but to intuitively explain how they work. As follows, we first introduce the artificial domain used in the case study and illustrate how the four re-sampling techniques modify this domain; then, we depict the knowledge created by the five different learners on this problem.

Artificial Problem Used in the Case Study

To illustrate the behavior of the different re-sampling techniques, we designed the two-dimensional artificial problem shown in figure 7.5(a). The problem consists of four concepts of the minority class (red dots) that have a circular shape and are distributed around the feature space. Moreover, the two concepts of the minority class placed in the top of the feature space contain a sub-concept of the majority class inside them, drawing the shape of a “doughnut”. Therefore, this data set shows that some of the small disjuncts belong to the majority class, highlighting

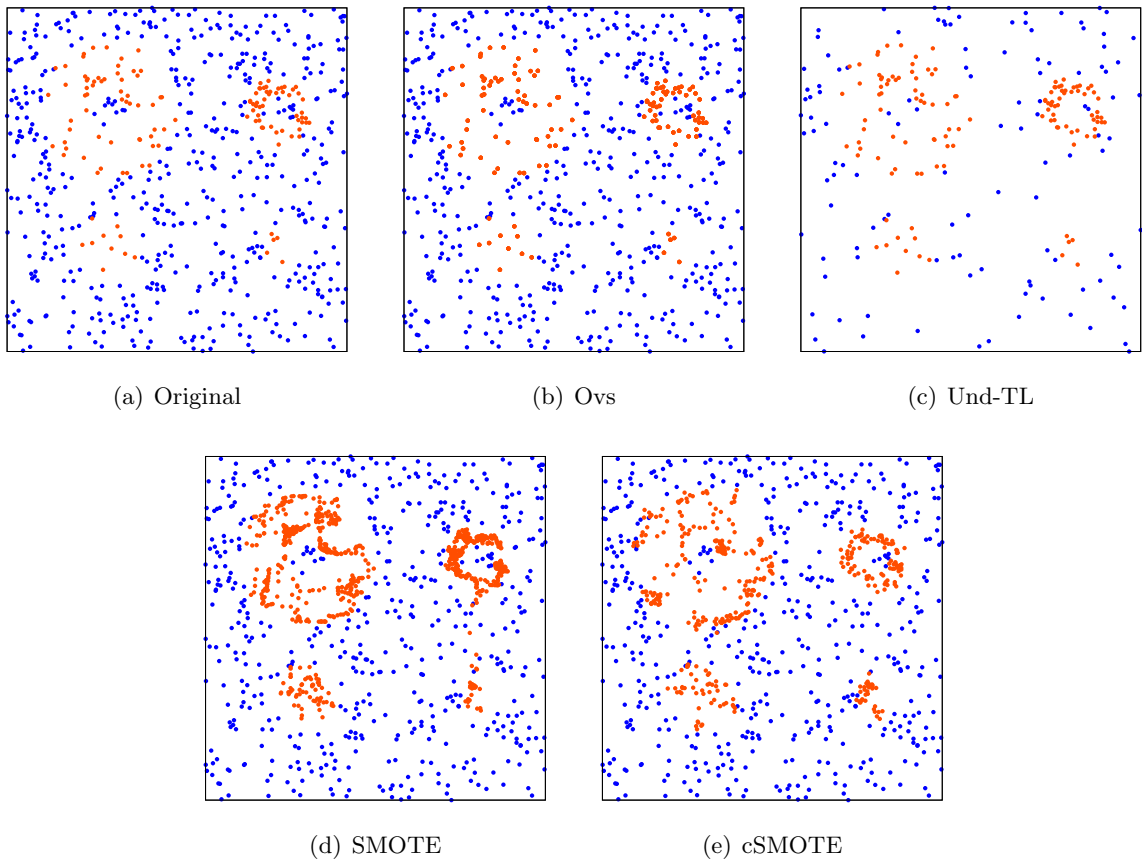


Figure 7.5: Original domain (a) and domains after applying random over-sampling (b), under-sampling with Tomek links (c), SMOTE (d), cSMOTE (e).

that there is not always a direct mapping between the imbalance ratio of the training data set and the imbalance ratio among the sub-solutions or clusters in the solution space.

Figures 7.5(b), 7.5(c), 7.5(d), and 7.5(e) show the modified data set after applying random over-sampling, under-sampling with Tomek links, SMOTE³, and cSMOTE⁴ respectively. Random over-sampling replicates some of the training instances until the data set contains the same number of instances of both classes; for this reason, the resulting domain is apparently equal to the original domain. Under-sampling based on Tomek links removes instances of the majority class that are not close to the class boundary. Note that the final data set contains a considerable lower number of examples. This may be beneficial in terms of run time of the final learner, especially when the original data set consists of a large number of instances; nonetheless, in domains with few instances, it may result in a problem of sparsity. SMOTE creates new instances of the minority class by interpolation. A potential problem of this technique is that, depending on the size of the small disjunct and the chosen k , SMOTE can generate noisy instances if one

³SMOTE was configured with $k=N=5$

⁴cSMOTE was configured with $N=10$

of the nearest neighbors selected to create a new example belongs to another disjunct. cSMOTE returns a domain that is similar to the one generated by SMOTE. The main difference is that the minority class instances generated by cSMOTE are closer to the class boundary. In the next section, we show the models evolved by the different learners on the original and the re-sampled data sets.

Models Built by the Learners

Figure 7.6 illustrates the domain learned by the five learners on the original and the re-sampled data sets. These results are complemented by table 7.4, which provides the TP rate and TN rate, measured on the training data set, of each learner and domain. The same configurations of the five learners used in the previous sections were employed for these experiments. The domains are depicted by exhaustively testing all the feature space. That is, we generated a test problem with ten million instances distributed uniformly around the feature space, and we used each learner to predict the class of each instance; then, we depicted a point—with a different color depending on the predicted class—in the solution subspace.

Several observations can be drawn from these results. We first analyze the behavior of the different learners on the original data set. In this case, note that all the learners, except for SMO, could discover totally or partially all the sub-concepts of the minority class. IBk (see figure 7.6(k)) is the learner that resulted in the model that is probably closer to the model that a human expert would define from the set of points depicted in Figure 7.5(a). On the other hand, C4.5, XCS, and UCS discovered concepts whose shape resembled rectangles. This is due to the knowledge representation employed by each learner. That is, C4.5, XCS, and UCS (see figures 7.6(a), 7.6(p), and 7.6(u)) used a knowledge representation based on hyper rectangles to discriminate between classes; for this reason, they had more difficulties to approximate curved boundaries. Despite this, note that these learners, and especially UCS, provided an accurate approximation of the class boundary for the given problem. On the other hand, SMO (see figure 7.6(f)) used a linear kernel which failed to discriminate between classes. Note that SMO predicted that any instance in the input space belonged to the minority class. We tried polynomial kernels with higher degree, but significantly better results were not found for this particular artificial problem. The first row of table 7.4 complements these visual results by reporting the TP rate and the TN rate, measured on the training instances, achieved by the five learners. The table shows that UCS predicted all the training instances correctly; IBk, C4.5, and XCS also yielded accurate results.

Let us now examine the results obtained with the re-sampling techniques. The figures show that, in general, all the learners, except for SMO, benefited from re-sampling the training data set. XCS especially benefited from random over-sampling and SMOTE and, to a lower extent, from cSMOTE. However, note that random over-sampling and under-sampling based on Tomek links result in some uncovered regions in the feature space (white regions in the figures). UCS seems to give the best results with cSMOTE. It is worth noting that UCS evolves a maximally accurate model with the original data set.

In general, SMOTE and random over-sampling were the most effective re-sampling techniques. But this general behavior needs to be analyzed carefully. For example, when C4.5 was trained with the domains under-sampled with Tomek links, and re-sampled with SMOTE, or cSMOTE, it was not able to identify the first majority class sub-concept. This behavior can

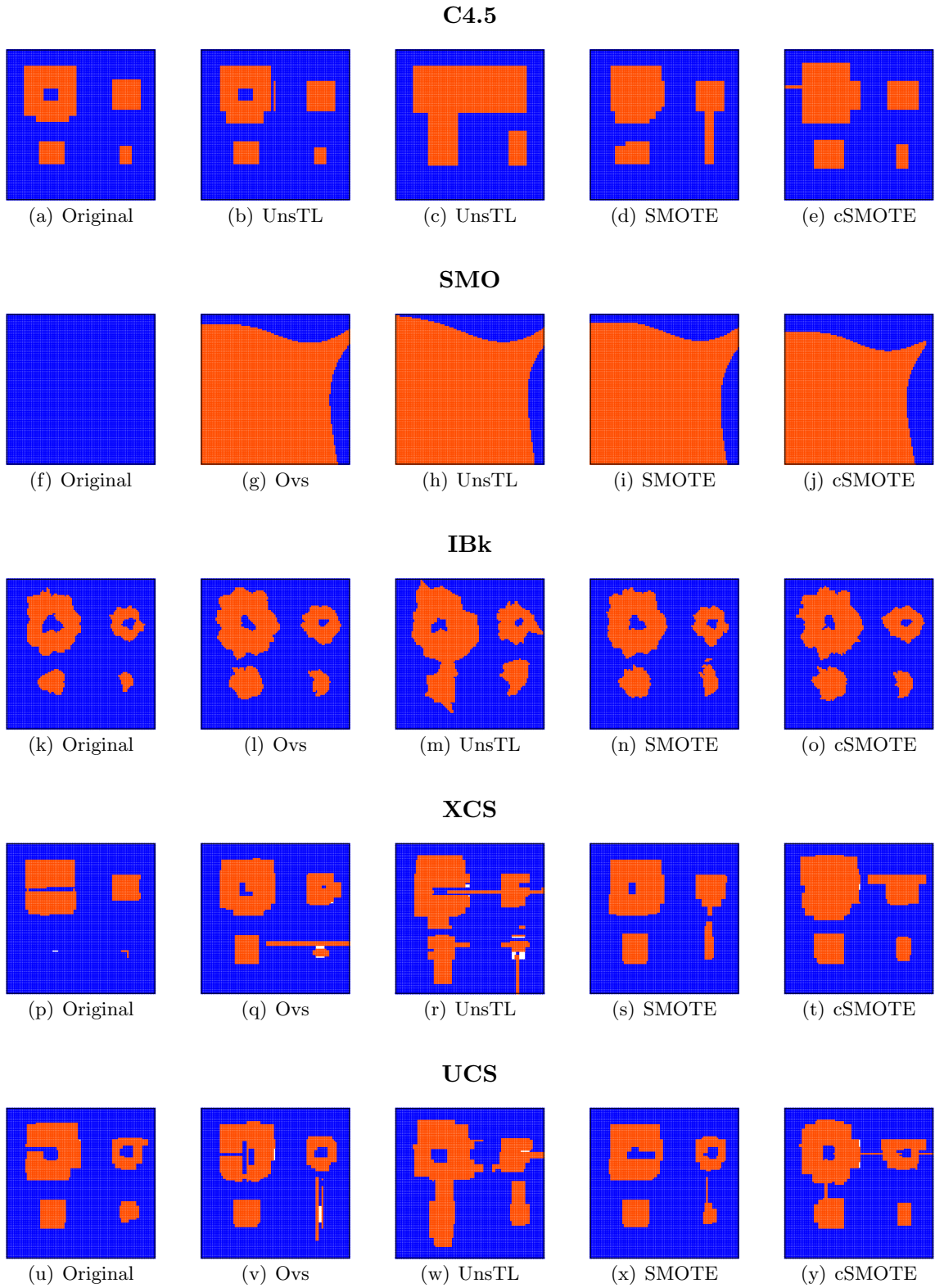


Figure 7.6: Models created by C4.5, SMO, IBk, XCS and l'UCS with the original and the re-sampled data sets.

Table 7.4: TP rate (TPR) i TN rate (TNR) obtained by C4.5, SMO, IBk, XCS and UCS with the original domain and the re-sampled data sets.

	Original									
	C4.5		SMO		IBk		XCS		UCS	
	TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR	TPR	TNR
<i>Original</i>	98.40	97.31	0.00	100.00	95.20	98.85	85.60	96.15	100.00	100.00
<i>Ovs</i>	100.00	97.31	100.00	29.04	100.00	94.04	100.00	96.73	100.00	99.62
<i>UnsTL</i>	99.20	79.49	100.00	28.21	99.20	91.45	99.20	90.58	100.00	88.08
<i>SMOTE</i>	98.40	97.50	100.00	30.00	99.40	97.50	98.40	97.50	100.00	100.00
<i>cSMOTE</i>	96.22	97.27	100.00	29.43	97.84	98.54	95.20	98.27	92.80	99.62

be observed in other problems, where the application of a re-sampling technique induced the learner to misclassify regions that belonged to the majority class. In the case of SMO, there may be some intrinsic complexities of the domain that created especial difficulties to the system.

The results provided here visually illustrated the behavior of the different re-sampling techniques, giving a more detailed insight on how they work. Nonetheless, conclusions cannot be extracted from this simple case study. In the next section, we extend the analysis and compare the four re-sampling techniques in combination with the five learning algorithms on the large collection of imbalanced real-world problems used in the previous section.

7.5 Results on Re-sampled Domains

In this section, we analyze whether the application of the four re-sampling techniques presented above improves the performance of the five learning methods. As proceeds, we first introduce the experimental methodology, especially focusing on the steps taken to generate the re-sampled data sets. Then, we summarize the results obtained for each learning method and extract general conclusions about the excellence of each re-sampling technique. The full results are provided in appendix C.

7.5.1 Experimental Methodology

To compare the effect of the re-sampling techniques on each learning method, we employed the following methodology. We applied each re-sampling technique to each one of the training folds of the 25 data sets presented in section 7.3.1. This resulted in 100 new data sets, each one with 10 re-sampled training folds. The test folds were not modified so that the learners were tested with exactly the same information used in the original experiments (see section 7.3.2).

The five learning methodologies were configured as specified in section 7.3.1. No particular configuration was set for each re-sampling data set since we aimed at analyzing the impact of these re-sampling techniques to the system. The performance of each learner was measured with the product of TP rate and TN rate, since this metric is not biased toward the imbalance ratio of the learning data set. Also, the statistical procedure followed in section 7.3.1 was employed to compare the results. That is, the multiple-comparison non-parametric Friedman

test (Friedman, 1937, 1940) was applied to contrast the null hypothesis that all the re-sampling techniques yielded the same results, on average, for a specific learner. If the multiple-comparison test rejected the null hypothesis, the post-hoc Nemenyi test (Nemenyi, 1963) was employed to detect significant differences among techniques.

With the purpose of compactness, the two next subsections present a summary of the results from which we extract general conclusions about the competitiveness of each re-sampling technique. We first provide the statistical analysis of the comparison of the four re-sampling techniques for each learner. The complete tables of results are supplied in appendix C. Thereafter, we summarize all the results in a table that gathers the average rank of each re-sampling technique for each of the five learning systems and highlight the main ideas and key conclusions from this summary.

7.5.2 Statistical Analysis of the Results

As proceeds, we statistically analyze the results of the comparisons of the four re-sampling techniques for each learner.

C4.5. The multiple-comparison Friedman test rejected the null hypothesis that the results obtained with the different re-sampling techniques were equivalent, on average, with $p = 0.0018$. Thus, we applied the post-hoc Nemenyi test, at $\alpha = 0.10$, to detect groups of re-sampling techniques which yielded equivalent results. Figure 7.7(a) connects the groups of learners that performed equivalently according to the Nemenyi test at $\alpha = 0.10$. Note that the test detected two groups. The first group comprised SMOTE, random over-sampling, and the original data set. The best ranked method was SMOTE.

The second group consisted of all the techniques except for SMOTE. Notice that the poorest results were obtained with the under-sampled data sets. This denoted that under-sampling the majority class might lead to sparsity problems in the particular data sets of the comparison since, on average, they contain a low number of instances. In general, the statistical analysis confirms the suitability of SMOTE and random over-sampling in combination with C4.5.

SMO. The multiple-comparison Friedman test rejected the hypothesis that all the re-sampling techniques resulted in the same performance, on average, with $p = 2.98 \cdot 10^{-6}$. Therefore, we applied the post-hoc Nemenyi test, whose results are provided in figure 7.7(b).

Several conclusions can be drawn from these results. Firstly, SMO and C4.5 benefited from different re-sampling techniques. In SMO, the best re-sampling technique was random over-sampling, which was also one of the best methods in C4.5. Nonetheless, notice that the second best ranked re-sampling method was under-sampling based on Tomek links, which, combined with C4.5, resulted in the poorest results. This highlights the idea that different learners benefit from different re-sampling methods.

Secondly, the statistical study detected that random over-sampling significantly outperformed the results obtained with the re-sampled data sets and the original data set. The Nemenyi test did not detect any further significant difference among the remaining re-sampling techniques and the original data set. Nevertheless, it is worth noting that the poorest average rank was obtained with the original data sets, which indicates that all re-sampling techniques are, on average, beneficial to SMO.

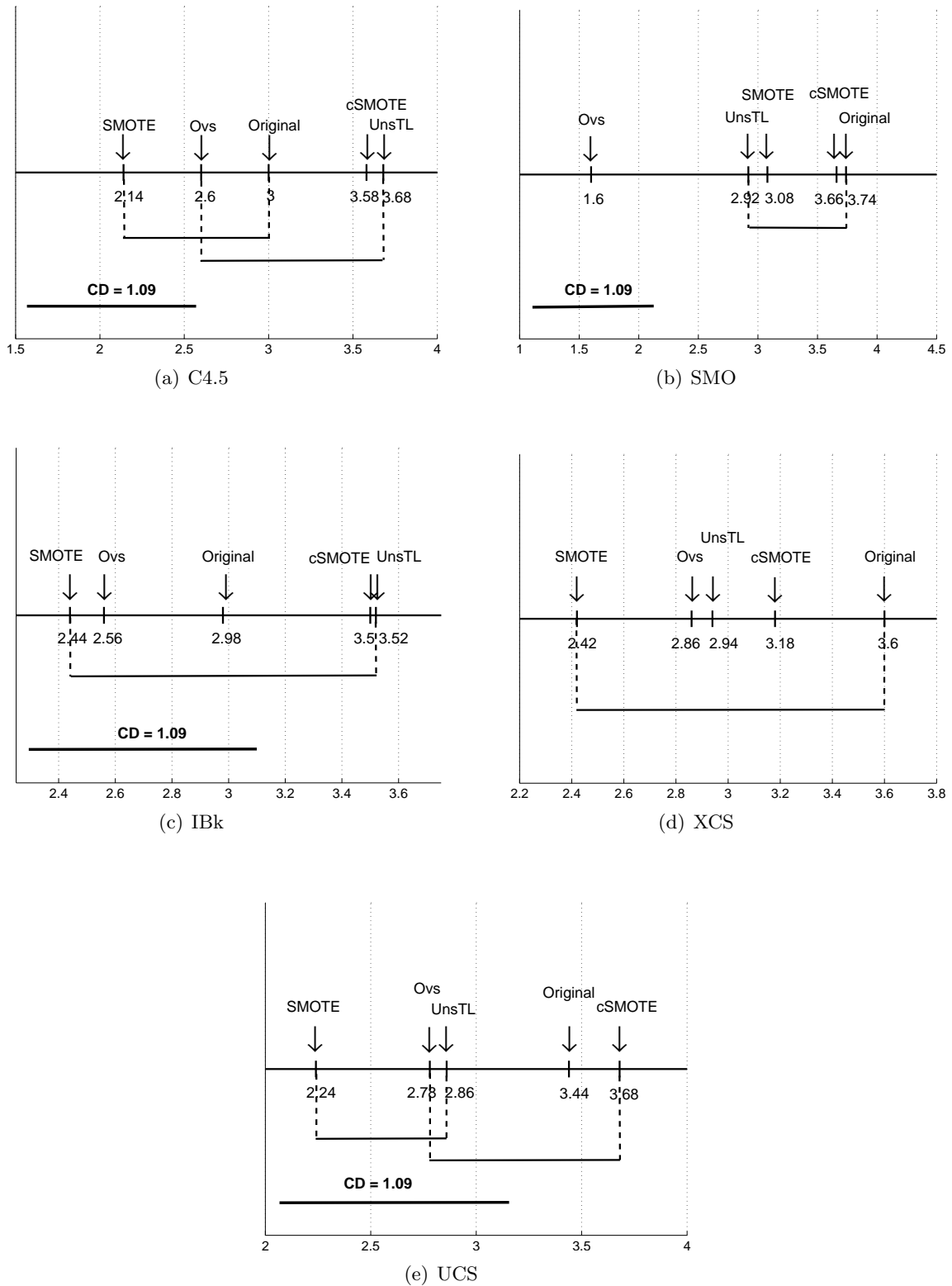


Figure 7.7: Comparison of the performance obtained by (a) C4.5, (b) SMO, (c) IBk, (d) XCS, and (e) UCS with the different re-sampling techniques. Groups of classifiers that are not significantly different at $\alpha = 0.10$ are connected.

IBk. The multiple-comparison Friedman test rejected the hypothesis that IBk obtained equivalent results with the original and re-sampled data sets with $p = 0.03$. However, the Nemenyi test, at $\alpha = 0.10$, could not find significant differences among the re-sampling methods. As follows, we provide some observations based on the average rank of each re-sampling algorithm, which is supplied in figure 7.7(c).

SMOTE was the best ranked method of the comparison, closely followed by random over-sampling. In the next positions of the ranking, we find the original data set, cSMOTE, and under-sampling based on Tomek links. Thence, under-sampling based on Tomek links provided, again, the poorest results. This was probably due to a problem of sparsity in the under-sampled data sets. Note that the position in the ranking of each re-sampling technique equals the corresponding position obtained by C4.5, although in C4.5 some of the differences were statistically significant.

XCS. The multiple-comparison Friedman test did not permit rejecting the hypothesis that the results obtained with the original data sets and the re-sampled data sets were equivalent, on average, at $\alpha = 0.05$ (the p-value returned by the test was $p = 0.1037$). Thence, as follows, we provide some remarks based on the average rank of each technique, which are illustrated in figure 7.7(d).

As observed for C4.5 and IBk, the two best ranked re-sampling techniques were SMOTE and random over-sampling. The third best method in the comparison was under-sampling with Tomek links. Thus, differently from IBk and C4.5—where under-sampling with Tomek links provided the worst performance—, the experimental results point out that XCS was not affected, on average, by decreasing the number of training examples, given that the points that lay close to the boundary were included in the final data set. Notice that the results achieved with the original data set were never significantly superior than those obtained with the under-sampled domain. A similar observation was drawn for SMO. This highlights that the competence of under-sampling with Tomek links was really dependant on the final learning system employed to learn the data model.

cSMOTE and the original data set come in the last positions of the ranking. Finally, note that the worst results were achieved with the original data sets. Therefore, as in SMO, all the re-sampling techniques appeared to improve the results obtained by XCS with the original data sets.

UCS. The multiple-comparison Friedman test rejected the null hypothesis that all the models extracted with the different re-sampled data sets and the original data set were equivalent, on average, with $p = 0.01$. Figure 7.7(e) groups the learners that performed equivalently according to the Nemenyi test at $\alpha = 0.10$.

The statistical analysis identified the same three best ranked methods as in XCS: SMOTE, random over-sampling, and under-sampling based in Tomek links. Therefore, the same conclusions provided in XCS can be extended to UCS. The poorest results were obtained with the original data sets and cSMOTE.

After analyzing each particular learner, the next section summarizes the results, gathering some key conclusions about the excellence of the different re-sampling techniques.

Table 7.5: Intra-method ranking for original and re-sampled data sets for C4.5, SMO, IBk, XCS, and UCS. Rows 1st to 5th indicate the number of times that each re-sampling technique was ranked in the correspondent position. The last column shows the average rank and its standard deviation.

	Resamp. Method	1st	2nd	3rd	4th	5th	Avg. \pm Std.
C4.5	<i>original</i>	6	3	4	9	3	3.00 ± 1.39
	<i>oversampling</i>	7	4	8	4	2	2.60 ± 1.26
	<i>undersampling TL</i>	0	5	6	6	8	3.68 ± 1.12
	<i>smote</i>	9.5	7.5	4	3	1	2.14 ± 1.17
	<i>csmote</i>	2.5	5.5	2	5	10	3.58 ± 1.44
SMO	<i>original</i>	4	3	1.5	3.5	13	3.74 ± 1.56
	<i>oversampling</i>	12	11	2	0	0	1.60 ± 0.63
	<i>undersampling TL</i>	3	6.5	8	4.5	3	2.92 ± 1.18
	<i>smote</i>	3	4.5	8.5	5.5	3.5	3.08 ± 1.20
	<i>csmote</i>	2	1	4	14.5	3.5	3.66 ± 1.03
IBk	<i>original</i>	6	5	2.5	6.5	5	2.98 ± 1.49
	<i>oversampling</i>	3.5	8.5	9.5	2.5	1	2.56 ± 0.98
	<i>undersampling TL</i>	4	2	6	3	10	3.52 ± 1.47
	<i>smote</i>	10.5	3.5	2.5	6.5	2	2.44 ± 1.44
	<i>csmote</i>	1	5	5.5	7.5	6	3.50 ± 1.17
XCS	<i>original</i>	3	4	2.5	6	9.5	3.60 ± 1.43
	<i>oversampling</i>	7	6	3	1.5	7.5	2.86 ± 1.61
	<i>undersampling TL</i>	1	6	11.5	6.5	0	2.94 ± 0.81
	<i>smote</i>	11	4	1.5	5.5	3	2.42 ± 1.51
	<i>csmote</i>	3	4	7.5	6.5	4	3.18 ± 1.23
UCS	<i>original</i>	2	4	6	7	6	3.44 ± 1.24
	<i>oversampling</i>	5.5	5.5	6	5	3	2.78 ± 1.32
	<i>undersampling TL</i>	5	4	8	5.5	2.5	2.86 ± 1.25
	<i>smote</i>	7	11	3	2	2	2.24 ± 1.18
	<i>csmote</i>	5.5	0.5	1	7.5	10.5	3.68 ± 1.55

7.5.3 Summary

Having compared the performance obtained with each learner with the original and re-sampled data sets, the aim of this section is to summarize all the experimental analysis. For this purpose, table 7.5 supplies, for each learner, the number of occurrences of each re-sampling technique in each position of the ranking. For each classifier, the re-sampling method that is placed first in the ranking is marked in bold. The last column provides the average rank and the standard deviation for each re-sampling method, which are used to highlight the main observations and conclusions of the present comparison.

The results show that, in general, data set re-sampling yielded better learning performance than the one reached with the original data set. On average, the best results were achieved with *SMOTE* and *random over-sampling*. The empirical observations agree with some studies concluding that over-sampling is more effective than under-sampling in C4.5 (Japkowicz and

Stephen, 2002; Batista et al., 2004) and SMO (Japkowicz and Stephen, 2002). The results obtained herein allow us to extend this conclusion to IBk, XCS, and UCS. We hypothesize that under-sampling may cause a problem of sparsity as it removes instances that may be needed for learning. In fact, a deeper inspection of the detailed results (see appendix C) shows that under-sampling was better ranked in the problems *pim*, *wavd1*, *wavd2*, and *wavd3*, which have the highest number of instances per dimension⁵, and poorly ranked in the problems with the lowest number of instances per dimension: *wdbc*, *wined1*, *wined2*, *wined3*, and *wdbc*.

The standard deviation of the rank somehow denotes the dependency of each re-sampling method on the characteristics of the training domain. For C4.5, SMOTE was the best ranked re-sampling method; besides, the rank deviation was small. In most of the cases, SMOTE was the first or the second method in the ranking. These results indicate that SMOTE should be used in combination with C4.5 to deal with class imbalances. For SMO, random over-sampling was the best ranked method and, at the same time, it showed a very low standard deviation. Consequently, SMO should be combined with random over-sampling in imbalanced domains. For IBk, SMOTE was the best ranked re-sampling technique followed very closely by random over-sampling. However, SMOTE had a much higher standard deviation, which indicates that its behavior highly depends on the domain. Therefore, this promotes the use of over-sampling in combination with IBk if we search for better robustness. For XCS, the best ranked re-sampling method, i.e., SMOTE, had one of the highest standard deviations. Thus, the behavior of this combination depended on the characteristics of the data. In this case, the practitioner may prefer to combine XCS with under-sampling based on Tomek Links, since its average rank was close to the SMOTE one and it had a very low standard deviation. For UCS, the best and the most robust re-sampling method was SMOTE.

Although the average results promote the application of re-sampling techniques to imbalanced domains, let us highlight that, in some cases, the best results were achieved with the original data set. That is, a detailed inspection of the results for each particular data set (see the full tables in appendix C) reveals that the performance of some learners on particular problems worsened when the data sets were re-sampled. This happened in problems such as *h-s*, *tao*, *wined1*, *wined2*, and *wined3*. In all these cases, some learners obtained a significantly lower performance than the one obtained with the original data set, which indicated that re-sampling was introducing some undesired characteristics to the training domain such as noise or new small disjuncts or niches. Having this in mind, the next section opens a discussion about several important aspects that have been made manifest throughout the comparative study, the answer of which will lead us to future work lines.

7.6 Discussion

The comparison performed in this chapter provided many valuable insights and enabled us to identify which learning systems were better than others and which combinations of learning algorithms with re-sampling techniques yielded the most accurate models on average. Nonetheless, we already pointed out a set of particular cases where the application of re-sampling techniques not only did not result in any improvement, but also provided significantly less accurate mod-

⁵The ratio between the number of instances and the number of attributes of a problem has been proposed elsewhere (Bernadó-Mansilla and Ho, 2005) as a measure of sparsity.

els. In general, after analyzing in detail all the results of the present comparison, the machine learning practitioner, who needs to solve a new problem, may still be wondering which learning algorithm and re-sampling technique he or she should use. In fact, these thoughts could be articulated in a more general way by posting the following two questions:

- *Why do some re-sampling techniques work well for some learners and data sets but fail with others?*
- *Why could not we find a re-sampling technique that works the best for all learners?*

These two questions are of especial interest in the current days, since the maturity of machine learning has resulted in the design of several algorithms to solve the same types of problems. Studies that mathematically analyze the properties of learning algorithms are not rare in the literature. However, they still cannot predict which learning algorithm will be the best performer for a new real-world problem, as long as the intrinsic characteristics of this problem are not known.

A particular contribution that provides a nice observation about these two questions can be found in the no-free-lunch (NFL) theorem (Wolpert, 1992, 1996). Loosely speaking, the NFL theorem mathematically demonstrates that, if we consider all the possible domains in our data set space, we can find as many data sets for which a learning algorithm ‘A’ outperforms another algorithm ‘B’ as viceversa. This theorem should be taken with a grain of salt, since we are not interested in all the possible domains, but in those domains that can be found in real-world problems. For example, we are not interested in domains with randomly generated instances, but in domains in which their instances define real-world concepts. Nevertheless, the NFL theorem gives mathematical formulation to a conclusion already observed in our experiments: that we cannot find a learning algorithm that performs the best for all possible domains, but for a particular range of them. Therefore, this conclusion demands the characterization of the real-world domains to relate them to the properties of each learning system, and so, to detect for which type of problems a learning algorithm is better suited than the others.

Some recent efforts have been taken to design measures to characterize classification problems. Michie et al. (1994) early highlighted the importance of domain characterization to analyze the performance of machine learning methods. The authors provided a set of measures—which consisted of statistical indicators and measures coming from the information theory—to characterize classification domains. These metrics were designed especially focusing on the analysis of decision trees. Later, Sohn (1999) used a more restricted set of metrics in a meta-model that compared the classification performance of several learning systems in terms of the data characterization. Although the results were promising, the authors already pointed out the need for further developing new metrics to capture more characteristics of the learning domains.

After these first promising works, Ho and Basu (2002) carefully examined the possible sources of data complexity and defined a set of measures that aimed at extracting some *geometrical characteristics* of the class distributions in the training data set. It was empirically shown that there was a considerable correlation between some of these measures and the error of some classifiers. For example, Bernadó-Mansilla and Ho (2005) and Bernadó-Mansilla et al. (2006) showed that the error of XCS was correlated with some metrics that estimated the length and linearity of the class boundary. The experimental results provided in these works evidenced that the characterization of the training data sets holds promise, being able to explain the behavior

of learning algorithms on particular data, and so, starting to bridge the gap between artificial data sets with known characteristics—for which we have developed different models in chapters 5 and 6—and real-world data sets with unknown characteristics. Nonetheless, some drawbacks were also detected, such as the incapacity of the collection of designed metrics to explain all the sources of complexity of a classification problem.

As further work, this thesis proposes to continue with the study and design of new complexity metrics and apply them to

1. examine for which types of data a learning algorithm outperforms the others,
2. analyze the impact, from a geometrical point of view, of applying the re-sampling techniques to the different data sets, and
3. study the feasibility of applying each re-sampling technique with each particular learning system.

The aim of each one of these aspects is explained in what follows.

First, we will employ the complexity metrics to analyze which geometrical characteristics affect the different learners, identifying the sweet spot in which each learner is the best performer. Thence, we aim at answering questions such as “*Which learner for which real-world problem?*”. To achieve this, an API that includes all the complexity metrics proposed by Ho and Basu (2002), extends their definition enabling their application to data sets with continuous and nominal attributes and multiple classes, and provides new measures was implemented and made available as open source code (Orriols-Puig et al., 2008b). In addition, the correlation of different complexity metrics with the test error of XCS was analyzed (Bernadó-Mansilla et al., 2006). In further work, we aim at extending the analysis to other learners.

Second, the impact that each re-sampling technique has on the original data set—which has been visually illustrated for a particular case study in section 7.4.5—will be analyzed from the point of view of the change of the geometrical structure that the re-sampling techniques cause. That is, re-sampling techniques change the distribution of the training data set since new instances are added or some of the existing instances are removed from the original data set. Thus, in further work, it will be interesting to study how these re-sampling methods change the original distributions.

This change in the initial distribution may be either beneficial or detrimental for the learner employed in each particular case depending on the change of the initial distribution. For example, suppose that we over-sample a linearly-separable data set and that the resulting data is no longer linearly separable. In this case, the results obtained by a linear classifier in the re-sampled data set will be probably worse than those achieved with the original data. This is because there would be a misalignment between the re-sampling technique—and the changes that it produces to the original data—and the learning heuristic. Therefore, the aim of the third future work line is to identify which re-sampling technique is best suited for a particular learning algorithm given a real-world problem with a certain apparent complexity, providing answer—or, at least, some guidelines—to the question of “*Which machine learning technique combined with which re-sampling technique for which problem?*”.

7.7 Summary and Conclusions

In this chapter, we moved to real-world imbalanced classification problems and showed that both LCSs can successfully deal with the challenges posed by learning accurate models from rare classes in continuous domains with unknown characteristics. We designed a self-adaptation mechanism that uses a heuristic procedure to detect the maximum imbalance ratio between niches that lay closely in the solution space and adjust the parameters of both LCSs according to the theory presented in the last chapter. The excellence of the two LCSs was made evident through a comparison with three of the most influential machine learning techniques. XCS and UCS were the two best ranked methods in the comparison, providing statistically better results than the other methods in a considerably large number of data sets. Later, we introduced four re-sampling techniques in the comparison and analyzed the improvements that they supplied in combination with each one of the five learning techniques, extracting several observations for each particular case.

Two main conclusions, as well as several future line works, can be extracted from the overall analysis of this chapter. The first conclusion is that the two LCSs are two of the best options to extract classification models from imbalanced data sets, since they presented the best rank on average. Moreover, although it has not been further discussed here, LCSs have two important assets that differentiate them from the other three learning methods: (1) their online learning architecture and (2) their rule-based representation. The online learning architecture enables LCSs to learn from streams of data, which are very common in current scientific and industrial applications (Aggarwal, 2007; Gama and Gaber, 2007). The rule-based representation permits extracting models that can be read by human experts to some extent. This is similar to C4.5, which creates decision trees. Nonetheless, note the difference with respect to SMO and IBk. The models built by SMO consist of a set of support vector machines defined by weights, which can be barely interpreted. On the other hand, IBk does not create a general model.

The second conclusion is that, on average, re-sampling techniques improve the discovery of rare classes. In particular, SMOTE and random over-sampling appeared as the two best re-sampling techniques. Therefore, according to the empirical evidence provided in the present comparison, the machine learning user may bet for a combination of LCSs and SMOTE or over-sampling to deal with new challenging imbalanced problems. Nevertheless, we have also discussed that the intrinsic complexities of each particular problem may need different treatments. This leads us to consider the study of domain characterization as further work.

Chapter 8

Fuzzy-UCS: Evolving Fuzzy Rule Sets for Supervised Learning

In the last three chapters, we studied the capabilities of XCS and UCS to learn from imbalanced domains, addressing the first challenge proposed in this thesis. Along the study, as we were concerned about modeling rare classes, we evaluated the quality of the learners with metrics related to the accuracy per class. Nevertheless, in addition to model's accuracy, human experts may require the creation of legible models so that they can understand why a particular machine learning technique returns an output for a given unlabeled example. In some domains, such in medical diagnosis, human experts are sometimes more interested in the explanation that yields a prediction than in the prediction itself (Robnik-Sikonja et al., 2003). Although both LCSs use a rule-based representation, the interpretability of their models can be impaired by (1) the large number of rules that they tend to create (Bernadó-Mansilla and Ho, 2005; Bacardit and Butz, 2004; Wilson, 2002a; Dixon et al., 2004; Fu et al., 2001) and (2) the use of reasoning mechanisms that may be counter-intuitive to human experts. This is not a particular problem of LCSs, but it is shared by other machine learning techniques. In order to solve it, several authors have proposed the use of fuzzy logic (Zadeh, 1965, 1973) to create machines that can extract legible models and that use reasoning mechanisms closer to human ones.

The purpose of this chapter is to incorporate fuzzy logic concepts into LCSs with the aim of letting the systems evolve more legible classification models and use principled reasoning mechanisms defined in the fuzzy set theory. With this objective in mind, we take a creative process to mix the ideas that come from both fields and design Fuzzy-UCS (Orriols-Puig et al., 2007a,b, 2008c,f), the first Michigan-style LCS that evolves a fuzzy representation online for supervised learning tasks. The system is inspired by UCS, but it is completely redesigned to be able to deal with the new fuzzy representation. The competitiveness of the system is shown by comparing it with a large collection of fuzzy and non-fuzzy systems, which contains several of the most influential learners (Wu et al., 2007). In addition, we illustrate the added value of the online learning architecture of Fuzzy-UCS with respect to other learners by using Fuzzy-UCS to mine large volumes of data online.

The remainder of this chapter is organized as follows. Section 8.1 further discusses on the motivation of the present work. Section 8.2 introduces the basic concepts of fuzzy logics, presents some approaches in which GAs and fuzzy logics have been combined to create machine learn-

ing systems, and reviews some particular LCSs that use a fuzzy representation, highlighting the key differences with respect to Fuzzy-UCS. Section 8.3 gives a detailed description of the proposed Fuzzy-UCS algorithm. Section 8.4 examines the sensitivity of Fuzzy-UCS to configuration parameters. Then, section 8.5 studies the limitations that a linguistic representation may impose and compares it to a more flexible fuzzy representation. Section 8.6 makes an extensive comparison of the Fuzzy-UCS representation with a set of fuzzy and general-purpose machine learning techniques, analyzing the differences between the learners in terms of test performance and interpretability of the models. Section 8.7 exploits the online architecture of Fuzzy-UCS to mine a large data set, the 1999 KDD Cup intrusion detection data set. Finally, section 8.8 gives a summary of the work and future lines of research and presents a SWOT analysis to reflect the strengths, the weaknesses, the opportunities, and the threats of the new system.

8.1 Why Using Fuzzy Logic in LCSs?

Pattern recognition (Theodoridis and Koutroumbas, 2006) is concerned with the design of algorithms that are able to extract novel, useful, and hidden patterns from repositories of data. In this context, a competent supervised learning technique is required to be able to (1) identify patterns hidden between a set of descriptive attributes and a dependent variable, i.e., the output or the class, (2) represent these patterns in some legible structure, and (3) generalize over the input patterns to produce compact representations. During the last few decades, a lot of research has been conducted to design approaches that totally or partially fulfill the aforementioned requirements (Mitchell, 1997, 2006). As proceeds, we discuss whether these three aspects are satisfied by Michigan-style LCSs and propose the use of fuzzy logic as a powerful mechanism to create highly legible models from domains with uncertainty and imprecision.

We have shown in the previous chapters that Michigan-style LCSs are one of the most competitive alternatives to generalize over the input patterns and extract highly accurate models from real-world data sets. Therefore, they fulfill the first and third requirement. Nonetheless, the second requirement is not completely achieved. That is, although most of the current implementations of Michigan-style LCS use a rule-based representation—and rules can be individually interpreted—the readability of the whole rule set may be impaired since LCSs

1. tend to evolve models with a large number of overlapped semantic-free interval-based rules, and
2. use reasoning mechanisms that can be little intuitive to human experts.

As follows we further discuss these two arguments in the context of interval-based rule representation, since it is the most used representation to deal with continuous attributes in LCSs.

The first reason that may hamper the readability of the models evolved by Michigan-style LCSs is that these systems tend to evolve a large number of overlapped semantic-free interval-based rules to define complex class boundaries (Bernadó-Mansilla and Ho, 2005; Bacardit and Butz, 2004; Wilson, 2002a; Dixon et al., 2004; Fu et al., 2001). That is, XCS and UCS systems alike usually represent continuous attributes with semantic-free intervals. Here, we use the term semantic free to refer to the fact that the lower and upper limits of each interval of each rule are modified independent of the value of the same attribute in other rules or the value of other

attributes. This has two negative effects on the readability of the final population. The first negative effect is that Michigan-style LCSs tend to evolve populations that contain rules which are highly overlapped. Particularly, the class boundary usually consists of a large number of overlapping rules that predict different classes (Bernadó-Mansilla and Ho, 2005). This is not only caused by the rule representation, but also by the online learning architecture; that is, as rules are evaluated online, similar rules are maintained in the population. This large number of rules may hamper the readability of the rule set. The second effect comes directly with the fact that attributes are defined by intervals, so defining abrupt boundaries of the region where the rule is applicable. That is, inside the covered region, the implication predicted by the rule is completely true. This is counterintuitive from a human point of view, seeming more reasonable to have rules in which the degree of matching decreases as the boundaries of the covered region are approached.

The second aspect that may have a negative influence in providing legible explanations to human experts is that the reasoning mechanisms used by LCSs to infer the class of test instances usually mix information of all the matching rules to predict the output class. For example, in the case of XCS, the reasoning mechanism is based on a weighted sum that involves the prediction, the fitness, and, indirectly, the numerosity—since it is included in the fitness—of each matching classifier. Therefore, this reasoning mechanism further impairs the interpretability of individual rules because the contribution of each individual rule to the final prediction is not clear.

These two problems are not only related to LCSs, but to many machine learning techniques. To improve models legibility, several authors have adhered to the use of fuzzy logic and fuzzy set theory (Zadeh, 1965, 1973) to define fuzzy systems, that is, systems that use fuzzy logic to create highly legible models that predict environments with uncertainty and imprecision. Basically, the fuzzy set theory provides a legible knowledge representation and a robust reasoning mechanism with a mathematical formulation. In the last few decades, fuzzy logic set theory has been applied to different machine learning techniques such as rule-based systems and GBML (Cordón et al., 2001a) or neural networks (Buckley and Hayashi, 1994, 1995).

The motivation of the work performed in this chapter is to design a new supervised machine learning technique which combines the ideas of accuracy-based LCSs, GAs, and fuzzy logic together. That is, the new approach will join the online evaluation capabilities of LCSs, the search robustness of GAs, and the legible representation and reasoning mechanisms of fuzzy systems. The new online system is addressed as Fuzzy-UCS. Therefore, Fuzzy-UCS aims at providing similarly accurate, but more readable models than those created by XCS and UCS by (i) using a more readable rule representation, since rule variables are represented by linguistic terms, and (ii) evolving smaller rule sets. It is worth noting that Fuzzy-UCS is not the first Michigan-style LCSs that uses a fuzzy representation, but it is the first *learning fuzzy-classifier system* (LFCSS) that works under a supervised learning scheme and builds the knowledge online.

Before proceeding with a detailed description of the learning architecture of Fuzzy-UCS, the next section provides a concise introduction to fuzzy logic, gives a general schema of how GAs have been used in fuzzy systems, and reviews some of the early approaches of LCSs that evolve a fuzzy representation, highlighting the differences with respect to the Fuzzy-UCS architecture.

8.2 Fuzzy Logics in GBML

Fuzzy Logic can be defined as an extension of the traditional logic systems, which has its origins in the ancient Greek philosophy. Fuzzy logics provides a conceptual framework that permits representing the knowledge in environments with uncertainty and imprecision, two characteristics that are really present in nature. In fact, the natural language itself abounds with vague and imprecise concepts. Therefore, fuzzy logic defines a set of procedures or modes of reasoning that are approximate rather than exact. In brief, it extends the concepts of “true” and “false” in bi-valued logic with a third value that indicates that something is “possible” and gives a numeric value between true and false. Recently, the notion of fuzzy logic gained importance due to the pioneer contributions of Zadeh (1965, 1973), who established the foundations of the *fuzzy set theory* and, by extension, of *fuzzy logic*. The following subsections briefly explain the basic mechanisms of fuzzy systems that will be required for the remainder of this chapter. Finally, the different formulas in which GAs and fuzzy logic have been used together in machine learning are briefly introduced, especially focusing on the existing proposals of LFCS.

8.2.1 Fuzzy Logic and Fuzzy Systems

Fuzzy Systems are fundamental methodologies to represent and process linguistic information. Fuzzy systems use fuzzy logic to either represent the knowledge or model the interactions and relationships among the system variables in environments where there is uncertainty and imprecision. Because of this ability to deal with ill-defined data, fuzzy systems have been widely applied to control, classification, and modeling problems (Klir and Yuan, 1995; Pedrycz, 1998) where classical tools were unsuccessful. In what follows, we briefly introduce the basic concepts of the fuzzy set theory and shortly review how they can be incorporated into rule-based systems.

In the fuzzy set theory, each fuzzy subset A of a “crisp”¹ set X is characterized by giving a *degree of membership* to each of its elements $x \in X$. Thence, given a certain observation x and a fuzzy set A , a function addressed as *fuzzy membership function* is defined to return a membership degree of x into A . For example, let us suppose that we define the fuzzy set that represents the term *old*. Then, given a new proposition x (e.g., $x = \text{“John is 54”}$), the fuzzy membership function would provide a degree of membership of x to the set A , which could be *absolutely true* (if John is old), *absolutely false* (if John is not old), or some *intermediate truth degree* (John is old with a degree of 0.75). Several propositions can be combined by connectives, e.g., conjunction, disjunction, and negation. Thence, the fuzzy set theory gives a mathematical interpretation to these connectives so that a new membership degree can be calculated from several propositions joined by connectives.

In particular, the fuzzy set theory has been successfully applied to rule-based systems, resulting in the so-called *fuzzy rule-based systems* (FRBSs). FRBSs consist of fuzzy rules, that is, *if-then* constructions that have the following general form:

$$\mathbf{IF} \ x_1 \text{ is } A_1 \text{ and } \dots \text{ and } x_n \text{ is } A_n \ \mathbf{THEN} \ B, \quad (8.1)$$

where the variables of the antecedent and the consequent contain linguistic labels, that is, the labels are represented by fuzzy sets. These rules are usually called linguistic FRBS or Mamdani

¹Crisp set is used to refer to a set that follows the bi-valued logic.

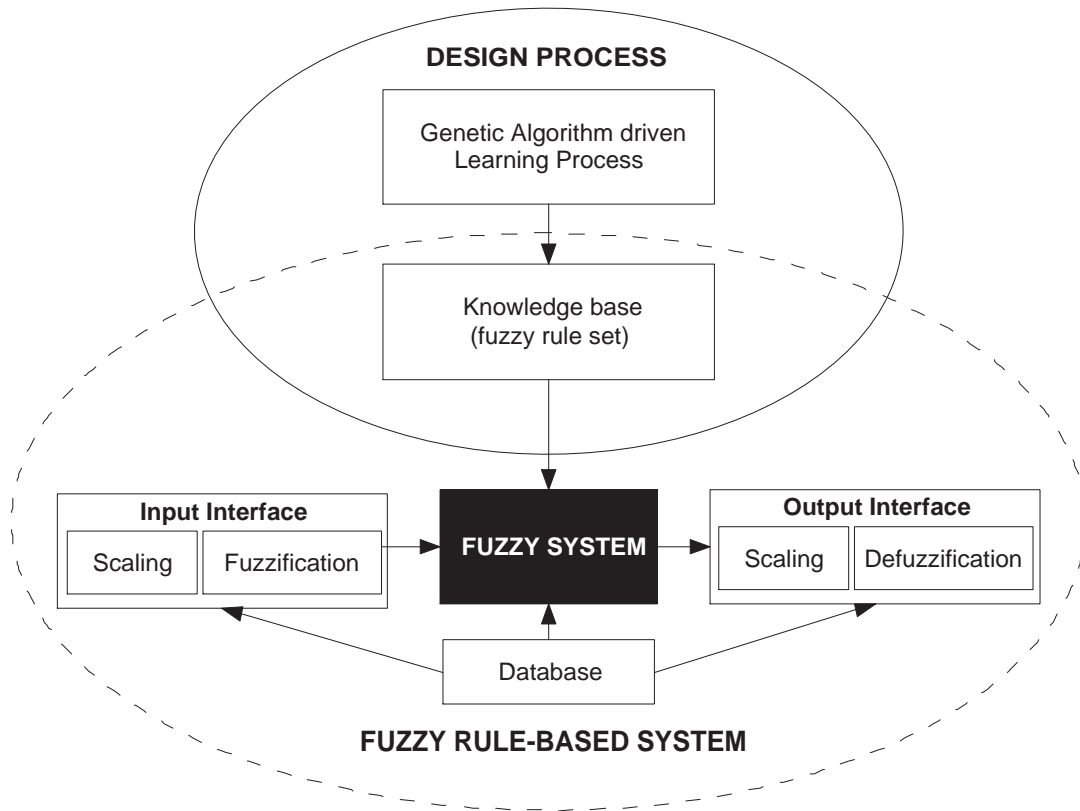


Figure 8.1: Schematic of GFRBS architecture.

FRBSs (Mamdani, 1974) and represent two essential advantages with respect to classical rule-based systems:

1. As variables use fuzzy sets, they naturally can handle uncertainty and vagueness.
2. Rule inference is driven by the reasoning methods defined in the fuzzy set theory, thus, providing inference with a mathematical framework.

Thereupon, the two main tasks in the development of a FRBS are: (1) generate a rule set that represents the problem domain accurately and (2) design an inference mechanism that permit combining the information of all the matching rules. In the next subsection we discuss different methodologies in which GAs are used to assist the building of FRBSs.

8.2.2 Genetic Algorithms in Fuzzy Systems

One of the main drawbacks associated with a FRBS is that the fuzzy rule set has to be defined by human experts, which may be a complex task in some real-world domains. In the last few decades, research has been conducted on designing methods to automatically extract fuzzy rules from a set of data, creating a model that represents the learning domain accurately. Many

authors have regarded the rule extraction as an optimization problem where a set of rules, whose attributes are defined by linguistic terms, need to be found. Then, several search and optimization procedures can be applied to solve this problem.

One of the most prominent proposals is the use of GAs—and evolutionary algorithms in general—to generate this fuzzy rule set. At this point of the thesis, there is not need to further discuss the several reasons that promote the use of GAs, instead of other search mechanisms, to tune different components of FRBS. We have extensively discussed, and empirically shown in the previous sections, (1) the capability of GAs to deal with complex, large search spaces, (2) the success of the use of genetic search in machine learning, and (3) the flexibility of GAs that permit introducing a-priori knowledge. These three reasons are also applicable here, making GAs one of the most appealing methods to be combined with fuzzy systems.

The combination of GA with FRBS has lead to a new branch of GBML that has been addressed as *genetic fuzzy rule-based systems* (GFRBSs) (Cordón et al., 2001a). The basic idea of GFRBS is that GAs are used for either (i) tuning a pre-existing set of fuzzy rules typically with the aim of increasing the global accuracy of the system, as well as the readability of the fuzzy rule set, or (ii) generating the fuzzy rule set from scratch. To further illustrate this process, figure 8.1 provides a general schematic of a GFRBS, which is composed by a FRBS and a genetic procedure that guides the design process. The FRBS is formed by the *knowledge base*, which contains the fuzzy rule set, and an inference engine which, in turn, consists of:

1. An *input interface*, which transforms the input data into fuzzy sets by using a fuzzification process.
2. An *output interface*, which translates the fuzzy rule action to a real action by using a defuzzification method.
3. A *database*, which contains the definition of all the linguistic terms and the membership functions defining the semantics of the linguistic labels.

GAs have assisted the design of the FRBS in several ways. Following the taxonomy provided by Herrera (2008), GFRBS can be grouped in the following three classes.

- GFRBS in which the GA tunes some component of the rule set such as (i) the parameters of the membership functions of the linguistic terms used in the rule set (Casillas et al., 2005; Karr, 1991); (ii) the inference system itself (Alcalá-Fdez et al., 2007; Crockett et al., 2006, 2007); and (iii) the defuzzification function (Kim et al., 2002).
- GFRBS in which the GA is applied to learn some components of the GFRBS. Four approaches can be followed in this case, i.e., algorithms that (i) learn the knowledge base from a set of numerical data (Thrift, 1991); (ii) select the best rules extracted by another algorithm (Ishibuchi et al., 1995, 1997); (iii) learn first the database and then derive the best fuzzy rules for the selected database (Cordón et al., 2001b)—this process can be repeated to get the best combination of both database and knowledge base; or (iv) simultaneously learn both the database and the knowledge base (Homaifar and McCormick, 1995).
- GFRBS in which both the knowledge base and the inference engine parameters are tuned (Marquez et al., 2007).

From these three branches of GFRBS, we are concerned about the GFRBS that learn the rule set from scratch. More specifically, our aim is to design an online Michigan-style LCSs that learns fuzzy classification models with improved legibility. In the following subsection, we review few approaches that fall under this definition, emphasizing the key differences with respect to Fuzzy-UCS.

8.2.3 Related Work on Learning Fuzzy-Classifier Systems

Since Valenzuela-Rendón (1991) presented the first proposal of LFCSs, several authors have adhered to the idea of building machine learning techniques that evolve models online. Most of these systems were applied to solve reinforcement learning and control tasks. As follows, we present the most successful approaches to this topic and finally highlight the differences with respect to the underlying ideas of Fuzzy-UCS.

Valenzuela-Rendón (1991) introduced the first Michigan-style LFCS, which consisted of a fixed-size fuzzy-rule set and a fuzzy message list. The system was applied to solve function approximation tasks. The quality of the fuzzy rules was given according to the accuracy in which the output was estimated. Thus, the initial approach was not a pure reinforcement learning architecture. Later, Nomura et al. (1998) enhanced the system with true reinforcement learning.

Several strength-based Michigan-style LFCS have been proposed since this first description. Parodi and Bonelli (1993) presented an LFCS that automatically learned fuzzy relations, fuzzy membership functions, and fuzzy weights. The fitness (strength) of each rule was used for a double purpose. First, it served to compute the selection and replacement probability of the rule. Second, it permitted stronger rules to participate more soundly in the inference process.

Furuhashi et al. (1994) designed an LFCS that used multiple stimulus-response fuzzy rules operating in tandem. The system was applied to a control task in which a simulated ship had to reach a target without moving the obstacles found on its way. The same problem was addressed by Nakaoka et al. (1994) by using a single rule list. This approach avoided coverage problems in high dimensional spaces by using a dual fitness, one based on environmental payoff, and the other based on the accumulation of the level of activation during simulation.

Velasco (1998) defined a new LFCS architecture specifically designed for fuzzy process control. The system introduced the so-called limbos, i.e., a special workspace where new rules were generated and evaluated before being used in the real process plant. In this way, the system avoided using poorly-evaluated rules in the control system.

Ishibuchi et al. (1999b) designed one of the first proposals of LFCS for pattern classification. They used a fixed-size rule set where *don't care* symbols were defined to permit generalization in the fuzzy rules. A certainty factor, derived from a heuristic procedure prior to fitness evaluation, together with the predicted class formed the consequent of the rule. The rule consequent consisted of the class that the rule advocated and a certainty factor which was derived from a heuristic procedure prior to fitness evaluation. An evolutionary algorithm, which operated only on the rule antecedent, was responsible for creating promising new rules. Although the authors referred to the approach as a Michigan-style LFCS, the rule evaluation process was performed offline; that is, new rules were matched with all the examples of the training data set to compute their fitness. Therefore, this system was not able to deal with data streams. Recently, Ishibuchi

et al. (2005) presented an hybridization of Pittsburgh-style and Michigan-style LFCSs. The system is mainly a Pittsburgh-style LCSs in which some individuals of the population can receive a local search procedure which is inspired by a Michigan-style LCSs. Again, the system evaluates the individuals offline.

Finally, the classic “competition versus cooperation” problem in genetic fuzzy systems was addressed by Bonarini (1996); Bonarini and Trianni (2001). Bonarini proposed a Michigan-style LCS called ELF, which faced the dilemma between the desired cooperation among fuzzy rules that match a given input state and the competition of these rules in the evolutionary algorithm. In ELF, the rule set was divided into several sub-populations, each one with the same antecedent. Then, the rules of different sub-populations cooperated to produce the control action, whilst the members of each subpopulation competed with each other. Moreover, ELF controlled the instability of general rules that participated in different sub-populations by providing each rule a reinforcement normalized on the difference between the maximum and the minimum reinforcement obtained by the subpopulation to which the rule belongs. In this way, ELF overcame some of the problems of strength-based LCSs. ELF was applied to several reinforcement learning problems, such as the coordination of autonomous agents.

All the LFCS described through this section are strength-based systems. In reinforcement learning, the first successful accuracy-based fuzzy rule-based system with generalization capability was proposed by (Casillas et al., 2007). To the best of our knowledge, no accuracy-based LFCS specifically designed for classification has been proposed until now. Therefore, Fuzzy-UCS is the first approach in this field. The system takes an accuracy-based approach to benefit from the advantages that these types of systems have introduced to LCSs, which are summarized as follows.

- Accuracy-based LCSs can distinguish over-general from accurate rules (Bull and Hurst, 2002).
- There are theoretical analyses that support the theory that, for binary representation, LCSs such as XCS will evolve a rule set with maximally-general and highly accurate rules if certain conditions are met (Butz et al., 2004b; Butz, 2006; Butz et al., 2007). Besides, there are further models, as those provided in chapters 5 and 6 that explain different facets of how these systems work. Although similar analyses in the continuous space are scarce, the positive conclusions extracted for the binary representation promote the use of Michigan-style LCSs.

The next subsection explains Fuzzy-UCS in detail.

8.3 Description of Fuzzy-UCS

The purpose of this section is to describe the design and implementation details of Fuzzy-UCS so that it can be used as an implementation guide. Figure 8.2 schematically illustrates the process organization of the system. The system works in two different modes: *exploration* or training and *exploitation* or test. In the exploration mode, Fuzzy-UCS seeks to evolve a maximally general rule set that minimizes the training error. In the exploitation mode, Fuzzy-UCS uses the evolved knowledge to infer the class of unlabeled examples. As proceeds, we first present the

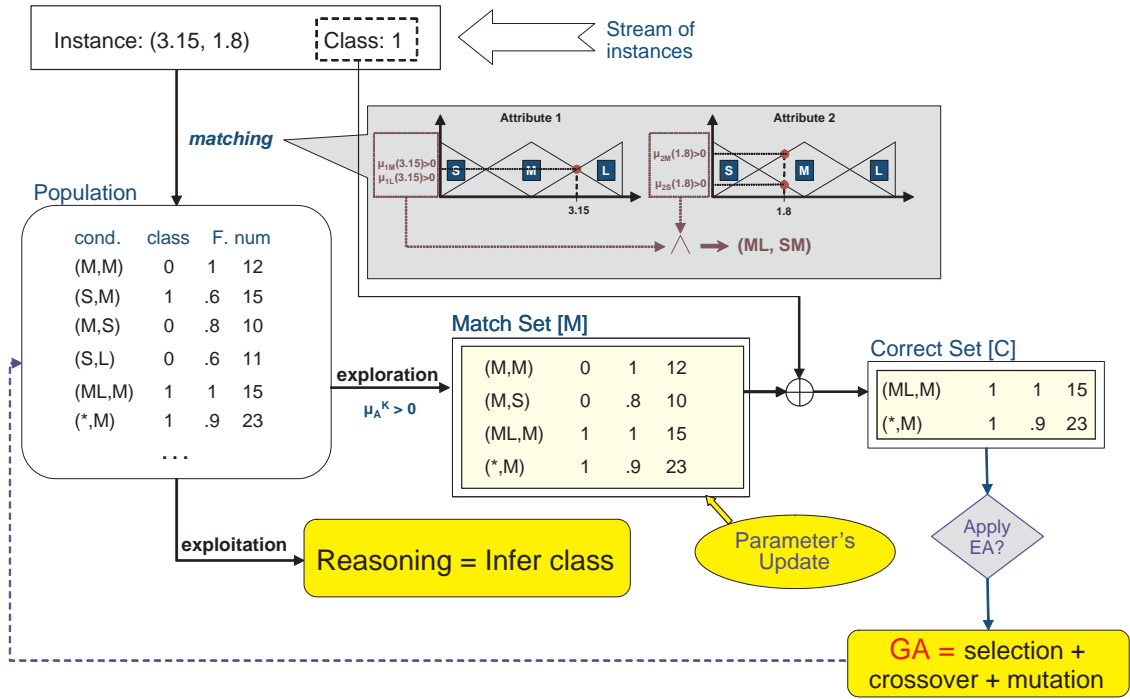


Figure 8.2: Schematic illustration of Fuzzy-UCS. The run cycle depends on whether the system is under exploration (training) or exploitation (test).

fuzzy knowledge representation used by Fuzzy-UCS and then introduce the learning interaction, the rule evaluation system, and the rule discovery component used in the training mode. Finally, we also introduce the reasoning mechanisms designed to infer the class of new unlabeled examples in test mode.

8.3.1 Knowledge Representation

We first introduce the fuzzy-rule-based representation of Fuzzy-UCS, explain how the matching mechanism works, and present the most relevant parameters that are associated with each classifier. Fuzzy-UCS evolves a *population* [P] of classifiers which jointly represent the solution to a problem. Each classifier consists of a rule whose condition is in *conjunctive normal form* and a set of parameters. The fuzzy rule follows the structure

$$\text{IF } x_1 \text{ is } \widetilde{A}_1^k \text{ and } \dots \text{ and } x_n \text{ is } \widetilde{A}_n^k \text{ THEN } c^k \text{ WITH } w^k, \quad (8.2)$$

where each input variable x_i is represented by a disjunction of *linguistic terms* or *labels* $\widetilde{A}_i^k = \{ A_{i1} \vee \dots \vee A_{in_i} \}$. In our experiments, all input variables share the same semantics, which are defined by means of triangular-shaped fuzzy membership functions. (see the examples of these semantics with different number of linguistic terms in figure 8.3). Note that this representation

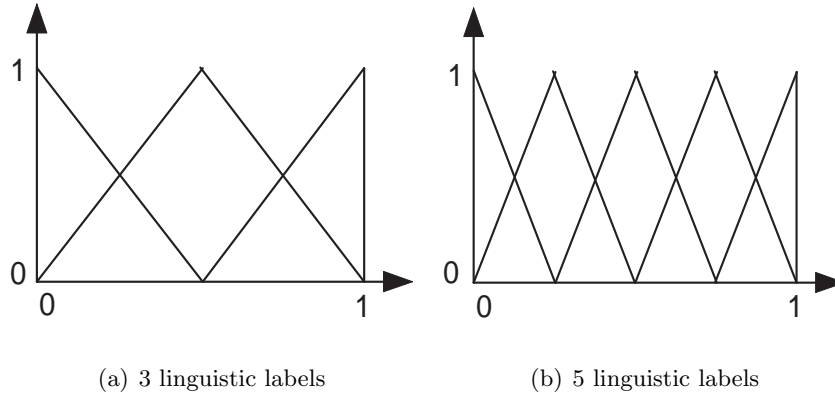


Figure 8.3: Representation of a fuzzy partition for a variable with (a) three and (b) five triangular-shaped membership functions.

intrinsically permits generalization since each variable can take an arbitrary number of linguistic terms. The consequent of the rule indicates the class c^k which the rule itself predicts. w^k is a weight ($0 \leq w^k \leq 1$) that denotes the soundness with which the rule predicts the class c^k . These types of rules with a weight in the consequent are known as fuzzy rules of type II (Cordón et al., 2001a).

The *matching degree* $\mu_{A^k}(e)$ of an example e with a classifier k is computed as follows. For each variable x_i , we compute the membership degree for each of its linguistic terms, and aggregate them by means of a T-conorm (disjunction). We enable the system to deal with missing values by considering that $\mu_{A^k}(e) = 1$ if the value e_i for the input variable x_i is not known. Then, the matching degree of the rule is determined by the T-norm (conjunction) of the matching degree of all the input variables. In our implementation, we used a *bounded sum* ($\min\{1, a + b\}$) as T-conorm and the *product* ($a \cdot b$) as T-norm. Note that, if the fuzzy partition guarantees that the addition of all membership degrees is greater than or equal to 1—the membership functions used in our experiments satisfy this condition—the selected T-norm and T-conorm allow for a maximum generalization. Therefore, an input variable x_i consisting of two consecutive linguistic terms will result in a matching degree of $\mu_{x_i}(e) = 1$ if the matching of e_i with both linguistic terms is greater than zero; thus, this choice supports the absence of the variable x_i .

Each classifier has four main parameters: 1) the fitness F , which estimates the accuracy of the rule; 2) the correct set size cs , which averages the sizes of the correct sets in which the classifier has participated (see section 8.3.2); 3) the experience exp , which computes the contributions of the rule to classify the input instances; and 4) the numerosity num , which counts the number of copies of the rule in the population. Some of the parameters such as the fitness and the experience have been “fuzzified” with respect the corresponding parameters in XCS and UCS.

To completely understand the new fuzzy rule representation, in the following subsections we detail how the different components of Fuzzy-UCS interact to evaluate the existing classifiers and create new promising rules.

8.3.2 Learning Interaction

Fuzzy-UCS inherits the process organization from UCS (see chapter 3), but it is adapted to deal with fuzzy rules. For this purpose, three main differences with respect to UCS need to be considered: the *matching calculation*, the *rule structure*, and the *inference methodology*.

1. *Matching calculation.* In UCS, the attributes are represented by intervals $[l_i, u_i]$, and thus, a rule matches an input example if $\forall e_i : l_i \leq e_i \leq u_i$. Therefore, the matching function returns a binary output indicating whether the classifier matches the example e or not. In Fuzzy-UCS, a rule k matches the input example with a matching degree $\mu_{A^k}(e)$, where $0 \leq \mu_{A^k}(e) \leq 1$. High values of $\mu_{A^k}(e)$ indicate that the prediction of rule k is fairly accurate.
2. *Rule structure.* In UCS, a rule predicts a single class with a certain fitness or quality. Consequently, the population may contain two rules with the same antecedent advocating different classes. To avoid this situation in Fuzzy-UCS, rules internally maintain a weight for each class that indicates the soundness in which this class is predicted. These weights are updated by the online learning architecture and are only used to determine the class that the rule predicts; that is, the class advocated by the rule is the class with the maximum weight. Therefore, the class predicted by the rule can change as the rule is evaluated online.
3. *Inference methodology.* In UCS, all the classifiers in $[M]$ emit a fitness-weighted vote for the class they advocate, and the most voted class is chosen as the predicted output. In Fuzzy-UCS, different fuzzy-logic inference methods can be used to infer the class from the final fuzzy rule set (Cordón et al., 1999). Section 8.3.5 presents the three types of inference used by the system.

The learning organization of Fuzzy-UCS was redesigned considering these differences. As follows, the learning mechanism used during training is carefully reviewed, focusing on the main differences with respect to UCS. The reader is referred to section 8.3.5 for the details on the reasoning methods used to classify new instances in exploitation mode.

At each learning iteration, Fuzzy-UCS receives a new input example e and its class c , and the system builds the match set $[M]$, which contains all the classifiers in $[P]$ that have a matching degree $\mu_{A^k}(e)$ greater than zero.² Then, the system creates the correct set $[C]$ with all the rules in $[M]$ that advocate the class of the input example. If none of the classifiers in $[C]$ match e with the *maximum matching degree*, the covering operator is triggered, which creates the classifier that maximally matches the input example. That is, for each attribute of the condition, we aggregate the linguistic term A_{ij} that maximizes the matching with the input value e_i . If e_i is not known, we randomly select a linguistic term and aggregate it to the attribute. Moreover, we introduce generalization by permitting the addition of other linguistic terms with probability $P_{\#}$. The initial values of the new classifiers are initialized according to the information provided by the current examples. Specifically, the fitness, the numerosity, and the experience are set to

²We do not require that rules have a matching degree greater than a certain threshold to be in $[M]$, as sometimes done in regression (Casillas et al., 2007). In regression, the output is formed by means of aggregating rules with different actions. Thus, a minimum matching degree with the input may be required to participate in this process. However, in Fuzzy-UCS, the rules in $[C]$ advocate the same class. In this way, Fuzzy-UCS avoids aggregating rules of different classes in the learning process, and so, a matching threshold appears to be less necessary.

1. The fitness of a new rule is set to 1 to give it opportunities to take over. Nonetheless, two important aspects should be noted. First, as the new classifiers participate in new match sets, their fitness and other parameters are quickly updated to their average values, and so, the initial value is not crucial. Second, as specified in the following sections, the system prevents young classifiers from having a strong presence in the genetic selection, and protects them from an early deletion. At the end of the covering process, the new classifier is inserted in the population, deleting another one if there is not room for it.

Next, in exploration mode, the classifiers in $[M]$ that advocate the class c form the correct set $[C]$. As in UCS, the correct set works as a niche where the genetic algorithm is applied. Besides, after each learning iteration, the parameters of all the classifiers in $[M]$ are updated. The following two subsections explicate these two procedures in detail.

8.3.3 Classifiers Update

At the end of each learning iteration, Fuzzy-UCS updates the parameters of the rules in $[M]$. As explained above, most of the parameters were redefined with respect to those of UCS to be able to deal with fuzzy rules—i.e, the parameters were “fuzzified”. As proceeds, the equations used to update the parameters are provided.

First, the experience of the rule is incremented according to the current matching degree:

$$exp_{t+1}^k = exp_t^k + \mu_{A^k}(e). \quad (8.3)$$

Thence, in Fuzzy-UCS, the experience parameter accounts for the contributions of the classifier in matching instances; that is, classifiers that match with high degree several instances will have high experience. Next, the fitness is updated. For this purpose, each classifier internally maintains a vector of classes $\{c_1, \dots, c_m\}$, each of them with an associated weight $\{v_1^k, \dots, v_m^k\}$. Each weight v_j^k indicates the soundness with which rule k predicts class j for an example that fully matches this rule. These weights are incrementally updated during learning as explained as follows. The class c^k advocated by the rule is the class with the maximum weight v_j^k . Thus, given that the weights may change due to successive updates, the class that a rule predicts may also vary.

To update the weights, we first compute the sum of correct matchings cm^k for each class j :

$$cm_{j_{t+1}}^k = cm_{j_t}^k + m(k, j), \quad (8.4)$$

where

$$m(k, j) = \begin{cases} \mu_{A^k}(e) & \text{if } j=c; \\ 0 & \text{otherwise.} \end{cases} \quad (8.5)$$

Then, $cm_{j_{t+1}}^k$ is used to compute the weights $v_{j_{t+1}}^k$:

$$\forall j : v_{j_{t+1}}^k = \frac{cm_{j_{t+1}}^k}{exp_{t+1}^k}. \quad (8.6)$$

For example, if a rule k only matches examples of class j , the weight v_j^k will be 1 and the remaining weights 0. Rules that match instances of both classes will have weights ranging from 0 to 1. Note that the sum of all the weights is 1.

The fitness is then computed from the weights with the aim of favoring classifiers that match examples of a single class. To carry this out, we use the following formula (Ishibuchi and Yamamoto, 2005):

$$F_{t+1}^k = v_{max_{t+1}}^k - \sum_{j|j \neq max} v_{j_{t+1}}^k, \quad (8.7)$$

where we subtract the values of the other weights from the weight with maximum value v_{max}^k . The fitness F^k is the value used as the weight w^k of the rule (see Equation 8.2). Note that this formula can result in classifiers with zero or negative fitness (for example, if the number of classes is greater than 2 and the class weights are equal). Lastly, the correct set size of all the classifiers in [C] is calculated as the arithmetic average of the sizes of all the correct sets in which the classifier has participated.

Finally, the rule k predicts the class c with the highest weight associated v_c^k . Thus, the class predicted is not fixed when the rule is created, and can change as the parameters of the rule are updated (especially during the first parameters updates). Once the parameters of all the classifiers in [M] have been updated, the GA can be applied to the current niche. In this case, the GA follows the process explained in the next subsection.

8.3.4 Classifiers Discovery

Fuzzy-UCS uses a steady-state niched *genetic algorithm* (GA) (Goldberg, 1989a) to discover new promising rules. The GA is applied to the classifiers that belong to [C]. Thus, the niching is intrinsically provided since the GA is applied to rules that match the same input with a degree greater than zero and advocate the same class.

The GA is triggered when the average time from its last application upon the classifiers in [C] exceeds the threshold θ_{GA} . It selects two parents p_1 and p_2 from [C] using proportionate selection (Goldberg, 1989a), where the probability of selecting a classifier k is

$$p_{sel}^k = \frac{(F^k)^\nu \cdot \mu_{A^k}(e)}{\sum_{i \in [C] | F^i \geq 0} (F^i)^\nu \cdot \mu_{A^k}(e)}, \quad (8.8)$$

where $\nu > 0$ is a constant that fixes the pressure toward maximally accurate rules (in our experiments, we set $\nu=10$). Therefore, the probability of a classifier being selected depends on the product of its fitness and the matching degree with the input instance. Rules with negative fitness are not considered for selection. The two parents are copied into offspring ch_1 and ch_2 , which undergo crossover and mutation with probabilities χ and μ respectively. The crossover operator crosses the antecedents of the rules by two points. The mutation operator checks whether each variable has to be mutated with probability μ . If so, three types of mutation can be applied: *expansion*, *contraction*, or *shift*. Expansion chooses a linguistic term not represented in the corresponding variable and adds it to this variable; thus, it can be applied only to variables that do not have all the linguistic terms. Contraction selects a linguistic term represented in the variable and removes it; so, it can be applied only to variables that have more than one linguistic term. By doing so, we avoid generating rules that do not match any example. Shift changes a linguistic term for its immediate inferior or superior.

The new offspring are introduced into the population. First, each classifier is checked for subsumption (Wilson, 1998) with their parents. Subsumption is a mechanism that prevents the creation of classifiers with specific conditions if there are more general and accurate classifiers in the population that cover the same region of the feature space. So, subsumption pressures toward maximally general and accurate classifiers. The process works as follows. If any parent's condition subsumes the condition of the offspring (i.e., the parent has, at least, the same linguistic terms per variable than the child), and this parent is highly accurate ($F^k > F_0^k$) and sufficiently experienced ($exp^k > \theta_{sub}$), the offspring is not inserted into the population and the numerosity of the parent is increased by one. Otherwise, we check [C] for the most general rule that can subsume the offspring. If no subsumer can be found, the classifier is inserted into the population.

If the population is full, excess classifiers are deleted from [P] with probability proportional to the correct set size estimate cs , following a method adapted from (Kovacs, 1999). Moreover, if the classifier is sufficiently experienced ($exp^k > \theta_{del}$) and the power of its fitness $(F^k)^\nu$ is significantly lower than the average fitness of the classifiers in [P] ($(F^k)^\nu < \delta F_{[P]}$, where $F_{[P]} = \frac{1}{N} \sum_{i \in [P]} (F^i)^\nu$), its deletion probability is further increased. That is, each classifier has a deletion probability p_k of:

$$p_k = \frac{d_k}{\sum_{\forall j \in [P]} d_j}, \quad (8.9)$$

where

$$d_k = \begin{cases} \frac{cs \cdot num \cdot F_{[P]}}{(F^k)^\nu} & \text{if } exp^k > \theta_{del} \text{ and } (F^k)^\nu < \delta F_{[P]}; \\ cs \cdot num & \text{otherwise.} \end{cases} \quad (8.10)$$

Thus, the deletion algorithm balances the classifier's allocation in the different correct sets by pushing toward deletion of rules belonging to large correct sets. At the same time, it favors the search toward highly fit classifiers, since the deletion probability of rules whose fitness is much smaller than the average fitness is increased.

8.3.5 Fuzzy-UCS in Test Mode

The aim of Fuzzy-UCS is to evolve a minimum set of maximally accurate rules that cooperate to cover all the input space. To achieve high classification accuracy, not only needs the system to create a population of highly accurate classifiers during learning, but it also has to define effective reasoning methods that use the information of the rule set to infer the class of new input examples. As these reasoning methodologies may not use all the rules in the inference process, rule set reduction techniques similar to those used in (Orriols-Puig and Bernadó-Mansilla, 2004) can be applied to remove the rules that are not considered for the reasoning technique. Herein, we discuss two different inference schemes. Furthermore, we present a reduction method for each one of these inference methods that permits a reduction the number of rules in the final rule set without decreasing training accuracy. Finally, we also introduce a third rule set reduction mechanism which allows for higher reductions, but does not guarantee that the reduced rule set results in the same training performance as the original one.

Class Inference

Once Fuzzy-UCS has evolved a population of highly general and accurate rules, this population is used to infer the class of new examples. Given a new unlabeled instance e , several rules predicting different classes can match (with different degrees) this instance. Thus, the knowledge contained in the set of matching classifiers has to be combined to decide the most likely output. For this purpose, several reasoning methodologies have been analyzed in the realm of fuzzy-rule based systems (Cordón et al., 1999; Ishibuchi et al., 1999a). Here, we adapt two inference approaches to Fuzzy-UCS. In both cases, only experimented rules ($exp^k > \theta_{exploit}$) are considered in the inference, where $\theta_{exploit}$ is a user-set parameter that indicates the minimum experience that a rule must have to participate in the inference process.

Weighted average inference. In this approach, all the experienced rules vote to infer the output. Each rule k emits a vote v_k for class j it advocates, where

$$v_k = F^k \cdot \mu_{A^k}(e). \quad (8.11)$$

The votes for each class j are added:

$$\forall j : vote_j = \sum_{k|c^k=j}^N v_k, \quad (8.12)$$

and the most-voted class is returned as the output.

Action winner inference. This approach selects the rule k that maximizes $\mu_{A^k}(e) \cdot F^k$, and chooses the class of the rule as output (Ishibuchi et al., 1999b). Thus, the knowledge of overlapping rules is not considered in this inference scheme.

Rule Set Reduction

At the end of the learning process, the population is reduced to obtain a minimum set of rules. We designed three types of reduction, which use one of the inference schemes presented above.

Reduction based on weighted average. Under the weighted average scheme, we reduce the final population by removing all the rules that a) are not experienced enough ($exp \leq \theta_{exploit}$) or b) have zero or negative fitness.

Reduction based on action winner. If action winner inference is used, only rules that maximize the prediction vote for a training example are necessary. Thus, after training, this reduction scheme infers the output for each training example. The rule that maximizes the vote v_j for each example is copied to the final population.

Reduction based on the fittest rules. This reduction tries to minimize the rule set size by selecting the most numerous and accurate rules for the final population. The methodology is a hybrid of the previous approaches. The reduction process is analogous to the reduction based on action winner, but now, the rule k that maximizes $F^k \cdot \mu_{A^k}(e) \cdot num^k$ for each input example is copied to the final population. By including the numerosity in the vote, we favor the most numerous and accurate rules. As this reduction may copy overlapping rules into the final population, weighted average is used to infer the class of a new example.

Overall, Fuzzy-UCS is an LFCS that evolves a set of linguistic fuzzy rules online and uses three different inference/reduction mechanisms to predict the class of test instances. Since Fuzzy-UCS is a brand new system that mixes different ideas coming from the LCSs, the GAs, and the fuzzy logic realms, the following sections take an empirical approach to analyze the behavior of the system. First, we study the influence of the different configuration parameters that have appeared along the description. Note that the majority of these parameters—or similar ones—are also present in XCS and UCS. Therefore, we hypothesize that they have a similar impact in the three systems; in any case, the next section empirically examines the effect of modifying the configuration parameters. This study results in a default configuration for Fuzzy-UCS, which is used in the remainder of this chapter. Subsequently, we analyze the differences between the three inference/reduction schemes of Fuzzy-UCS.

8.4 Sensitivity of Fuzzy-UCS to Configuration Parameters

In common with many competitive Michigan-style LCSs, Fuzzy-UCS has several configuration parameters, which enable to adjust the behavior of the system to evolve models of maximal quality for particular problems. At first glance, choosing a correct configuration may seem a crucial task only suitable to expert users. Nonetheless, several analyses identified the robustness of Michigan-style LCSs to the majority of configuration parameters. Actually, most of the applications of Michigan-style LCSs used the same default parameters to solve pattern recognition problems (Bernadó-Mansilla et al., 2002; Bernadó-Mansilla and Garrell, 2003; Butz, 2006; Orriols-Puig and Bernadó-Mansilla, 2008b; Dixon et al., 2002, 2004; Fu et al., 2001; Wilson, 2000). We consider that this robustness is also present in Fuzzy-UCS. Thence, this section empirically illustrates the behavior of Fuzzy-UCS with different configurations and relate this analysis to theoretical and empirical studies of the sensitivity of LCSs—particularly XCS and UCS—to configuration parameters. For the sake of compactness, here we also present the summary of the results and the statistical analysis that leads us to the most important conclusions. The current analysis is further detailed in appendix C.

Theoretical and empirical analyses of the sensitivity of LCSs³ to configuration parameters detected four crucial parameters: (1) population initialization (Butz et al., 2001), (2) fitness pressure (Kharbat et al., 2005; Brown et al., 2007), (3) GA application rate (Butz et al., 2007), and (4) deletion pressure (Butz et al., 2007). The influence of the other parameters is less important, and most of LCSs works use a standard configuration for them.

Herein, we empirically study the sensitivity of Fuzzy-UCS to the configuration parameters. For this purpose, we analyzed the accuracy and size of the models evolved by Fuzzy-UCS related to the changes of four parameters or groups of parameters: (1) rules generalization in initialization, i.e., $P_{\#}$; (2) fitness pressure, i.e., ν ; (3) setting of the genetic algorithm, i.e., θ_{GA} , θ_{del} , and θ_{sub} ; and (4) deletion pressure, i.e., δ . We compared different configuration settings to the following default configuration (Cp), which sets the configuration parameters to standard values in literature: $N=6\,400$, $F_0 = 0.99$, $\nu = 10$, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\} = 50$, $\theta_{exploit} = 10$, $\chi = 0.8$, $\mu = 0.6$, $\delta=0.1$, and $P_{\#} = 0.6$. Note that this configuration will be used in the following experiments of this chapter.

³These analyses refer to XCS and UCS, but could be easily extended to other Michigan-style LCSs.

8.4. SENSITIVITY OF FUZZY-UCS TO CONFIGURATION PARAMETERS

Table 8.1: Properties of the data sets. The columns describe: the identifier of the data set (Id.), the name of the data set (dataset), the number of instances (#Ins), the total number of features (#Fea), the number of continuous features (#Cnt), the number of nominal features (#No), the number of classes (#C), the proportion of instances of the minority class (%Min), the proportion of instances of the majority class (%Maj), the proportion of instances with missing values (%MI), and the proportion of features with missing values (%MA).

Id.	dataset	#Ins	#Fea	#Cnt	#No	#C	%Min	%Maj	%MI	%MA
<i>ann</i>	Annealing	898	38	6	32	5	0.9	76.2	0	0
<i>aut</i>	Automobile	205	25	15	10	6	1.5	32.7	22.4	28
<i>bal</i>	Balance	625	4	4	0	3	7.8	46.1	0	0
<i>bpa</i>	Bupa	345	6	6	0	2	42	58	0	0
<i>cmc</i>	Contrac. method choice	1473	9	2	7	3	22.6	42.7	0	0
<i>col</i>	Horse colic	368	22	7	15	2	37	63	98.1	95.5
<i>gls</i>	Glass	214	9	9	0	6	4.2	35.5	0	0
<i>h-c</i>	Heart-c	303	13	6	7	2	45.5	54.5	2.3	15.4
<i>h-s</i>	Heart-s	270	13	13	0	2	44.4	56.6	0	0
<i>irs</i>	Iris	150	4	4	0	3	33.3	33.3	0	0
<i>pim</i>	Pima	768	8	8	0	2	34.9	65.1	0	0
<i>son</i>	Sonar	208	60	60	0	2	46.67	53.33	0	0
<i>tao</i>	Tao	1888	2	2	0	2	50	50	0	0
<i>thy</i>	Thyroid	215	5	5	0	3	14	60	0	0
<i>veh</i>	Vehicle	846	18	18	0	4	23.5	25.8	0	0
<i>wbcd</i>	Wisc. breast-cancer	699	9	9	0	2	34.5	65.5	2.3	11.1
<i>wdbc</i>	Wisc. diag. breast-cancer	569	30	30	0	2	37.3	62.7	0	0
<i>wne</i>	Wine	178	13	13	0	3	27	39.9	0	0
<i>wdbc</i>	Wisc. prog. breast-cancer	198	33	33	0	2	23.7	76.3	2	3
<i>zoo</i>	Zoo	101	17	1	16	7	4	40.6	0	0

Table 8.2: Configurations used to test the sensitivity of Fuzzy-UCS to configuration parameters.

	Cp	$N=6\,400$, $F_0 = 0.99$, $\nu = 10$, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\} = 50$, $\theta_{exploit} = 10$, $\chi = 0.8$, $\mu = 0.1$, $\delta=0.1$, and $P_{\#} = 0.2$
$P_{\#}$	C1	$P_{\#} = 0.2$
	C2	$P_{\#} = 0.4$
ν	C3	$\nu = 1$
	C4	$\nu = 5$
$\theta_{GA}, \theta_{del}, \theta_{sub}$	C5	$\theta_{GA} = \theta_{del} = \theta_{sub} = 100$ and $numIter = 100\,000$
	C6	$\theta_{GA} = \theta_{del} = \theta_{sub} = 200$ and $numIter = 100\,000$
	C7	$\theta_{GA} = \theta_{del} = \theta_{sub} = 100$ and $numIter = 200\,000$
	C8	$\theta_{GA} = \theta_{del} = \theta_{sub} = 200$ and $numIter = 400\,000$
δ	C9	$\delta = 1$

Table 8.3: Comparison of the sensitivity of Fuzzy-UCS to configuration parameters. Each cell shows the average rank of each configuration for a given inference scheme. The best ranked method is in bold. The \ominus symbol indicates that the corresponding method significantly degrades the results obtained with the best ranked method.

		Performance			Rule set size		
		wavg	awin	nfit	wavg	awin	nfit
$P_{\#}$	Cp	1.83	1.83	1.83	1.67	1.50	1.75
	C1	2.42 \ominus	2.50 \ominus	2.25 \ominus	1.75	2.92 \ominus	3.00 \ominus
	C2	1.75	1.67	1.92	2.58 \ominus	1.58	1.25
ν	Cp	1.25	1.42	1.42	1.08	2.33 \ominus	2.67 \ominus
	C3	2.83 \ominus	2.75 \ominus	2.83 \ominus	3.00 \ominus	1.25	1.17
	C4	1.92	1.83	1.75	1.92 \ominus	2.42 \ominus	2.17 \ominus
$\theta_{GA}, \theta_{del} \& \theta_{sub}$	Cp	1.92	2.13	2.13	3.42 \ominus	3.17	3.17
	C5	4.00 \ominus	3.42	3.58 \ominus	3.42 \ominus	3.50	2.92
	C6	4.33 \ominus	4.63 \ominus	4.17 \ominus	1.75	2.83	2.58
	C7	2.33	2.25	2.29	3.42 \ominus	3.33	3.17
	C8	2.42	2.58	2.83	3.00	2.17	3.17
δ	Cp	1.25	1.54	1.50	1.33	1.75	1.75
	C9	1.75	1.46	1.50	1.67	1.25	1.25

We ran the experiments on a collection of real-world classification problems, whose characteristics are described in table 8.1. Due to the large number of tested configurations, we used a reduced collection of data sets to perform these experiments, that is: *bal*, *bpa*, *gls*, *h-s*, *irs*, *pim*, *tao*, *thy*, *veh*, *wbcd*, *wdbc*, and *wne*.

Table 8.2 summarizes the different configurations and the changes that they introduced with respect to the default configuration in each of the four experiments. Table 8.3 provides the average rank of the model’s accuracy and size for each configuration and inference scheme. We divided the configuration settings into four groups, and each group was compared to the default configuration. The best ranked configurations for each comparison are marked in bold. The \ominus symbol indicates that the corresponding configuration significantly degraded the results obtained with the best configuration according to a Bonferroni-Dunn test at $\alpha = 0.1$ (Dunn, 1961).

The results show that the generalization in the initial population is essential to the success of Fuzzy-UCS, supporting the theoretical analyses in the literature (Butz et al., 2001). For all the inference schemes, configurations *Cp* and *C2* (i.e., $P_{\#} = \{0.6, 0.4\}$) were statistically equivalent, on average, and significantly better than *C1* (i.e., $P_{\#} = 0.2$) in terms of accuracy. In terms of model size, the following significant differences were found: (i) for weighted average inference, *Cp* and *C1* evolved the smallest rule sets; (ii) for action winner and fittest rules inference, *C1* created significantly larger rule sets than *Cp* and *C2*. The last point can be easily explained as follows. As *C1* used a low value of $P_{\#}$, the final populations contained more specific classifiers than populations created with *Cp* and *C2*. Action winner and fittest rules schemes only kept the classifiers that maximized the product of fitness and matching degree with a training instance

in the final populations. As classifiers were more specific, a larger number of them were placed in the final population. On the other hand, with weighted average, the biggest population sizes were obtained with $C2$. This could be due to the existence of slightly general classifiers that were all maintained in the final population.

The second comparison shows the negative influence of having low fitness pressure. In terms of accuracy, better results were obtained as the fitness pressure increased (i.e., ν took higher values). Population sizes varied with the fitness pressure depending on the inference scheme. For weighted average inference, Cp led to the significantly smaller rule sets. This is because the fitness pressure drove toward a highly general and accurate set of rules. For the other two inference schemes, configuration $C1$ resulted in the significantly smaller rule sets. That is, as the fitness pressure was low, populations were full of over-general rules, which were kept in the final populations in detriment of fitter and more specific classifiers.

The third comparison shows the influence of the parameters related to the genetic algorithm, i.e., θ_{GA} , θ_{del} , and θ_{sub} . Initial intuition indicates that, if all niches receive the same number of genetic opportunities, the quality of the final models should remain the same. To test this, configurations $C7$ and $C8$ set $\theta_{GA} = \theta_{del} = \theta_{sub} = \{100, 200\}$ and increased $numIter = \{200\,000, 400\,000\}$ respectively. In this way, all niches received approximately the same number of genetic events. On the other hand, configurations $C5$ and $C6$ fixed $\theta_{GA} = \theta_{del} = \theta_{sub} = \{100, 200\}$ but maintained the same number of iterations as Cp . So, we expected that the quality of the models evolved by $C5$ and $C6$ was significantly lower than the quality of the models created by the three other configurations. This hypothesis was clearly supported by the experimental analysis, which showed that Cp , $C7$, and $C8$ resulted in the most accurate models. Moreover, significant differences on the population sizes were only found for the weighted average inference. The multiple-comparison test detected that the smaller models were created with configurations $C6$ and $C8$, the two configurations in which the period of application of the GA was higher.

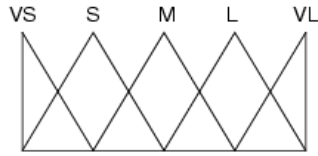
Finally, the fourth comparison highlights the robustness of Fuzzy-UCS to the deletion pressure toward unfit classifiers, that is, the parameter δ . The pairwise analysis indicated that the hypothesis that configurations Cp and $C9$ are equivalent could not be rejected, according to a Wilcoxon signed-ranks test at $\alpha = 0.05$.

The study conducted in this section empirically showed that there are two crucial parameters to guarantee the success of Fuzzy-UCS: generalization in initialization $P_{\#}$ and fitness pressure ν . On the other hand, changing the setting of the other parameters had little effect on Fuzzy-UCS behavior. We acknowledge that better results could be individually obtained if we tuned Fuzzy-UCS for each particular problem. Nonetheless, as we are interested in robust systems that perform well on average, we use the default configuration for all the experiments in the following sections.

8.5 Knowledge Representation and Decision Boundaries

So far, we have described the Fuzzy-UCS classifier system with a *descriptive* or *linguistic* representation of fuzzy rules, which is referred to as linguistic Fuzzy-UCS in the remainder of this chapter, and have analyzed its robustness with respect to its configuration parameters. Linguistic rules are highly interpretable since they share common semantics; however, as this






Fuzzy partition



Rule set

R_1 : IF X is {VS,S} THEN Y is C_1
 R_2 : IF X is {M,L} THEN Y is C_2
 R_3 : IF X is VL THEN Y is C_3

(a) Linguistic fuzzy rule set

R_1 : IF X is  THEN Y is C_1
 R_2 : IF X is  THEN Y is C_1
 R_3 : IF X is  THEN Y is C_2
 R_4 : IF X is  THEN Y is C_3
 R_5 : IF X is  THEN Y is C_3

(b) Approximate fuzzy rule set

Figure 8.4: Graphical comparison between (a) linguistic and (b) approximate fuzzy rule sets.

representation implies the discretization of the feature space, a single rule may not have the required granularity to define the class boundary of a given domain accurately. Thus, Fuzzy-UCS would evolve a set of overlapping fuzzy-rules around the decision boundaries which match examples of different classes, and the output would depend on how the reasoning mechanism combines the knowledge of all these overlapping rules. Fuzzy-UCS includes three inference and reduction schemes which lead to a trade-off between the amount of information used for the inference process (i.e., the precision of the prediction) and the size of the rule set. Consequently, not only the linguistic representation but also the chosen inference and reduction schemes may impose a maximum limit on the accuracy rate that the system can reach.

To achieve better accuracy rates, several authors introduced the so-called *approximate* rule representation (also known as non-grid-oriented fuzzy systems, prototype-based representation, or fuzzy graphs) (Alcalá et al., 2001; Bardossy and Duckstein, 1995; Carse et al., 1996; Cordon and Herrera, 1997). This representation allows the variables of fuzzy rules to define their own fuzzy sets instead of representing linguistic variables. In this way, approximate fuzzy rules are semantic free; that is to say, the fuzzy sets of any variable of each rule can be independently tuned. However, this also results in a degradation of the interpretability of the final rule set, since the fuzzy variables no longer share a common linguistic interpretation. Figure 8.4 illustrates the two representations.

This section studies the interpretability-performance trade-off in Fuzzy-UCS and analyzes if the flexibility provided by the approximate representation allows the system to achieve higher levels of performance. For this purpose, we include the approximate representation in Fuzzy-UCS and adapt several mechanisms to deal with approximate rules. This alternative algorithm is described in the next section. This modification of Fuzzy-UCS is addressed as approximate Fuzzy-UCS.

Thus, our analysis consists of two parts:

- We first illustrate how both representations approximate the decision boundaries of an

artificial problem with complex decision boundaries. The study demonstrates how the approximate representation can fit the training examples more accurately.

- Then, we compare the differences in terms of interpretability and accuracy between 1) the three inference schemes of linguistic Fuzzy-UCS and 2) linguistic Fuzzy-UCS versus approximate Fuzzy-UCS.

As follows, in section 8.5.1, we first design an approximate representation for Fuzzy-UCS. Then, sections 8.5.2 and 8.5.3 respectively develop each one of the two studies.

8.5.1 Approximate Fuzzy-UCS

In the approximate representation (Orriols-Puig et al., 2008h), the rule is similar to the descriptive one, but the variables in the rule condition take fuzzy sets instead of linguistic terms. Thus, the approximate fuzzy rule has the following form:

$$\mathbf{IF} \ x_1 \text{ is } FS_1^k \text{ and } \dots \text{ and } x_n \text{ is } FS_n^k \ \mathbf{THEN} \ c_j \ \mathbf{WITH} \ F^k, \quad (8.13)$$

where each variable x_i is represented by an independent fuzzy set FS_i , and each fuzzy set is defined by

$$FS_i = (a, b, c), \quad (8.14)$$

where a, b, and c are the x-axis value of the lower, middle and upper vertices of a triangular-shaped membership function, i.e.,

$$\mu_{FS_i} = \begin{cases} \frac{x-a}{b-a}, & a \leq x < b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & \text{otherwise.} \end{cases} \quad (8.15)$$

The operators that directly manipulate the rules were adapted to deal with the approximate representation. This includes matching, covering, crossover, mutation, and subsumption, which are explained in the following sections. Moreover, the inference process was also revised.

Matching. The matching operator calculates the matching degree of each input variable with its corresponding fuzzy set and aggregates all them by means of a T-norm (conjunction). As before, we used the product as T-norm. Note that the main difference with respect to linguistic Fuzzy-UCS is that, now, each variable is represented by a single semantic-free triangular shaped membership function.

Covering. The covering operator creates an independent triangular-shape fuzzy set for each input variable as follows.

$$a = \text{rand} \left(\min_i - \frac{\max_i - \min_i}{2}, e_i \right); \quad (8.16)$$

$$b = e_i; \quad (8.17)$$

$$c = \text{rand} \left(e_i, \max_i + \frac{\max_i - \min_i}{2} \right); \quad (8.18)$$

where min_i and max_i are the minimum and maximum value that the attribute i can take (both values are extracted from the training data set), e_i is the attribute i of the example e for which covering has been fired, and $rand$ generates a random number between both arguments. Thus, covering creates a triangle-shaped fuzzy set that maximally matches the input instance.

Crossover. The crossover operator generates a new offspring from two parents by crossing the rule antecedent as follows. First, it crosses the middle vertex b of the fuzzy membership function:

$$b_{child_1} = b_{parent_1} \cdot \alpha + b_{parent_2} \cdot (1 - \alpha); \quad (8.19)$$

$$b_{child_2} = b_{parent_1} \cdot (1 - \alpha) + b_{parent_2} \cdot \alpha; \quad (8.20)$$

where $0 \leq \alpha \leq 1$ is a configuration parameter. As we wanted to generate offspring whose middle vertex b was close to the middle vertex of one of his parents, we set $\alpha = 0.005$ in our experiments. Next, for both children, the procedure to cross the most-left and most-right vertices is the following. First, the two most-left and two most-right vertices are chosen

$$min_{left} = \min(a_{parent_1}, a_{parent_2}, b_{child}); \quad (8.21)$$

$$mid_{left} = \text{middle}(a_{parent_1}, a_{parent_2}, b_{child}); \quad (8.22)$$

$$mid_{right} = \text{middle}(c_{parent_1}, c_{parent_2}, b_{child}); \quad (8.23)$$

$$max_{right} = \max(c_{parent_1}, c_{parent_2}, b_{child}). \quad (8.24)$$

And then, these two values are used for generating the most-left and most-right vertices:

$$a_{child} = \text{rand}(min_{left}, mid_{left}); \quad (8.25)$$

$$c_{child} = \text{rand}(mid_{right}, max_{right}); \quad (8.26)$$

where the functions min , $middle$, and max return respectively the minimum, middle, and maximum values between their arguments.

Mutation. The mutation operator decides randomly if each vertex of a variable has to be mutated. The central vertex is mutated as follows:

$$b = \text{rand}(b - (b - a) \cdot m_0, b + (c - b) \cdot m_0), \quad (8.27)$$

where m_0 ($0 < m_0 \leq 1$) defines the strength of the mutation. The left-most vertex is mutated as

$$a = \begin{cases} \text{rand}(a - \frac{b-a}{2} \cdot m_0, a) & \text{if } F > F_0 \text{ \& no crossover} \\ \text{rand}(a - \frac{b-a}{2} \cdot m_0, a + \frac{b-a}{2} \cdot m_0) & \text{otherwise.} \end{cases} \quad (8.28)$$

And the right-most vertex

$$c = \begin{cases} \text{rand}(c - \frac{c-b}{2} \cdot m_0, c) & \text{if } F > F_0 \text{ \& no crossover} \\ \text{rand}(c - \frac{c-b}{2} \cdot m_0, c + \frac{c-b}{2} \cdot m_0) & \text{otherwise.} \end{cases} \quad (8.29)$$

That is to say, if the rule is accurate enough ($F > F_0$) and has not been generated through crossover, mutation forces to generalize it. Otherwise, it can be either generalized or specified. In this way, we increase the pressure toward maximally general and accurate rule sets.

Subsumption. Subsumption was redefined as follows. We considered that a classifier k_1 , which is experienced enough ($exp^{k_1} > \theta_{sub}$) and accurate ($F^{k_1} > F_0$), could subsume another classifier k_2 if for each variable i

$$a_{k_1}^i \leq a_{k_2}^i; \quad (8.30)$$

$$c_{k_1}^i \geq c_{k_2}^i; \quad (8.31)$$

$$b_{k_1}^i - (b_{k_1}^i - a_{k_1}^i) \cdot \delta \leq b_{k_2}^i \leq b_{k_1}^i + (c_{k_1}^i - b_{k_1}^i) \cdot \delta; \quad (8.32)$$

where δ is a discount parameter (in our experiments we set $\delta = 0.001$). Thus, a rule's condition subsumes another if the supports of the subsumed rule are enclosed in the supports of the subsumer rule, and the middle vertex of their triangular-shaped fuzzy sets are close in the feature space.

Inference. Given a new test example, the most likely output is the class predicted by the rule k that maximizes $F^k \cdot \mu_{A^k}(e)$. We have considered this action winner scheme as inference process because the prototype-based representation considered in the approximate approach inherently advocates independence among the fuzzy rules.

8.5.2 Decision Boundaries: Study on an Artificial Domain

Before proceeding with a large comparison between the two types of representations and the three types of inference algorithms in the linguistic representation, we first analyzed linguistic Fuzzy-UCS and approximate Fuzzy-UCS on a case study. We also included UCS with interval-based representation (Bernadó-Mansilla and Garrell, 2003; Orriols-Puig and Bernadó-Mansilla, 2006b; Orriols-Puig and Bernadó-Mansilla, 2008) in the analysis. We graphically studied how the two fuzzy representations approximated the decision boundaries of an artificially designed domain with respect to interval-based UCS. We chose a two-dimensional problem to facilitate the visualization: the *tao* problem (Bernadó-Mansilla et al., 2002) (see figure 8.5(a)). This problem presents curved-shaped boundaries, whose approximation poses a challenge to the linguistic fuzzy representation. Moreover, we compared the training accuracies, as well as the size of the evolved rule set. This analysis was restricted to the features of the tested problem, and only estimated the training error; thus, our aim was not to extract general conclusions, but to provide an intuitive visualization of the knowledge evolved by the different techniques. This analysis is complemented in the next section, where the three learners are compared in a set of real-world problems.

We configured UCS with the following parameter values: $numIter=100\,000$, $N=6\,400$, $acc_0 = 0.99$, $\nu=10$, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\}=50$, $\chi=0.8$, $\mu=0.04$, $\delta=0.1$, $r_0=0.2$. For Fuzzy-UCS, we used the default configuration (see section 8.4), except for $P_{\#} = 0.2$. We modified this configuration parameter only for the case study; in all the remaining experiments, the default configuration is used. This change was because we aimed at initializing the population with quite specific rules since the problem has only two dimensions and a high density of instances. Besides, for the approximate representation we set $r_0 = 0.2$. The three types of inference presented in section 8.3.5 were used: weighted average (wavg), action winner (awin), and fittest rules (nfit). Figure 8.5(b) depicts the boundaries evolved by interval-based UCS. Figures 8.6, 8.7, and 8.8 report the decision boundaries for linguistic Fuzzy-UCS with weighted average inference, action winner inference, and fittest rules inference respectively. In each case, we experimented with

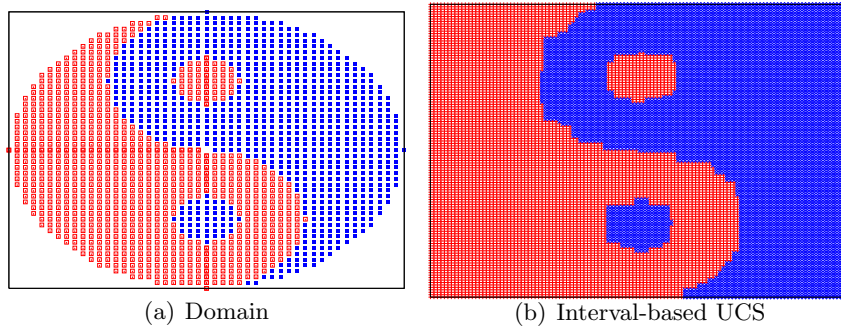


Figure 8.5: (a) Domain of the tao problem and (b) decision boundaries obtained by UCS.

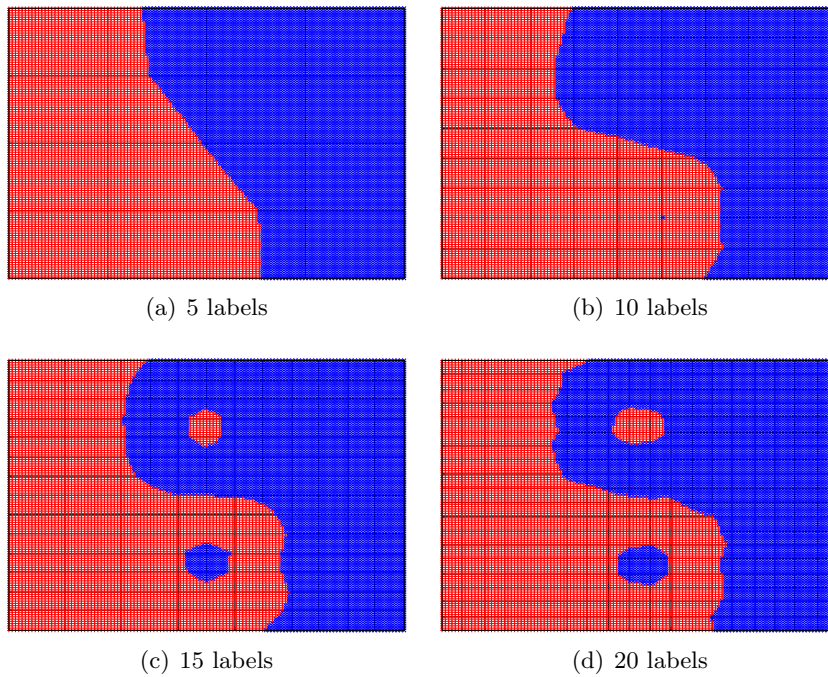


Figure 8.6: Decision boundaries obtained by linguistic Fuzzy-UCS with weighted average inference and 5 (a), 10 (b), 15 (c) and 20 (d) linguistic terms per variable.

5, 10, 15, and 20 linguistic terms per variable; the grid in the plots indicates the partitions in the feature space made by the cross-points of the triangular membership functions associated with the different fuzzy sets. Figure 8.9 shows the decision boundaries obtained by approximate Fuzzy-UCS. Table 8.4 summarizes the training accuracies and population sizes in each case. The results are averages over ten runs with different seeds.

Several observations can be drawn from the evolved decision boundaries. Firstly, the results show the generalization capabilities of all learners. The rules tend to expand as much as possible while they are accurate, covering regions in the feature space where there are no examples. This

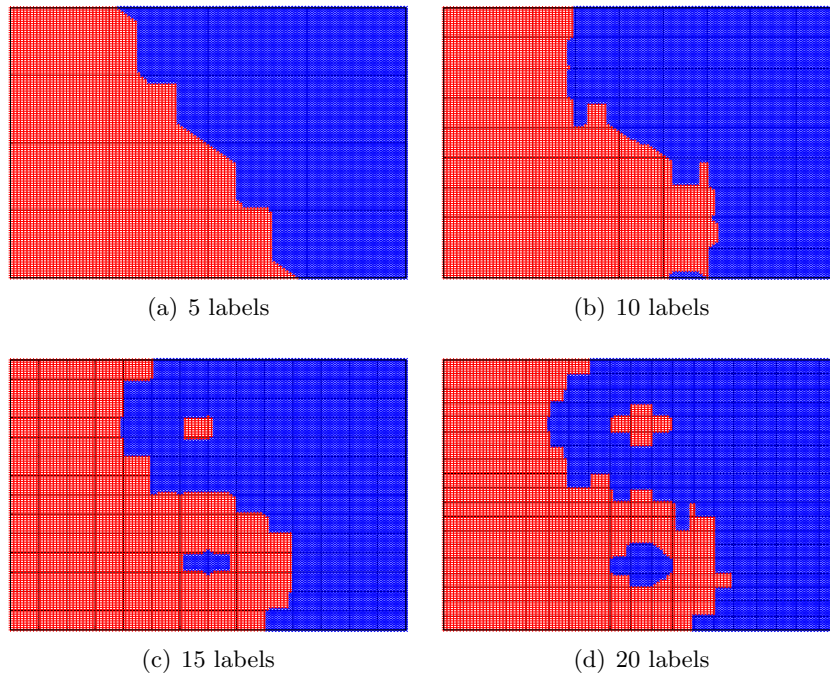


Figure 8.7: Decision boundaries obtained by linguistic Fuzzy-UCS with action winner inference and (a) 5, (b) 10, (c) 15, (d) and 20 linguistic terms per variable.

generalization pressure is mostly due to subsumption, which replaces the offspring for most general and accurate rules when possible. Thus, this operator gives highly general and accurate rules more strength.

Interval-based UCS reached the maximum accuracy among all learners. It evolved a population consisting of 1230 rules which accurately defined the decision boundaries (see figure 8.5(b)), with 99.8% training accuracy. The accuracy obtained by linguistic Fuzzy-UCS depended on the number of linguistic terms per variable (see the models built in figures 8.6, 8.7, and 8.8). With 5 linguistic labels per variable, linguistic Fuzzy-UCS could not discover the two inner concepts of the tao problem regardless of the used inference method. The models only defined one linear class boundary that did not fit the curved boundary of the domain accurately. As the number of linguistic terms per variable increased, the boundaries were defined more accurately. With 20 linguistic terms per variable, the three types of inference achieved high training performances.

The models evolved by linguistic Fuzzy-UCS with the three types of inference differed in the shape of the decision boundaries and the rule set size. Weighted average inference defined smooth boundaries which resulted from the vote of several overlapping rules (see figure 8.6). However, it maintained a large number of rules in the final population. Action winner inference created more reduced rule sets, but the boundaries were more abrupt. Note that the decision boundaries followed the partitions produced by the fuzzy membership functions, especially when 15 and 20 linguistic terms were used. This is because only the rules that maximized the product of $\mu_{A^k}(e) \cdot F$ were kept in the final population. Fittest rules inference evolved the most compact rule sets. Furthermore, the boundaries were smoother than the ones obtained with action winner

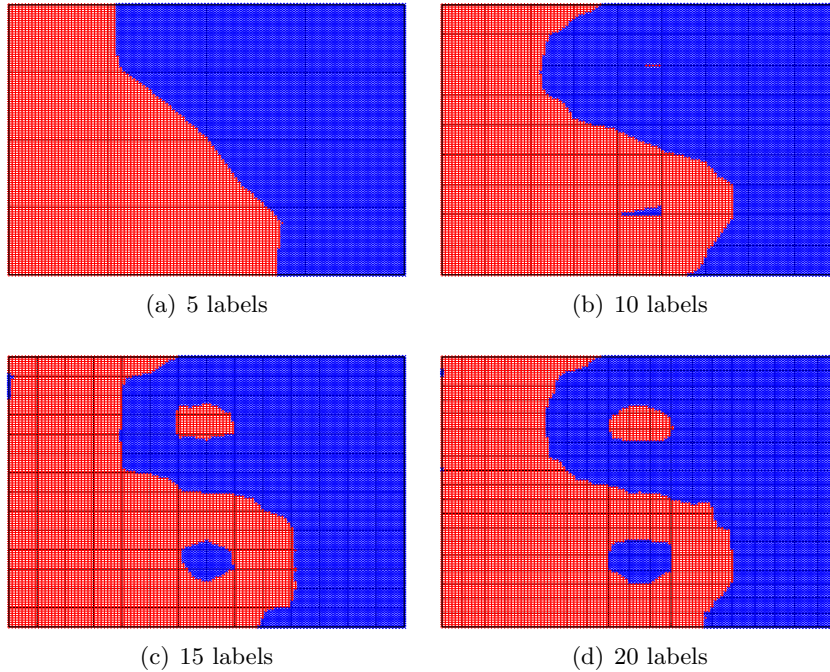


Figure 8.8: Decision boundaries obtained by linguistic Fuzzy-UCS with fittest rules inference and (a) 5, (b) 10, (c) 15, (d) and 20 linguistic terms per variable.

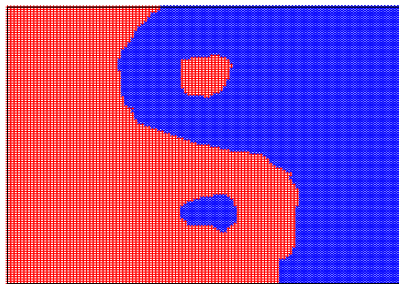


Figure 8.9: Decision boundaries obtained by approximate Fuzzy-UCS.

scheme. This type of inference maintained the most numerous and accurate rules in the final population. As this process could insert overlapping rules into the final population, the weighted average inference was used to infer the class, thus forwarding the interpolative reasoning. For this reason the decision boundaries were not as abrupt as the ones evolved by the action winner inference.

Finally, figure 8.9 shows that approximate Fuzzy-UCS built a model that accurately fitted the training examples. Approximate Fuzzy-UCS used an inference method based on action winner, similar to that used in linguistic Fuzzy-UCS. Regardless of this inference scheme, the decision boundaries were smoother because each variable evolved an independent fuzzy set. However, as a consequence of not sharing a unique semantic, the interpretability of the fuzzy-rules was

Table 8.4: Summary of Fuzzy UCS results with interval-based, approximate and linguistic representation with 5, 10, 15, and 20 linguistic terms per variable in the tao problem. Columns show the training accuracy and the number of rules for action winner and weighted average inference schemes.

	Training acc.			Num. rules		
<i>Interval-based UCS</i>	99.80			1230		
<i>App. Fuzzy-UCS</i>	96.94			555		
	<i>wavg</i>	<i>awin</i>	<i>nfit</i>	<i>wavg</i>	<i>awin</i>	<i>nfit</i>
<i>Lin. Fuzzy-UCS 5L</i>	82.95	83.24	88.31	112	17	15
<i>Lin. Fuzzy-UCS 10L</i>	91.85	91.19	91.85	441	78	30
<i>Lin. Fuzzy-UCS 15L</i>	96.68	94.74	96.68	618	144	52
<i>Lin. Fuzzy-UCS 20L</i>	97.15	95.57	97.15	763	200	65

degraded with respect to the linguistic representation. For example, one of the most numerous rules evolved by approximate Fuzzy-UCS was:

$$\mathbf{IF} \ x_1 \text{ is } (-3.3, -1.50, -1.13) \ \mathbf{and} \ x_2 \text{ is } (5.50, 6.48, 11.85) \ \mathbf{THEN} \ c_1 \ \mathbf{WITH} \ w = 0.998, \quad (8.33)$$

where each variable was represented by a triangular-shaped fuzzy set whose vertices could take any possible value in the feature space. Moreover, note that the number of rules evolved by approximate Fuzzy-UCS was larger than those created by any configuration of linguistic Fuzzy-UCS, except for linguistic Fuzzy-UCS with weighted average inference with 15 and 20 linguistic terms per variable. This large number of rules degraded even more the interpretability of the semantic-free approach.

8.5.3 Comparison Between Linguistic and Approximate Representations

This section furthers the study on the three types of inference of linguistic Fuzzy-UCS and compares them to the approximate representation. Specifically, we examine the trade-off between precision and rule set size already pointed out in the previous section for the three types of inference in linguistic Fuzzy-UCS. Besides, we include approximate Fuzzy-UCS in the comparison, which is expected to fit the training data more accurately. Considering the approximate representation, we aim to a) confirm the intuition that the approximate representation permits fitting significantly better the training instances, b) analyze whether this improvement is also present in the prediction of previously unseen instances, and c) evaluate the impact of the approximate representation on the interpretability of the evolved rule set.

Methodology

We selected a collection of 20 real-world data sets whose characteristics are summarized in table 8.1. All the data sets were obtained from the UCI Repository (Asuncion and Newman, 2007), except for *tao*, which was selected from a local repository (Bernadó-Mansilla et al., 2002).

To measure the precision of the method in fitting the training instances, we used the training accuracy rate, i.e., the proportion of correctly classified examples of the training set. The performance of the method was measured by the test accuracy rate, i.e., the proportion of correct predictions on previously unseen instances. To obtain reliable estimates of these metrics, we used a ten-fold cross validation procedure (Dietterich, 1998). We collected the evolved rule set sizes to compare the interpretability of the three configurations of linguistic Fuzzy-UCS. Since the types of rules created by the linguistic representation are different from those of the approximate representation, we qualitatively compared the rule sets built by both approaches.

The results were statistically analyzed following the recommendations pointed out by Demšar (2006). In all the analysis, we used non-parametric statistical tests to compare the results obtained by the different learning algorithms. Parametric tests require that the input data (in our case, the tables of results) satisfy strong conditions, and the tests to check these conditions need large amounts of data (i.e., large number of data sets) to be effective (Sheskin, 2000). For this reason, non-parametric tests are recommended (Demšar, 2006), since they relax the requirements on the input data.

We applied multiple-comparison statistical procedures to test the null hypothesis that all the learning algorithms performed equivalently on average. Specifically, we used the Friedman’s test (Friedman, 1937, 1940), a non-parametric equivalent of the repeated-measures ANOVA (Fisher, 1959). If the Friedman’s test rejected the null hypothesis, we used the non-parametric Nemenyi test (Nemenyi, 1963) to compare all learners to each other. The Nemenyi test is said to be quite conservative, especially when a large number of learners is compared, so that it might not detect some existent differences between the learners. Therefore, we complemented the statistical analysis by comparing the performance of each pair of learners by means of the non-parametric Wilcoxon signed-ranks test (Wilcoxon, 1945). The approximate p-values resulting from the pairwise analysis, calculated as indicated in (Sheskin, 2000), were provided in the analysis. For further information about the statistic tests, the user is referred to appendix B.

We used the default configuration for Fuzzy-UCS (see section 8.4), since it used equivalent parameter values to those usually set for XCS and UCS. Moreover, we fixed the number of linguistic labels to 5. We did not consider a larger number of linguistic terms since it could hinder the interpretability desired in a linguistic representation. For approximate Fuzzy-UCS, we fixed $r_0 = 1$.

Results

Our first concern was to analyze the precision in fitting the training instances of linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS. Thus, we computed the training accuracy obtained with the four approaches, as reported in table 8.5. The two last rows supply the average rank and the position of each algorithm in the ranking. The ranks were calculated as follows. For each data set, we ranked the learning algorithms according to their performance; the learner with highest accuracy held the first position, whilst the learner with the lowest accuracy held the last position of the ranking. If a group of learners had the same performance, we assigned the average rank of the group to each of the learners in the group.

The multiple-comparison test enabled us to reject the null hypothesis that all learners were equally accurate at a significance level of 0.001. Thus, we ran the Nemenyi test at a significance

Table 8.5: Comparison of the training accuracy of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fittest rules inference (nfit), and approximate Fuzzy-UCS on a set of twenty real-world problems.

	Linguistic			Approximate
	wavg	awin	nfit	
<i>ann</i>	99.35	98.34	99.48	98.83
<i>aut</i>	99.30	92.67	98.87	97.82
<i>bal</i>	91.03	90.97	89.97	98.61
<i>bpa</i>	68.57	68.30	69.79	86.29
<i>cmc</i>	67.34	67.77	70.70	65.29
<i>col</i>	93.04	91.76	96.22	98.95
<i>gls</i>	71.04	65.84	71.46	94.46
<i>h-c</i>	89.75	91.11	92.02	98.77
<i>h-s</i>	94.75	92.46	96.70	98.92
<i>irs</i>	95.78	95.59	94.56	97.47
<i>pim</i>	77.05	77.74	79.16	89.91
<i>son</i>	100.00	99.89	99.50	99.91
<i>tao</i>	81.70	83.31	87.42	89.64
<i>thy</i>	89.03	89.92	92.62	95.70
<i>veh</i>	76.77	72.97	77.49	89.52
<i>wbcd</i>	96.38	95.97	96.51	99.69
<i>wdbc</i>	96.34	95.50	96.18	99.55
<i>wne</i>	98.48	97.28	98.12	100.00
<i>wpbc</i>	97.57	94.01	95.39	96.98
<i>zoo</i>	99.71	99.98	99.90	100.00
Rank	<i>2.70</i>	<i>3.45</i>	<i>2.40</i>	<i>1.45</i>
Pos	<i>2</i>	<i>4</i>	<i>3</i>	<i>1</i>

level of 0.10. Figure 8.10 ranks the four learners and connects those that performed equivalently according to the Nemenyi procedure. The test indicates that approximate Fuzzy-UCS achieved significantly better training performance than all the other algorithms. Moreover, linguistic Fuzzy-UCS with action winner significantly degraded the training performance achieved with Fuzzy-UCS with fittest rules inference. As the Nemenyi test is said to be quite conservative, we also performed pairwise comparisons between the learners by means of the non-parametric Wilcoxon signed-ranks test. Table 8.6 provides the approximate p-values. The \oplus and \ominus symbols indicate that the method in the row significantly improved/degraded the performance obtained with the method in the column. The $+$ and $-$ symbols denote a non-significant improvement/degradation. The pairwise analysis confirmed the conclusions extracted from the Nemenyi test. No other significant differences were found by this statistical test.

As expected, the approximate representation fitted the training examples more accurately since there was no semantic shared among all variables—that is, each variable could define its own fuzzy sets. Next, we analyzed if this improvement was also present in the test performance,

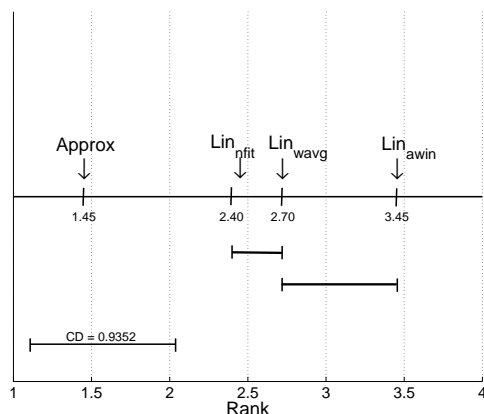


Figure 8.10: Comparison of the training performance of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.

Table 8.6: Pairwise comparisons of the training accuracy achieved by linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS.

	wavg	awin	nfit	approx
wavg		.0793	.0929	.0017
awin	—		.0012	.0002
nfit	+	\oplus		.0017
approx	\oplus	\oplus	\oplus	

which is shown in table 8.7. The multiple-comparison test rejected the hypothesis that all learners performed the same on average at a significance level of 0.001. Figure 8.11 shows the rank of each method and connects the groups of learners that performed equivalently according to the Nemenyi test at a significance level of 0.10. The statistical procedure identified two groups of techniques that performed equivalently. The first group included linguistic Fuzzy-UCS with weighted average inference and approximate Fuzzy-UCS. The second group comprised approximate Fuzzy-UCS and linguistic Fuzzy-UCS with action winner and fittest rules inferences. The same significant differences were found with the pairwise statistical analysis. Table 8.8 supplies the approximate p-values calculated from the Wilcoxon signed-ranks test.

Although the flexibility of the approximate representation allowed Fuzzy-UCS to evolve models that fitted the training examples more accurately, no significant differences were observed in the prediction of previously unseen instances. Moreover, approximate Fuzzy-UCS did not perform as well as linguistic Fuzzy-UCS with weighted average inference, though the difference was not statistically significant. Further analysis pointed out that approximate Fuzzy-UCS was over-fitting the training data in some of the tested domains. To contrast this hypothesis, we monitored the evolution of the training and test performance of the problems in which approx-

Table 8.7: Comparison of the test accuracy of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fittest rules inference (nfit), and approximate Fuzzy-UCS on a set of twenty real-world problems.

	Linguistic			Approximate
	wavg	awin	nfit	
<i>ann</i>	98.85	97.39	98.61	95.44
<i>aut</i>	74.42	67.42	69.32	69.54
<i>bal</i>	88.65	84.40	83.40	82.73
<i>bpa</i>	59.82	59.42	58.93	64.19
<i>cmc</i>	51.72	49.67	49.42	44.79
<i>col</i>	85.01	82.46	78.50	87.96
<i>gls</i>	60.65	57.21	57.43	71.82
<i>h-c</i>	84.39	82.62	82.05	80.16
<i>h-s</i>	81.33	80.78	78.11	78.59
<i>irs</i>	95.67	95.47	93.73	95.80
<i>pim</i>	74.88	74.11	74.32	74.32
<i>son</i>	80.78	73.71	71.66	76.34
<i>tao</i>	81.71	83.02	87.53	89.39
<i>thy</i>	88.18	89.49	91.25	92.28
<i>veh</i>	67.68	65.35	65.34	65.80
<i>wbcd</i>	96.01	95.73	95.29	95.69
<i>wdbc</i>	95.20	94.61	94.51	93.87
<i>wne</i>	94.12	94.86	91.82	95.42
<i>wpbc</i>	76.06	76.05	71.69	59.78
<i>zoo</i>	96.50	94.78	95.90	83.53
Rank	<i>1.60</i>	<i>2.75</i>	<i>3.20</i>	<i>2.45</i>
Pos	<i>1</i>	<i>3</i>	<i>4</i>	<i>2</i>

imate Fuzzy-UCS degraded the results obtained by linguistic Fuzzy-UCS with any inference type. Figure 8.12 plots the evolution of the training and test performance for the *bal* problem. During the first 5 000 learning iterations, both training and test performance rapidly increased, achieving about 90% and 84% accuracy rates respectively. After that, the training performance continued to increase whilst the test performance slightly decreased. After 100 000 iterations, the training performance reached 98%; nonetheless, the test performance decreased to 82%. Thus, at a certain point in the learning process, the flexibility of the approximate representation led Fuzzy-UCS to over-fit the training instances in order to create more accurate classifiers, which was detrimental to the test performance.

Finally, table 8.9 shows the number of rules evolved in each configuration. Friedman's test rejected the hypothesis that the population sizes were equivalent on average at a significance level of 0.001. The post-hoc Nemenyi test supported the hypothesis that the four learners evolved populations with significantly different sizes. The pairwise comparisons yielded the same conclusions (see in table 8.10 the approximate p-values according to a Wilcoxon signed-

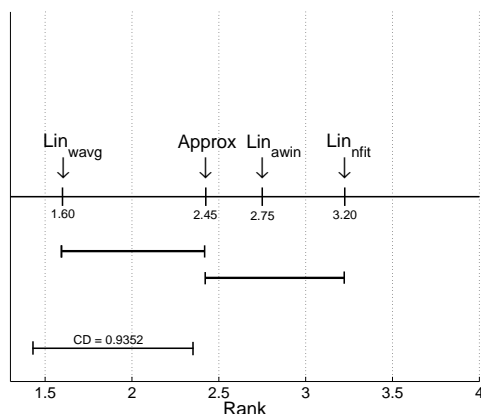


Figure 8.11: Comparison of the test performance of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.

Table 8.8: Pairwise comparisons of the test accuracy achieved by linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS.

	wavg	awin	nfit	approx
wavg		.0032	.0051	.1913
awin	\ominus		.2627	.7369
nfit	\ominus	—		.4781
approx	—	+	+	

ranks test). In fact, a simple quantitative analysis highlighted the differences in the population sizes. Fuzzy-UCS with weighted average inference built populations that consisted of thousands of rules. Consequently, although using a linguistic representation, this large number of rules hampered the interpretability of the rule set. Approximate Fuzzy-UCS resulted in smaller populations; however, these consisted of hundreds of rules. This, together with the loss of interpretability due to the approximate representation, hindered the readability of the rule set. The other two types of inference of Linguistic Fuzzy-UCS, especially the fittest rules inference, resulted in populations with a moderate number of rules. Fuzzy-UCS with fittest rules inference built populations that ranged from tens of to few hundreds of rules.

These results showed the performance-interpretability trade-off in linguistic Fuzzy-UCS already pointed out in the previous section. Weighted average inference significantly outperformed the other two inference schemes since it combined the knowledge of all experienced rules in the final population. As shown in the case study of the previous section, this allowed Fuzzy-UCS to fit complex boundaries even though the fuzzy representation made a discretization of the feature space. Linguistic Fuzzy-UCS could approximate these boundaries by means of evolving a set of partially overlapping fuzzy rules. However, the interpretability of the rule set was degraded by

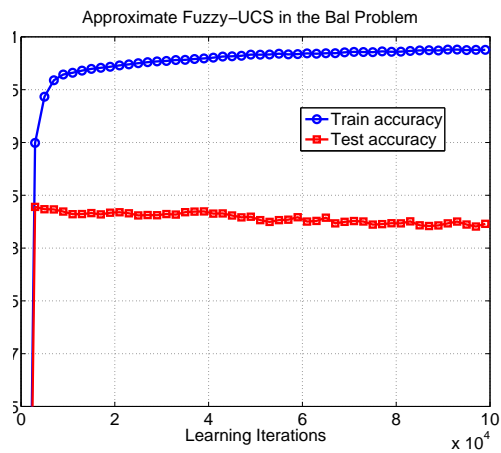


Figure 8.12: Evolution of the training and test accuracies with approximate Fuzzy-UCS on the *bal* problem.

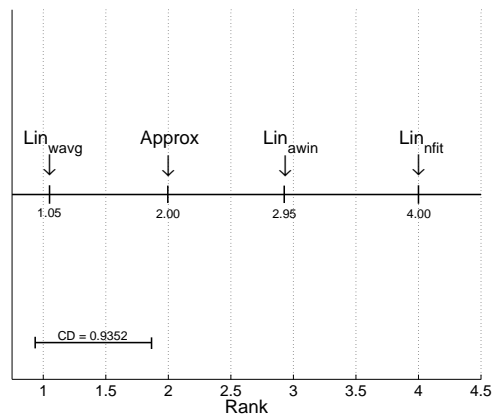


Figure 8.13: Comparison of the number of rules evolved by all learners against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.

the large number of rules. The other two inference schemes considerably improved the readability, since they produced large reductions of the rule set. Nonetheless, this went against the test performance, which was significantly surpassed by the weighted average inference scheme.

The overall results presented in this section pointed out the viability of the linguistic representation with respect to its approximate counterpart. While approximate Fuzzy-UCS created models that fitted the training data very accurately, there was no statistical evidence of this improvement in the test performance. Furthermore, we also identified that approximate Fuzzy-UCS may over-fit the training instances in complex domains. Thus, the flexibility provided by the approximate representation did not imply an improvement of the test accuracy, although it degraded the readability of the fuzzy-rules. Besides, the rule sets created by approximate Fuzzy-

Table 8.9: Comparison of the population sizes of linguistic Fuzzy-UCS with weighted average (wavg), action winner (awin), and most numerous and fittest rules inference (nfit), and approximate Fuzzy-UCS on a set of twenty real-world problems.

	Linguistic			Approximate
	wavg	awin	nfit	
<i>ann</i>	2769	75	36	409
<i>aut</i>	3872	114	74	158
<i>bal</i>	1212	114	75	441
<i>bpa</i>	1440	73	39	207
<i>cmc</i>	1881	430	271	402
<i>col</i>	4135	154	81	297
<i>gls</i>	2799	62	36	146
<i>h-c</i>	3574	113	46	257
<i>h-s</i>	3415	117	62	231
<i>irs</i>	480	18	7	103
<i>pim</i>	2841	192	62	538
<i>son</i>	3042	178	160	186
<i>tao</i>	111	19	14	464
<i>thy</i>	1283	37	11	134
<i>veh</i>	3732	332	147	532
<i>wbcd</i>	3130	138	28	360
<i>wdbc</i>	5412	276	101	490
<i>wne</i>	3686	95	26	160
<i>wpbc</i>	3772	156	115	175
<i>zoo</i>	773	16	10	55
Rank	<i>1.05</i>	<i>2.95</i>	<i>4.00</i>	<i>2.00</i>
Pos	<i>1</i>	<i>3</i>	<i>4</i>	<i>2</i>

Table 8.10: Pairwise comparisons of the sizes of the rule sets evolved by linguistic Fuzzy-UCS with the three types of inference and approximate Fuzzy-UCS.

	wavg	awin	nfit	approx
wavg		.0001	.0001	.0001
awin	\ominus		.0001	.0001
nfit	\ominus	\ominus		.0001
approx	\ominus	\oplus	\oplus	

UCS were significantly larger than the ones obtained by linguistic Fuzzy-UCS with action winner and fittest rules inference. For all these reasons, we focus our analysis on linguistic Fuzzy-UCS in the remainder of this chapter, and leave further analysis of approximate Fuzzy-UCS as future research.

8.6 Comparison of Fuzzy-UCS to Several Machine Learning Techniques

So far, we have clearly shown the competitiveness of linguistic Fuzzy-UCS with respect to its approximate counterpart. In this section, we study whether the behavior of linguistic Fuzzy-UCS is comparable to some of the most-used machine learning techniques. For this purpose, we compared Fuzzy-UCS to two sets of learners: fuzzy rule-based learners and “non-fuzzy” (crisp) learners. With the former comparison, we analyzed the behavior of Fuzzy-UCS with respect to other techniques that use the same representation, which may limit the maximum performance that can be achieved in certain domains. With the latter comparison, we study whether, even with the limitations that may impose the fuzzy representation, Fuzzy-UCS is competitive with a large number of the most-representative learners, regardless of the knowledge representation they use. Below, we first present the experimental methodology, and then compare Fuzzy-UCS to the other learners.

8.6.1 Experimental Methodology

The followed methodology is similar to the one presented in the previous section. We selected the same collection of 20 real-world problems, whose characteristics are summarized in table 8.1. The experiments were ran on a ten-fold cross validation, and the test accuracy rate was used to measure the performance of the different learners.

The performance of Fuzzy-UCS was compared with a large variety of learning algorithms, which we organized in two groups. The first group consisted of the following fuzzy rule-based classification systems: Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost, Fuzzy LogitBoost, and Fuzzy MaxLogitBoost. Fuzzy GP (Sánchez and Couso, 1998, 2000; Sánchez et al., 2001) is a genetic programming algorithm that builds a fuzzy classifier for each class of the domain by searching for a tree that represents an analytic expression that relates the input and the output variables as accurately as possible. Fuzzy GAP (Sánchez and Couso, 1998, 2000) works similarly to Fuzzy GP, but the optimization system is a hybrid between genetic algorithms and genetic programming. Fuzzy SAP (Sánchez et al., 2001) combines genetic operators with simulated annealing (Korst and Aarts, 1997) to create data models similar to those built by Fuzzy GP and Fuzzy GAP. Fuzzy AdaBoost (del Jesus et al., 2004) is a modification of the boosting algorithm AdaBoost (Freund and Schapire, 1996) to deal with fuzzy rules; Fuzzy AdaBoost generates a compound classifier which decides the output as a linear combination of the outputs of weak classifiers. Fuzzy LogitBoost (Otero and Sánchez, 2006) and Fuzzy MaxLogitBoost (Sánchez and Otero, 2007) are boosting algorithms that iteratively invoke a genetic algorithm to extract simple fuzzy rules that are combined to decide the output of new examples. The basic difference between both algorithms is that Fuzzy MaxLogitBoost may reject a new rule provided by the genetic algorithm if it does not improve the expected global performance. All these methods were run using KEEL (Alcalá-Fdez et al., 2008). We followed the recommended parameter values given in the KEEL platform to configure the methods (Alcalá-Fdez et al., 2008), which also corresponded to the settings used in the bibliography of these methods. We only changed the maximum population size of AdaBoost, LogitBoost, and MaxLogitBoost. We tried population sizes of $N=\{8, 25, 50, 100\}$ for all the data sets, and selected the results of $N=50$ since they generally allowed us to achieve higher performance ratios than $N=8$ and $N=25$, and did not

significantly differ from $N=100$. For all the methods, we used 5 linguistic terms per variable. Fuzzy-UCS was configured as detailed in section 8.5.3.

The second group gathered a large number of learners with different knowledge representations: ZeroR, C4.5, IBk, Naïve Bayes, Part, SMO, GAssist, and UCS. Among them, C4.5, IBk, Naïve Bayes, and SMO have been identified as the top ten data mining algorithms, including supervised and unsupervised learning techniques (Wu et al., 2007). Therefore, the comparison aims at measuring the quality of Fuzzy-UCS with several of the best learners. ZeroR is a simple classifier system that always predicts the majority class in the training data set. We employed this algorithm to provide a baseline result. C4.5 (Quinlan, 1995) is one of the most used decision trees, which derives from ID3 and introduces methods to deal with continuous variables and missing values. IBk (Aha et al., 1991) is a nearest neighbor algorithm; it decides the output of a new example as the most numerous class of the k nearest neighbors. Naïve Bayes (John and Langley, 1995) is a probabilistic classifier that estimates the parameters of a Bayesian model. Part (Frank and Witten, 1998) is a learning architecture that combines the creation of rules from partial decision trees and the separate-and-conquer rule learning technique to create a classifier without using global optimization. SMO (Platt, 1998) is a support vector machine (Vapnik, 1995) that implements the *Sequential Minimization Algorithm*. GAssist (Bacardit, 2004) is a recent Pittsburgh-style LCS. UCS (Bernadó-Mansilla and Garrell, 2003) is a Michigan-style LCS derived from XCS (Wilson, 1995, 1998) and specialized for supervised learning tasks (see chapter 3 for an extensive description of the system). All the methods except for GAssist and UCS were run using Weka (Witten and Frank, 2005). For GAssist, we used the open source code provided in (Bacardit, 2007). For UCS, we used our own code. If not stated differently, all open source methods were configured with the parameters values recommended by default. For UCS we set: $numIter=100\,000$, $N=6400$, $acc_0 = 0.99$, $\nu=10$, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\}=50$, $\chi=0.8$, $\mu=0.04$, $\delta=0.1$, $r_0=0.6$. Fuzzy-UCS was configured with standard values as indicated in the previous section.

We applied the following statistical analysis to the results. We used the non-parametric Friedman’s test (Friedman, 1937, 1940) to check whether all the learning algorithms performed the same on average. If significant differences were found, two procedures were applied to detect differences among methods. We first aimed at comparing the performance obtained by each of the inference types of Fuzzy-UCS to all other learners (instead of comparing all learners with the others as done in section 8.5). To achieve this, we applied the non-parametric Bonferroni-Dunn (Dunn, 1961) test. Moreover, the analysis is complemented by performing pairwise comparisons among the learners by means of a Wilcoxon signed-ranks test (Wilcoxon, 1945). For further details on the statistical tests, the reader is referred to appendix B.

8.6.2 Comparison to Fuzzy Rule-Based Classification Systems

In the following, we compare the test performance and the interpretability of Fuzzy-UCS with the three types of inference to the aforementioned set of fuzzy rule-based learners.

Comparison of the performance. Table 8.11 details the test accuracies obtained with the fuzzy classifiers Fuzzy AdaBoost, Fuzzy GAP, Fuzzy GP, Fuzzy LogitBoost, Fuzzy MaxLogitBoost, Fuzzy SAP and Fuzzy-UCS with three different types of inference: weighted average (wavg), action winner (awin), and fittest rules (nfit). The average performance of AdaBoost and MaxLogitBoost for the *ann* and *aud* problems is not provided since neither system was able to

Table 8.11: Comparison of the test accuracy of Fuzzy-UCS with weighted average (wavg), action winner (awin), and fittest rules (nfit), to Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost, Fuzzy LogitBoost, and Fuzzy MaxLogitBoost.

	GP	GAP	SAP	AdaBoost	LogitBoost	MaxLogitBoost	Fuzzy-UCS		
							wavg	awin	nfit
<i>ann</i>	77.86	77.20	78.02	-	76.20	-	98.85	97.39	98.61
<i>aut</i>	44.65	45.21	41.00	-	32.63	-	74.42	67.42	69.32
<i>bal</i>	69.73	64.33	65.80	85.54	88.30	75.58	88.65	84.40	83.40
<i>bpa</i>	56.62	57.91	62.30	65.34	64.46	56.53	59.82	59.42	58.93
<i>cmc</i>	47.00	46.57	46.27	49.55	51.10	45.21	51.72	49.67	49.42
<i>col</i>	79.15	73.51	81.89	63.06	63.06	63.06	85.01	82.46	78.50
<i>gls</i>	48.89	47.24	46.42	62.52	68.18	62.18	60.65	57.21	57.43
<i>h-c</i>	73.98	75.09	74.18	60.40	62.09	57.48	84.39	82.62	82.05
<i>h-s</i>	73.70	72.00	72.07	57.56	59.33	57.33	81.33	80.78	78.11
<i>irs</i>	94.47	90.80	91.53	95.47	95.33	92.00	95.67	95.47	93.73
<i>pim</i>	75.32	76.62	77.92	70.69	71.84	72.54	74.88	74.11	74.32
<i>son</i>	64.52	65.99	68.70	46.62	53.38	46.62	80.78	73.71	71.66
<i>tao</i>	80.36	81.75	81.15	91.46	91.73	84.52	81.71	83.02	87.53
<i>thy</i>	86.98	84.94	85.55	97.35	97.08	95.33	88.18	89.49	91.25
<i>veh</i>	46.16	44.59	42.96	30.82	37.25	38.05	67.68	65.35	65.34
<i>wbcd</i>	93.31	92.53	92.72	94.88	94.12	91.83	96.01	95.73	95.29
<i>wdbc</i>	90.93	90.49	91.52	37.26	62.74	37.26	95.20	94.61	94.51
<i>wne</i>	82.91	78.23	79.85	85.59	85.02	77.68	94.12	94.86	91.82
<i>wpbc</i>	74.77	74.47	74.37	23.65	76.35	23.65	76.06	76.05	71.69
<i>zoo</i>	71.18	66.65	66.08	41.89	41.89	41.89	96.50	94.78	95.90
Rank	<i>5.55</i>	<i>6.25</i>	<i>5.80</i>	<i>5.80</i>	<i>4.95</i>	<i>7.48</i>	<i>2.10</i>	<i>3.18</i>	<i>3.90</i>
Pos.	<i>5</i>	<i>8</i>	<i>6.5</i>	<i>6.5</i>	<i>4</i>	<i>9</i>	<i>1</i>	<i>2</i>	<i>3</i>

extract competent fuzzy rules from the two domains, leaving nearly all the feature space uncovered. The authors confirmed that this behavior could be due to the large number of nominal attributes that these two problems have. The last two rows of the table provide the average rank and the absolute position in the ranking of each learner.

The experimental results show that the three configurations of Fuzzy-UCS were the best ranked in the comparison. The next methods in the ranking were the boosting algorithm Fuzzy LogitBoost, the genetic programming-based systems Fuzzy-GP and Fuzzy-SAP, and Fuzzy AdaBoost. Finally, the last methods were Fuzzy GAP, and Fuzzy MaxLogitBoost.

We used the multiple-comparison Friedman's test to analyze whether the differences in the ranking were statistically significant. The statistical test rejected the hypothesis that all the methods performed the same on average at a significance level of 0.001. To evaluate the differences among them, we applied different statistical tests. First, we compared Fuzzy-UCS with each inference type with all the other learners. Figure 8.14 graphically represents the rank of

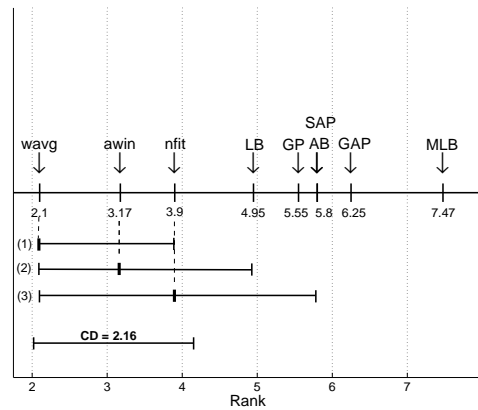


Figure 8.14: Comparisons of one learner against the others with the Bonferroni-Dunn test at a significance level of 0.1. All the learners are compared to three different control groups: (1) Fuzzy-UCS with weighted average inference, (2) Fuzzy-UCS with action winner inference, and (3) Fuzzy-UCS with fittest rules inference. The learners connected are those that perform equivalently to the control learner.

each learner and groups the classifiers that perform equivalently to (1) Fuzzy-UCS with weighted average inference, (2) Fuzzy-UCS with action winner inference, and (3) Fuzzy-UCS with fittest rules inference according to a Bonferroni-Dunn test at a significance level of 0.1. The statistical procedure supported the following hypotheses:

- Using Fuzzy-UCS with weighted average inference as the control learner, the statistical procedure supported the hypothesis that the performance of the control learner was equivalent to the performance of Fuzzy-UCS with the other two inference types. Moreover, Fuzzy-UCS with weighted average outperformed all the other learners.
- Using Fuzzy-UCS with action winner inference as the control learner, the test indicated that this learner performed equivalently to Fuzzy-UCS with the other two types of inference and Fuzzy LogitBoost.
- With respect to Fuzzy-UCS with fittest rules inference, the test did not reject the hypothesis that all the fuzzy learners except for Fuzzy MaxLogitBoost and Fuzzy GAP performed equivalently on average.

As the Bonferroni-Dunn test is said to be quite conservative (Sheskin, 2000), especially when a large number of learners are included in the analysis as in our experimentation, we complemented the statistical study by comparing each pair of learners. Table 8.12 shows the approximate p-values for the pairwise comparison according to a Wilcoxon signed-ranks test. The \oplus and \ominus symbols indicate that the method in the row significantly improves/degrades the performance obtained with the method in the column. Similarly, the $+$ and $-$ symbols denote a non-significant improvement/degradation. The $=$ symbol indicates that each method outperforms and degrades the other in the same number of data sets. Furthermore, figure 8.15

Table 8.12: Pairwise comparison of the test accuracy of fuzzy learners Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost (ABoost), Fuzzy LogitBoost (LBoost), Fuzzy MaxLogitBoost (MLBoost), and Fuzzy UCS with weighted average inference (wavg), action winner inference (awin), and fittest rules inference (nfit) by means of a Wilcoxon signed-ranks test.

	GP	GAP	SAP	ABoost	LBoost	MLBoost	Fuzzy-UCS		
							wavg	awin	nfit
GP		.0366	.2627	.0522	.4115	.0090	.0001	.0001	.0006
GAP	\ominus		.2180	.1005	.4781	.0187	.0002	.0002	.0002
SAP	-	+		.0674	.4330	.0111	.0003	.0005	.0036
ABoost	-	-	-		.0038	.0231	.0045	.0079	.0137
LBoost	=	=	=	\oplus		.0003	.0100	.0276	.0438
MLBoost	\ominus	\ominus	\ominus	\ominus	\ominus		.0005	.0009	.0007
wavg	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus		.0032	.0051
awin	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\ominus		.2627
nfit	\oplus	\oplus	\oplus	\oplus	\oplus	\oplus	\ominus	-	

graphically illustrates the significant differences between methods. That is, each method is depicted in one vertex of the graph, and significant improvements (at $\alpha=0.05$) of one learner with respect to another are plotted with a directed edge labeled with the corresponding p-value. To facilitate the visualization, Fuzzy AdaBoost and Fuzzy MaxLogitBoost were not included in the graph, since all the other learners significantly improved both methods, except for Fuzzy-GAP, which did not significantly outperform Fuzzy AdaBoost. At a significance level of 0.05, the test indicated that Fuzzy-UCS with weighted average inference significantly outperformed all the other learners, including the two other types of inference of Fuzzy-UCS. Moreover, Fuzzy-UCS with action winner and fittest inference schemes significantly improved the other fuzzy learners, i.e., Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost, Fuzzy LogitBoost, and Fuzzy MaxLogitBoost.

Comparison of the interpretability. The study conducted in section 8.5.3 already illustrated the interpretability-performance trade-off among the different inference schemes in Fuzzy-UCS. As shown, the excellent results of Fuzzy-UCS with weighted average with respect to all the other learners were hampered by the large number of fuzzy rules evolved by the method. The other two types of inference appeared as a positive alternative since, although they slightly degraded the accuracy rate with respect to the former approach, they resulted in a moderate number of rules. Aligned with these conclusions, we confirmed the suitability of Fuzzy-UCS by empirically demonstrating that the three schemes of Fuzzy-UCS resulted in significantly more accurate models than those obtained with all the other fuzzy learners. In this section, we further the study and qualitatively analyze if the rule set evolved by these two methods is competitive in terms of readability.

As the type of rules evolved by the systems differ, we qualitatively evaluated the size of the models by extracting some characteristics. Figure 8.16 shows examples of partial models evolved by the fuzzy learners for the *tao* problem. The models built by Fuzzy GP, Fuzzy GAP, and Fuzzy SAP consisted of a rule for each class of the domain. Each rule was directly extracted

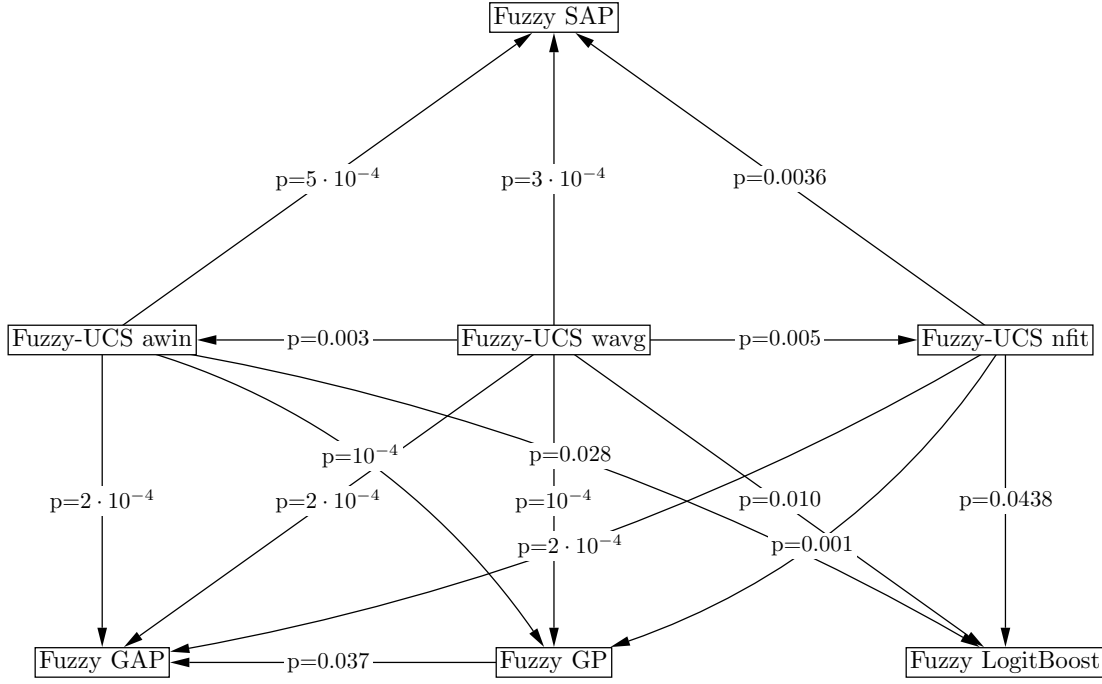


Figure 8.15: Illustration of the significant differences (at $\alpha = 0.05$) of the test accuracy among the fuzzy-methods and Fuzzy-UCS. An edge $L_1 \xrightarrow{p_{value}} L_2$ indicates that the learner L_1 outperforms the learner L_2 with the corresponding p_{value} . To facilitate the visualization, Fuzzy-AdaBoost and Fuzzy MaxLogitBoost, the two most outperformed algorithms, were not included in the graph.

from an expression codified in a tree. The rules were represented by an arbitrary number of conjunctions (AND) and disjunctions (OR) of conditions over the variables of the domain. One example of these types of rules for a two-dimensional problem is

$$\mathbf{IF} (x_1 \text{ is } \widetilde{A}_1^1 \text{ AND } x_2 \text{ is } \widetilde{A}_2^1) \text{ OR } (x_1 \text{ is } \widetilde{A}_1^2 \text{ AND } x_2 \text{ is } \widetilde{A}_2^2) \text{ THEN } c_1, \quad (8.34)$$

where each variable x_i was represented by a linguistic term $\widetilde{A}_i = \{ A_{i1} \vee \dots \vee A_{in_i} \}$. All variables shared the same semantics which were defined by the combination of triangular-shaped and trapezoidal-shaped fuzzy membership functions (see figure 8.16(a)).

On the other hand, the fuzzy rule-based boosting algorithms created a set of linguistic fuzzy rules that took the following form:

$$\mathbf{IF} x_1 \text{ is } \widetilde{A}_1 \text{ and } \dots \text{ and } x_n \text{ is } \widetilde{A}_n \text{ THEN } c_1 \text{ WITH } w_1^k, \dots, c_m \text{ WITH } w_m^k, \quad (8.35)$$

where each variable x_i was represented by a linguistic term $\widetilde{A}_i = \{ A_{i1} \vee \dots \vee A_{in_i} \}$. All variables shared the same semantics, which was defined by means of triangular-shaped fuzzy membership functions (see figure 8.16(b)). In the consequent part, the rule maintained a weight for each class, which were used for the inference process. Therefore, these individuals rules are less interpretable than the ones of Fuzzy-UCS, which use a single value—the fitness—to infer

```

if y is triangle(-6.0,-3.0,0.0) then red
if ( ( x is trapezoid(3.0, 6.0) or x is trapezoid(3.0, 6.0))
  or ( x is triangle(0, 3.0, 6.0) or x is trapezoid(3.0, 6.0)) )
  and
    ( x is triangle ( -3, 0.0, 3.0) or x is trapezoid(3.0, 6.0)
    or . . . )
  ...
then blue

```

(a) GP-based learners

```

if x is L and y is L then blue with -5.42 and red with 0.0
if x is M and y is XS then blue with 2.21 and red with 0.0
if x is M and y is XL then blue with -2.25 and red with 0.0
  ...

```

(b) Boosting learners

```

if x is XL then blue with  $w=1.00$ 
if x is XS then red with  $w=1.00$ 
if x is {XS or S} and y is {XS or S} then red with  $w=0.87$ 
  ...

```

(c) Fuzzy-UCS

Figure 8.16: Examples of part of the models evolved by (a) the GP-based methods, i.e., Fuzzy GP, Fuzzy GAP, and Fuzzy SAP; (b) the boosting learners, i.e., Fuzzy AdaBoost, Fuzzy LogitBoost, and Fuzzy MaxLogitBoost; and (c) Fuzzy-UCS for the two-dimensional tao problem. In the fuzzy learners, we used the following five linguistic terms per variable: {XS, S, M, L, XL}. All fuzzy learners use triangular-shaped membership functions. Moreover, GP-based learners also use trapezoid-shaped membership functions.

the class of test instances. The three boosting algorithms supported the absence of a variable by not assigning any linguistic term to the variable. The maximum size of the rule set was a configuration parameter. In our experiments, the maximum population size was set to 50.

To compare these two types of representations to the rule sets evolved by Fuzzy-UCS, we evaluated the size of the models as follows:

- We calculated the size of the models built by Fuzzy GP, Fuzzy GAP, and Fuzzy SAP by counting the number of AND, OR, and IS of the model. This gave us an idea of the average size of the rule. However, note that, due to the flexibility of these types of rules, it was not possible to directly compare them with the rules evolved by the three boosting algorithms and Fuzzy UCS. The rules constructed by Fuzzy GP, Fuzzy GAP, and Fuzzy SAP permit the combination of different logic operators, whose associativity and priority is given by the position of the operators in the tree. An equivalent conjunctive normal form for these rules could be found by applying De Morgan's laws. However, this transformation is not in the scope of this chapter, and so, we only qualitatively evaluated the model sizes.
- The size of the rule sets created by the boosting algorithms and Fuzzy-UCS were computed

with the following formula:

$$size = \sum_{i=1}^N \frac{1}{\ell} \sum_{j=1}^{\ell} \frac{maxLabels - numLabels(x_i)}{maxLabels - 1}, \quad (8.36)$$

where N is the number of rules in the population, ℓ is the number of variables, and $maxLabels$ is the number of linguistic labels (in our experiments, $maxLabels = 5$). This formula reckons the total number of variables in the model that have, at least, one linguistic term assigned. It also benefits general variables that have more than one linguistic label. To achieve a totally fair comparison, we also referred to the number of rules evolved by Fuzzy-UCS (see table 8.9).

Table 8.13 shows the size of the models created by each fuzzy learner. Table 8.14 illustrates the approximate p-values resulting from the pairwise comparison between the learners according to a Wilcoxon signed-ranks test. For the three methods based on genetic programming, we considered the average number of variables for each rule (i.e., column *is* divided by the number of classes of the problems). The comparison shows that Fuzzy SAP, followed by Fuzzy GP, Fuzzy GAP, and Fuzzy MaxLogitBoost, were the methods that created the smallest models according to a Wilcoxon signed-ranks test at a significance level of 0.05. We have already discussed how the representation of Fuzzy GP, Fuzzy GAP, and Fuzzy SAP was much more flexible and by far less interpretable than the representation of the other learners (see the number of conjunctions and disjunctions with different associativity and priority in the rules). Thus, although the number of attributes per rule was smaller, the interpretability of the model was poor due to the flexibility of the rule (see the partial example provided for the tao problem in figure 8.16(a)). Fuzzy-UCS with weighted average and with action winner inference created the largest and the second largest populations of the comparison respectively. On the other hand, Fuzzy-UCS with fittest rules inference created rule sets that, on average, were not significantly larger than the rule sets built by Fuzzy-GP, Fuzzy AdaBoost, and Fuzzy LogitBoost. Thus, disregarding the three learners based on genetic programming, whose rule sets were poorly readable due to the rule form, only Fuzzy MaxLogitBoost created more reduced populations. However, individual rules of Fuzzy MaxLogitBoost are less interpretable than those of Fuzzy-UCS since they maintain a weight per each class, and all these weights are used in the inference process.

The results provided in this section highlighted the high competitiveness of Fuzzy-UCS in terms of performance and interpretability with respect to other fuzzy learners. In terms of performance, Fuzzy-UCS with any of the three types of inference significantly outperformed all the other fuzzy learners. In terms of interpretability, Fuzzy-UCS with fittest rules inference evolved a number of rules comparable to those evolved by Fuzzy AdaBoost, Fuzzy LogitBoost, Fuzzy GP, Fuzzy GAP, and Fuzzy SAP. In the next section, we broaden the analysis and compare Fuzzy-UCS to a set of general purpose non-fuzzy learners.

8.6.3 Comparison with Non-Fuzzy Learners

Now, we compare Fuzzy-UCS to a set of general-purpose learners that use different knowledge representations: ZeroR, C4.5, IBk, Part, Naïve Bayes, SMO with polynomial kernels of order 3, SMO with Gaussian kernels, GAssist, and UCS. The systems were configured as recommended in the open source implementation, with exception of the following aspects. We ran IBk with

8.6. COMPARISON OF FUZZY-UCS TO SEVERAL MACHINE LEARNING TECHNIQUES

Table 8.13: Size of the models evolved by Fuzzy GP, Fuzzy GAP Fuzzy SAP, Fuzzy AdaBoost (ABoost), Fuzzy LogitBoost (LBoost), Fuzzy MaxLogitBoost (MLBoost), and Fuzzy UCS with weighted average inference (wavg), action winner inference (awin), and fittest rules inference (nfit).

	GP			GAP			SAP			ABoost	LBoost	MLBoost	Fuzzy-UCS		
	and	or	is	and	or	is	and	or	is				wavg	awin	nfit
<i>ann</i>	30.0	34.4	64.3	27.4	31.4	58.8	5.0	6.8	11.8	-	17.9	-	1038.6	27.2	12.7
<i>aut</i>	27.3	31.8	59.1	30.3	35.5	65.8	5.3	6.9	12.1	-	39.2	-	1555.7	45.1	28.7
<i>bal</i>	27.5	32.8	60.3	21.0	20.2	41.1	4.1	4.5	8.6	19.8	23.8	14.3	578.0	54.3	39.0
<i>bpa</i>	17.6	37.3	54.9	17.9	25.2	43.1	1.8	2.9	4.6	35.3	36.0	13.3	795.5	40.0	19.9
<i>cmc</i>	22.2	25.1	47.2	17.6	18.3	35.9	4.8	3.5	8.3	29.1	27.9	2.0	984.6	223.1	135.8
<i>col</i>	14.6	23.8	38.4	12.2	15.1	27.3	2.1	2.5	4.6	45.0	44.1	0.8	1469.3	50.8	26.0
<i>gls</i>	28.8	32.9	61.7	27.5	29.2	56.7	7.8	7.4	15.2	29.2	31.0	22.8	1293.7	27.8	14.5
<i>h-c</i>	13.6	20.0	33.6	11.4	13.5	24.9	1.7	2.6	4.3	39.6	38.9	1.0	1188.3	34.2	14.2
<i>h-s</i>	16.8	26.1	42.9	8.7	13.2	21.9	2.4	3.4	5.9	40.3	39.2	15.0	1173.0	37.2	18.8
<i>irs</i>	18.7	21.6	40.4	11.5	12.3	23.7	2.5	2.7	5.2	23.4	26.9	4.0	231.2	7.6	2.8
<i>pim</i>	18.3	19.9	38.2	13.7	14.6	28.3	2.0	1.7	3.7	36.9	34.0	13.3	1327.1	86.8	28.0
<i>son</i>	18.2	28.3	46.5	15.1	17.1	32.2	2.0	2.3	4.3	22.3	21.6	0.9	1208.1	70.7	63.4
<i>tao</i>	19.0	20.3	39.2	13.3	19.1	32.4	3.3	3.4	6.7	40.1	43.2	18.0	75.3	12.1	8.6
<i>thy</i>	18.2	20.0	38.1	12.4	13.0	25.4	2.8	2.4	5.1	25.7	29.4	8.8	624.4	16.3	5.0
<i>veh</i>	18.8	21.7	40.4	16.9	18.9	35.8	3.4	4.1	7.4	40.3	37.3	25.6	1641.7	143.8	63.7
<i>wbcd</i>	20.4	40.1	60.5	17.9	20.2	38.1	2.6	3.9	6.5	24.0	27.7	13.1	1033.9	38.9	8.4
<i>wdbc</i>	14.7	15.7	30.4	10.2	12.3	22.5	1.9	2.0	3.9	44.9	43.9	0.9	2108.7	105.4	38.4
<i>wne</i>	15.8	19.5	35.3	14.5	14.9	29.4	2.5	2.7	5.2	30.8	31.2	26.9	1437.7	33.1	9.0
<i>wpbc</i>	24.0	38.1	62.1	11.9	19.2	31.1	2.8	4.6	7.4	44.9	44.0	0.8	1536.6	62.3	45.3
<i>zoo</i>	34.0	34.9	68.9	37.8	38.2	76.0	8.2	9.7	17.9	42.0	36.2	0.9	263.4	5.0	3.3

Table 8.14: Pairwise comparisons of the sizes of the models of Fuzzy GP, Fuzzy GAP, Fuzzy SAP, Fuzzy AdaBoost (ABoost), Fuzzy LogitBoost (LBoost), Fuzzy MaxLogitBoost (MLBoost), and Fuzzy UCS with weighted average inference (wavg), action winner inference (awin), and fittest rules inference (nfit) by means of a Wilcoxon signed-ranks test.

	GP	GAP	SAP	ABoost	LBoost	MLBoost	Fuzzy-UCS		
							wavg	awin	nfit
GP		.0003	.0001	.0005	.0001	.0137	.0001	.0003	.2179
GAP	⊖		.0001	.0002	.0001	.1671	.0001	.0002	.0228
SAP	⊖	⊖		.0001	.0001	.0276	.0001	.0001	.0001
ABoost	⊕	⊕	⊕		.8666	.0001	.0001	.0793	.1790
LBoost	⊕	⊕	⊕	-		.0001	.0001	.1005	.1084
MLBoost	⊖	-	⊕	⊖	⊖		.0001	.0002	.0105
wavg	⊕	⊕	⊕	⊕	⊕	⊕		.0001	.0001
awin	⊕	⊕	⊕	+	+	⊕	⊖		.0001
nfit	=	⊕	⊕	-	-	⊕	⊖	⊖	

$k = \{1, 3, 5\}$. We ranked the performance obtained by the three configurations, and we only provide the results with the settings that maximized the average rank, that is, $k = 5$ (IB5). The analogous process was carried out for SMO with polynomial kernels. We experimented with polynomial kernels of order $\{1, 3, 5, 10\}$, and supplied the results obtained with polynomial kernels of order 3 since they maximized the average rank. We did not introduce the same system with different configurations in the comparison to avoid biasing the statistical analysis of the results.

Comparison of the performance. Table 8.15 shows the accuracy of the aforementioned learners on the same collection of real-world problems. The two last rows of the table provide the average rank and the position in the ranking of each learner. As proceeds, we discuss several observations that can be drawn from these results.

Firstly, let us highlight the good performance presented by Fuzzy-UCS with weighted average inference. This learner was the third best method in the ranking. Its average rank was really close to UCS, by which Fuzzy-UCS was inspired. Thus, the fuzzy representation did seem not to limit the capabilities of Fuzzy-UCS if all the evolved rules were used to infer the class of new examples. Moreover, the average rank was also close to the best ranked method: SMO with polynomial kernels. The other two inference schemes presented higher average ranks. Fuzzy-UCS with action winner inference and fittest rules inference occupied the 7th and 9th position in the ranking.

We statistically analyzed the results to identify significant differences among the learners. The multiple-comparison Friedman's test rejected the hypothesis that all the learners performed the same on average at a significance level of 0.001. We applied the post-hoc Bonferroni-Dunn test on the results. The test could only reject the hypothesis that the best ranked learners performed equivalently to Fuzzy-UCS with fittest rules inference, SMO with Gaussian kernel, and Zero-R. However, the test has a low discriminatory power for a large number of learners (Demšar, 2006). Thus, we also compared the performance of each pair of learners by means of a Wilcoxon signed-ranks test (see table 8.16). Figure 8.17 uses a graph to illustrate the significant differences between the learners. The test confirmed that Fuzzy-UCS with weighted average inference was one of the best learners in the comparison. It significantly outperformed Naïve Bayes, SMO with Gaussian kernels, ZeroR, and Fuzzy-UCS with the other two types of inference. Moreover, Fuzzy-UCS with weighted average inference did not significantly degrade the results obtained with any other learner. Fuzzy-UCS with action winner inference was only significantly outperformed by SMO with polynomial kernels, and Fuzzy-UCS with weighted average inference. Besides, it significantly improved SMO with Gaussian kernel and ZeroR. Fuzzy-UCS with fittest rules inference presented the poorest results among the three configurations of Fuzzy-UCS. It significantly degraded the results obtained by SMO with polynomial kernels, UCS, IB5, Part, and Fuzzy-UCS with weighted average inference. However, note that it performed equivalently to well-known algorithms such as C4.5, Naïve Bayes, and GAssist.

Comparison of the interpretability. Herein, we qualitatively compare the interpretability of the models created by the different learners. We do not consider IBk, SMO, and Naïve Bayes since their knowledge representation can hardly be compared to the other learners. IBk is a lazy classifier that does not use any knowledge representation; to predict the output of a new input example, IBk searches for the k nearest neighbors and returns the majority class among them. SMO represents the knowledge by $\binom{n_c}{2}$ support vector machines (where n_c is the number

Table 8.15: Comparison of the test accuracy of Fuzzy-UCS with weighted average (wavg), action winner (Awin), and fittest rules inference (nft) to ZeroR (0R), C4.5, IB5, Part, Naïve Bayes (NB), SMO with polynomial kernels of order 3 (SMO_{p3}), SMO with Gaussian kernels (SMO_{rbf}), and GAssist.

	Fuzzy-UCS											
	0R	C4.5	IB5	Part	NB	SMO _{p3}	SMO _{rbf}	GAssist	UCS	wavg	Awin	nft
<i>ann</i>	76.20	98.90	97.34	98.57	86.33	99.34	91.90	97.88	99.05	98.85	97.39	98.61
<i>aut</i>	32.63	80.94	64.03	74.41	58.79	78.09	45.55	68.63	77.41	74.42	67.42	69.32
<i>bal</i>	45.46	77.42	88.18	82.86	90.57	91.20	88.30	79.57	77.32	88.65	84.40	83.40
<i>bpa</i>	57.99	62.31	58.85	67.56	55.97	59.97	57.99	62.24	67.59	59.82	59.42	58.93
<i>cmc</i>	42.70	52.62	46.51	50.04	50.65	48.75	42.70	53.58	50.27	51.72	49.67	49.42
<i>col</i>	63.06	85.32	81.49	84.51	78.23	75.59	82.41	94.30	96.26	85.01	82.46	78.50
<i>gls</i>	35.65	66.15	64.68	66.62	48.95	66.15	35.65	65.06	70.04	60.65	57.21	57.43
<i>h-c</i>	54.45	78.45	83.16	74.20	82.80	78.59	82.48	80.09	79.72	84.39	82.62	82.05
<i>h-s</i>	55.56	79.26	80.74	80.00	83.33	78.89	82.59	77.67	74.63	81.33	80.78	78.11
<i>irs</i>	33.33	94.00	96.00	94.00	96.00	92.67	93.33	96.20	95.40	95.67	95.47	93.73
<i>pim</i>	65.11	74.23	73.32	74.88	75.80	76.70	65.11	73.76	74.61	74.88	74.11	74.32
<i>son</i>	53.38	71.07	84.05	74.38	69.71	85.52	69.26	75.81	76.49	80.78	73.71	71.66
<i>tao</i>	49.89	95.92	97.14	94.33	80.98	84.22	83.63	91.59	87.00	81.71	83.02	87.53
<i>thy</i>	69.83	94.91	94.85	94.33	97.16	88.91	69.83	92.52	95.13	88.18	89.49	91.25
<i>veh</i>	25.42	71.14	68.91	73.39	46.28	83.30	41.71	67.00	71.40	67.68	65.35	65.34
<i>wbed</i>	65.52	94.99	97.14	95.71	96.15	96.42	96.13	95.59	96.28	96.01	95.73	95.29
<i>wdbc</i>	63.11	94.40	96.78	94.46	93.13	97.58	92.88	94.24	95.96	95.20	94.61	94.51
<i>wme</i>	39.93	93.89	96.67	93.30	97.19	97.75	39.93	93.19	96.13	94.12	94.86	91.82
<i>wdbc</i>	72.97	71.61	78.85	70.05	69.45	81.25	72.97	72.33	69.40	76.06	76.05	71.69
<i>zoo</i>	41.89	92.81	90.47	93.81	94.47	97.83	76.03	93.97	96.78	96.50	94.78	95.90
Rank	<i>11.50</i>	<i>5.95</i>	<i>5.28</i>	<i>5.95</i>	<i>6.63</i>	<i>4.28</i>	<i>8.95</i>	<i>6.25</i>	<i>4.65</i>	<i>4.68</i>	<i>6.50</i>	<i>7.40</i>
Pos	<i>12</i>	<i>5.5</i>	<i>4</i>	<i>5.5</i>	<i>9</i>	<i>1</i>	<i>11</i>	<i>7</i>	<i>2</i>	<i>3</i>	<i>8</i>	<i>10</i>

Table 8.16: Pairwise comparison of the test accuracy of Fuzzy-UCS with weighted average (wavg), action winner (Awin), and fittest rules inference (nft) to ZeroR (OR), C4.5, IB5, Part, Naive Bayes (NB), SMO with polynomial kernels of order 3 (SMO_{p3}), SMO with Gaussian kernels (SMO_{rbf}), and GAssist by means of a Wilcoxon signed-ranks test.

	OR	C4.5	IB5	Part	NB	SMO _{p3}	SMO _{rbf}	GAssist	UCS	Fuzzy-UCS		
										wavg	awin	nft
OR	⊕	.0001	.0001	.0001	.0001	.0001	.0010	.0001	.0001	.0001	.0001	.0001
C4.5	⊕		.7089	.9039	.2627	.4209	.0072	.9405	.2043	.7938	.3905	.0793
IB5	⊕	=		.7938	.0534	.4115	.0004	.4553	.8519	.8228	.1084	.0304
Part	⊕	+	-		.2471	.2959	.0057	.4330	.2180	.4209	.3135	.0251
NB	⊕	-	-	-		.0674	.0333	.1084	.1354	.0333	.1790	.3703
SMO _{p3}	⊕	+	+	+	+		.0032	.2959	.6542	.1672	.0400	.0366
SMO _{rbf}	⊕	⊖	⊖	⊖	⊖	⊖		.0032	.0064	.0004	.0025	.0152
GAssist	⊕	-	-	-	+	-	⊕		.2180	.5016	.3135	.0859
UCS	⊕	+	=	+	+	-	⊕	+		.4330	.0674	.0366
wavg	⊕	+	=	+	⊕	-	⊕	+	-		.0032	.0051
awin	⊕	=	-	-	+	⊖	⊕	-	-	⊖		.2627
nft	⊕	-	⊖	⊖	+	⊖	⊕	-	⊖	⊖	-	

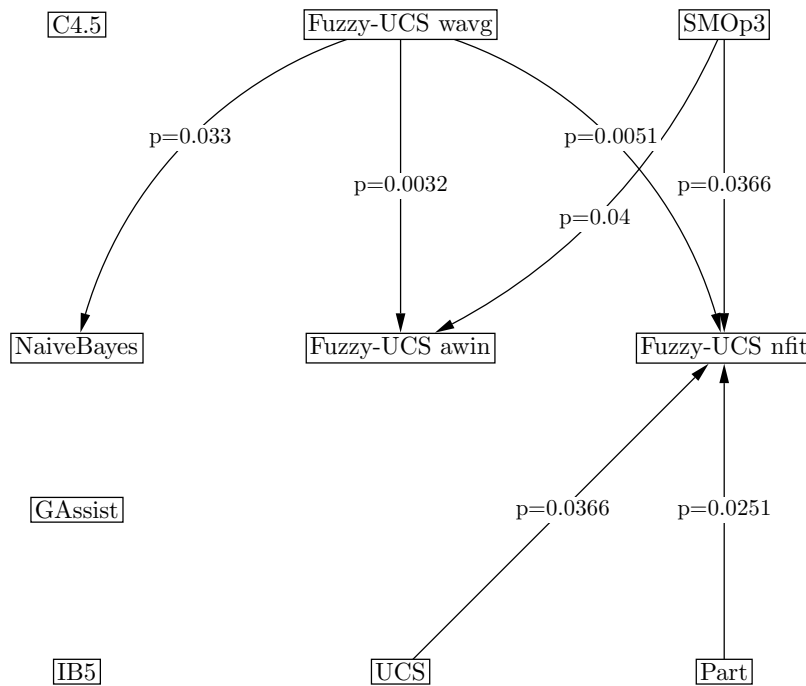


Figure 8.17: Illustration of the significant differences (at $\alpha = 0.05$) of the test accuracy among non-fuzzy methods and Fuzzy-UCS. An edge $L_1 \xrightarrow{pvalue} L_2$ indicates that the learner L_1 outperforms the learner L_2 with the corresponding $pvalue$. To facilitate the visualization, ZeroR and SMO with Gaussian kernels, the two most outperformed algorithms, were not included in the graph.

of classes), each one consisting of a set of real-valued weights. Therefore, the models created by these two learners are very difficult to interpret. On the other hand, Naïve Bayes builds interpretable models formed by a set of parameters which estimate the independent probability functions and the so-called class-prior of a Bayesian model. Nurnberger et al. (1999) identified a close connection between Naïve Bayes and *neuro-fuzzy classifier systems*, providing a framework that maps a Naïve Bayes classifier into a neuro-fuzzy classifier with the aim of improving its capabilities. The discussion on the difference in the interpretability of Naïve Bayes and their similarity to neuro-fuzzy classifier systems or fuzzy rule-based systems is out of the scope of this chapter. The reader is referred to (Nurnberger et al., 1999) for further details.

Thus, in the remainder of this analysis, we focus on the comparison of the rule-based and tree-based learners, i.e., C4.5, Part, GAssist, UCS, and Fuzzy-UCS. Figure 8.6.3 plots examples of the models evolved by these learners for the two-dimensional tao problem; besides, an example of the weights created by SMO is also depicted. C4.5 evolves trees in which the nodes represent a decision over one variable (see figure 8.18(b)). We evaluated the model size by counting the number of leaves of the tree. Part and GAssist create a set of rules which are defined by conjunction of conditions over their variables, and are interpreted as an ordered activation list (see figures 8.18(c) and 8.18(d)). Moreover, GAssist uses a default rule. UCS evolves a rule set similar to Fuzzy-UCS, but replacing linguistic rules by interval-based rules (see figure 8.18(e)).

Table 8.17: Average sizes of the models build by C4.5, Part, GAssist, UCS and Fuzzy-UCS with weighted average (wavg), action winner (awin), and fittest rules inference (nfit).

	C4.5	Part	GAssist	UCS	Fuzzy-UCS		
					wavg	awin	nfit
<i>ann</i>	38	15	5	4494	2769	75	36
<i>aut</i>	44	21	7	4565	3872	114	74
<i>bal</i>	45	37	8	2177	1212	114	75
<i>bpa</i>	25	9	6	2961	1440	73	39
<i>cmc</i>	162	168	15	3634	1881	430	271
<i>col</i>	5	9	5	3486	4135	154	81
<i>gls</i>	24	15	5	3359	2799	62	36
<i>h-c</i>	29	21	6	2977	3574	113	46
<i>h-s</i>	17	18	5	3735	3415	117	62
<i>irs</i>	5	4	3	1039	480	18	7
<i>pim</i>	19	7	7	3605	2841	192	62
<i>son</i>	14	8	5	520	3042	178	160
<i>tao</i>	36	17	6	807	111	19	14
<i>thy</i>	8	4	4	1994	1283	37	11
<i>veh</i>	69	32	7	4941	3732	332	147
<i>wbcd</i>	12	10	3	2334	3130	138	28
<i>wdbc</i>	11	7	4	5206	5412	276	101
<i>wne</i>	5	5	3	3685	3686	95	26
<i>wpbc</i>	12	7	4	5299	3772	156	115
<i>zoo</i>	11	8	6	1291	773	16	10

test (at $\alpha = 0.05$). Thus, Fuzzy-UCS achieved one of the main objectives of this work: to create smaller models than those evolved by UCS. Notice that, in addition of evolving smaller rule sets, the individual rules are also more interpretable since the variables are defined by linguistic terms instead of intervals.

- Fuzzy-UCS was the only method in the comparison in which the same semantics (adapted to the universe of discourse of each variable) is shared among all variables, and only 5 linguistic terms were specified. Consequently, Fuzzy-UCS rules were more readable.

The results also indicate that, even Fuzzy-UCS with action winner and fittest rules inferences resulted in moderate-sized populations, the system is still not competitive, in terms of interpretability, with Part, C4.5, and especially with GAssist. However, two important distinctions need to be considered to justify these differences:

- Fuzzy-UCS and, in general, Michigan-style LCSs evolve rules that, by themselves, are experts on the region of the feature space that they cover and collaborate to classify all the input space. Thus, each rule can be regarded as an expert classifier; if the human expert is only interested in a region of the feature space, only the rules involved in this

region need to be considered. On the other hand, the rules evolved by Part and GAssist form an ordered activation list. That is, to classify a new example, rules are checked in order and the first rule that matches determines the output. In the case of GAssist, a default rule is used to classify all the examples not matched by any rule in the activation list. This implies that all the previous rules need to be considered to understand why the system is giving this prediction.

- Fuzzy-UCS evolves the rule set incrementally, whilst the other learners go through the data several times to build a model of the data. Incremental learning gives a big advantage to Fuzzy-UCS when learning from large data sets.

The analysis supplied in this section showed that Fuzzy-UCS is highly competitive with respect to a large set of general-purpose machine learning techniques, which include several of the most influential machine learning techniques (Wu et al., 2007). The proposed weighted average version of Fuzzy-UCS was one of the best performers. Thus, a fuzzy rule-based system could achieve accuracy rates as good as—or even better than—other machine learning techniques with knowledge representations that can barely be read by human experts such as support vector machines or instance based algorithms. Moreover, Fuzzy-UCS with the two other inference schemes appeared also to be competitive. Fuzzy-UCS with action winner inference evolved substantially reduced rule sets, although not as much as the ones evolved by GAssist and Part, and it was only statistically surpassed by SMO with polynomial kernels, and our Fuzzy-UCS with weighed average inference. In the next section, we explore the capabilities of the online learning architecture of Fuzzy-UCS to learn from large volumes of data.

8.7 Fuzzy-UCS for Mining Large Data Sets

The two essential differences between Fuzzy-UCS and other rule-based machine learning techniques are that Fuzzy-UCS a) does not perform any form of global optimization, and b) evolves the rule-based knowledge online. Based on a rule set roughly initialized in the first learning iterations by the covering operator, the system is responsible for incrementally evaluating the parameters of the rules and refining the rule-based knowledge by creating more general and more accurate rules. This process provides two main advantages with respect to other learners:

- Fuzzy-UCS learns from a stream of examples. This enables the system to learn from changing environments. This differs from other machine learning methods, such as C4.5, IBk, SMO, and Pittsburgh-style LCSs, which need to process all the training data set in order to produce the final model.
- The learning can be stalled whenever required, and the evolved rule set can be used for predicting the class of new input examples. The more learning iterations the system has performed, the more general and accurate the rules should be. Consequently, the cost of the algorithm increases linearly with the maximum population size N , the number of variables per rule a , and the number of learning iterations n_{learn}

$$Cost_{Fuzzy-UCS} = O(a \cdot N \cdot n_{learn}), \quad (8.37)$$

but it does not depend directly on the number of examples. In static data sets, it is recommended that n_{learn} be, at least, the number of examples in the training data set.

Table 8.18: Properties of the 1999 KDD Cup intrusion detection data set. The columns describe: the identifier of the data set (Id.), the number of instances (#Inst), the total number of features (#Fea), the number of real features (#Re), the number of nominal features (#No), the number of classes (#Cl), the proportion of instances with missing values (%MisInst), and the dispersion of the data set (Disp) computed as #Fea/#Inst.

Id.	#Inst	#Fea	#Re	#No	#Cl	%MisInst	%Disp
<i>kdd'99</i>	494,022	41	35	6	23	0	$8.3 \cdot 10^{-5}$

In this section, we exploit the benefits of online learning in Fuzzy-UCS and apply the system to mine very large data sets. Specifically, we test the performance of Fuzzy-UCS on the 1999 KDD Cup intrusion detection data set (Hettich and Bay, 1999). In the following, we describe the data set and present the experimental results.

8.7.1 Data Set Description

The 1999 KDD Cup intrusion detection data set gathers a large collection of examples of a wide variety of network intrusions simulated in a military environment. We used the subset of 494 022 examples provided in (Hettich and Bay, 1999) that advocate 23 different classes. Each example consists of 41 attributes, which usually characterize network traffic behavior. Table 8.18 summarizes the main traits of the data set.

8.7.2 Results

We ran Fuzzy-UCS on the KDD'99 domain with the default configuration as in the previous section except for $\theta_{GA} = 200$ and $P_{\#} = 0.2$. We increased the period of GA application ($\theta_{GA} = 200$) to permit the classifiers to receive more parameter updates before undergoing a genetic event. We also diminished the probability of generalization in covering ($P_{\#} = 0.2$) since the number of examples per dimension was very high. We ran the experiment for 2 000 000 learning iterations, so that Fuzzy-UCS only received each learning instance an average of 4 times. As in all the experiments performed in this work, to obtain reliable estimates, we employed a 10-fold cross validation methodology (Dietterich, 1998).

Figure 8.19 plots the evolution of the test performance and the population size of Fuzzy-UCS with action winner inference in the first 50 000 learning iterations. That is, we stopped the learning every 2 500 iterations and tested the rule set with the test fold; each dot in the plot corresponds to one of these measurements. Note that the system quickly evolved a highly accurate population. After seeing the first 35 000 examples, that is, a 7% of the whole training data set, the test performance was already about 99%. Increasing the number of learning iterations did not significantly improve the average performance, but it did create more general and equally accurate classifiers. This behavior can be observed in table 8.19, which depicts the test accuracy and the rule set size obtained by Fuzzy-UCS with weighted average inference (1st column), action winner inference (2nd column), and fittest rules inference (3rd column) at different learning iterations. That is, every 500 000 learning iterations, we used the corresponding

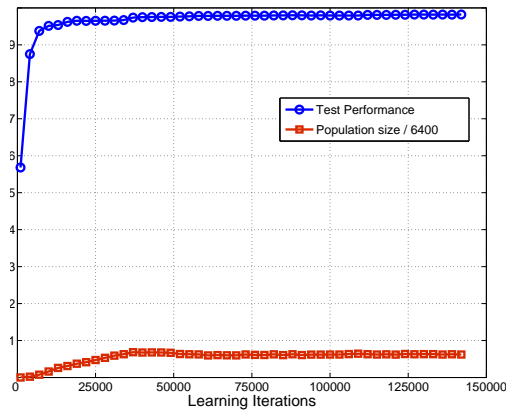


Figure 8.19: Evolution of test accuracies and the population size of Fuzzy-UCS with action winner inference in first 50 000 learning iterations of the 1999 KDD Cup data set.

Table 8.19: Test performance and number of rules evolved by Fuzzy-UCS with weighted average (wavg), action winner (awin), and fittest rules (nfit) in the 1999 Kdd Cup intrusion detection data set at different number of learning iterations.

#Iter	wavg		awin		nfit	
	perf.	#rules	perf.	#rules	perf.	#rules
500 000	99.32	1944	99.13	541	99.27	417
1 000 000	99.36	2089	99.07	492	99.25	369
1 500 000	99.37	2178	99.02	460	99.24	350
2 000 000	99.36	2257	99.00	428	99.19	323

test set to calculate the accuracy with the three types of inference. While sampling the test examples, all the learning mechanisms of Fuzzy-UCS were disabled, so that the rule set was not modified.

The results show that the number of rules in the final population for the action winner and fittest rules inference decreased as the number of learning iterations increased. Thus, the system was pushing the population toward obtaining maximally general and accurate rules. This behavior was not so clear with the weighted average inference since this inference scheme used all the experienced rules in the final population that have positive fitness, regardless of their generality.

Finally, let us highlight the differences of Fuzzy-UCS with respect to a Pittsburgh-style LCS. In the last section, we configured GAssist with a population of 400 individuals. At the initialization phase, these 400 individuals needed to be evaluated. Thus, the condition of all the rules of each classifier was matched with each of the training instances. This means that, in the population initialization, a Pittsburgh-style LCSs would go through all the data set 400 times, seeing about 180 000 000 instances. The use of windowing mechanisms, as the ones implemented

in GAssist, would permit reducing the number of instances that each individual matches to be evaluated by a constant value; nevertheless, note that, in any case, the number of evaluations would increase linearly with the number of input examples. After this, the system would have only the first approximation, and the evolutionary pressures would create new individuals that needed to be evaluated. This makes these types of systems computationally expensive for large data sets. On the other hand, note that Fuzzy-UCS only needed to see 35 000 examples to extract a highly accurate model, and that further iterations were to create a more general rule set. These results emphasize the advantages of online learning, which will be further exploited in future work.

8.8 Summary, Conclusions, and Further Work

The comparative analysis performed throughout this paper has provided many valuable insights on the behavior of Fuzzy-UCS in real-world data sets. As proceeds, we briefly summarize the work. Later, we provide the main conclusions and future work that will be made in light of the promising results supplied in this section. For this purpose, and with the aim of clearly identifying where Fuzzy-UCS is placed in the machine learning community, we perform a SWOT analysis, identifying the strengths, weaknesses, opportunities, and threats of Fuzzy-UCS.

8.8.1 Summary

In this chapter, we proposed a Michigan-style online learning fuzzy-classifier system for supervised learning which iteratively evolves a set of linguistic fuzzy rules which collaborate to cover all the input space. Three schemes of inference and reduction algorithms were designed to infer the output of unknown examples from reduced rule sets. These three mechanisms were designed to offer different levels of rule set reduction and consequently lead to different accuracy rates.

We performed a detailed analysis of the performance and interpretability of the rule sets evolved by Fuzzy-UCS. First, we carefully analyzed the three inference and reduction mechanisms in Fuzzy-UCS with a linguistic representation, and studied if Fuzzy-UCS could generally benefit from the flexibility provided by an approximate representation. This analysis showed that the approximate representation resulted in models that fit the decision boundaries of the problem more accurately. However, it was also detected that this may result in data over-fitting in some real-world domains. In addition, the approximate representation implied a loss of interpretability of the fuzzy rules. All these results supported the use of a linguistic representation.

We also compared Fuzzy-UCS to six fuzzy-rule-based learners and nine general-purpose learners with different types of representations. The analysis showed that Fuzzy-UCS was highly competitive with both groups of learners. In comparison with the fuzzy learners, Fuzzy-UCS with the three types of inference presented the best performances in the study; furthermore, it evolved rule sets with a moderate size. In comparison with the general-purpose machine learning techniques, Fuzzy-UCS with weighted average inference was ranked among the best learners. This showed the suitability of Fuzzy-UCS in spite of the limitations imposed by the discretization of the feature space produced by the linguistic representation. The many benefits of the online fuzzy-rule-based architecture, as well as some drawbacks detected in this study, are detailed in the SWOT analysis of the next section.

Table 8.20: SWOT analysis of Fuzzy-UCS.

Strengths	Weaknesses
<ul style="list-style-type: none"> - It evolves highly accurate models; comparable with the state-of-the-art in classification - It uses a highly legible knowledge representation based on linguistic fuzzy rules - It performs incremental, on-line learning - It is capable of mining large data sets 	<ul style="list-style-type: none"> - It generates moderate or large sized rule sets (depending on the chosen configuration) - Although it can deal with real, integer or categorical features, only application for real and integer data types is recommended
Opportunities	Threats
<ul style="list-style-type: none"> - It may be applied in data streams; further analysis of this problem will be carried out - Because of the use of fuzzy logic, the algorithm could be adapted to deal with vague and uncertain data - The proposed seminar system opens opportunities for further research on fuzzy knowledge representation, the fuzzy inference engine, and evolutionary operators 	<ul style="list-style-type: none"> - A small number of interval-based rules can be interpreted more easily than a large number of linguistic fuzzy rules - Other learning approaches combined with preprocessing can also deal with large data sets

In the final step of the analysis, we exploited the incremental learning architecture of Fuzzy-UCS to extract a model from a large data set: the 1999 KDD Cup intrusion detection data set. It was found that Fuzzy-UCS could quickly evolve a highly accurate model, having seen only the ten percent of the total number of examples in the domain. Incremental learning enabled us to have a rough approximation of the model after a few thousand learning iterations, further refining the rule set as the system received more examples.

8.8.2 SWOT Analysis

All the evidence provided through the experimentations is summarized in the SWOT analysis presented in table 8.20, where *strengths* represent the main advantages of Fuzzy-UCS, *weaknesses* show its drawbacks, *opportunities* outline some suggested further lines of investigation, and *threats* include some optional approaches considered by other methods that could compete with our proposal.

Fuzzy-UCS has four main strengths. First, the system presented high accuracy, which supports the use of Fuzzy-UCS in complex problems. Second, it uses linguistic fuzzy rules, which are much more readable than interval-based rules since all the variables share the same semantics and only a small number of linguistic terms per variable are defined (specifically, in our experi-

ments we only used five linguistic terms per variable). This is really important for domains with high dimensionality where each variable presents different ranges. Third, Fuzzy-UCS is an online process that performs incremental learning, and so, the system neither has knowledge about the data set nor does any kind of global optimization. And fourth, since the run-time complexity of Fuzzy-UCS does not depend on the number of instances in the data set, our system is very useful for mining large amounts of data as we showed with the KDD'99 problem, which consists of about half a million instances, 41 features, and 23 classes.

The main weakness of the system is that, despite the application of reduction schemes, Fuzzy-UCS evolves slightly larger rule sets than those created by other machine learning techniques such as GAssist. Consequently, the number of rules may hinder the interpretability of the evolved knowledge. However, as discussed in previous sections, it is worth highlighting the key differences between the types of rules build by GAssist and Fuzzy-UCS. That is, GAssist does not share any semantics between variables, makes the rules available in an ordered decision list—therefore, each rule depends on the previous rules in the list—, and uses a default rule. Conversely, Fuzzy-UCS evolves independent classifiers that do not depend on the context, and no pressure is applied to evolve default rules. A less important feature of our system is that, although it can work with categorical input variables, fuzzy rules are especially useful for real or integer-valued variables, since in the former case the rule would be equivalent to a classical crisp (or non-fuzzy) one.

We also want to honestly mention some possible threats to Fuzzy-UCS. On the one hand, an expert might find a small number of interval-based rules more legible than many linguistic fuzzy rules (in fact, the degree of interpretability of a system is very difficult to assess when different knowledge representations are compared). On the other hand, there are hybrid learning approaches to deal with problems with large data sets, such as the inclusion of data set reduction techniques, which would allow some of the systems compared in this chapter to address these problems.

Finally, the proposed Fuzzy-UCS algorithm shows some interesting opportunities which will be developed in future work. Firstly, because of its incremental learning capability, the system can be applied to extract information from data streams, which is currently a topic of increasing interest (Arasu et al., 2003; Aggarwal, 2007). Furthermore, the use of fuzzy logics allows the system to be adapted for managing vague and uncertain data, very common in many real-world problems (Sánchez and Couso, 2007; Sánchez et al., 2007). Finally, as future work, we can consider the inclusion of some of the techniques proposed by other systems (such as inference based on an activation list with default rule as in GAssist) and the design of new techniques to achieve greater reductions of the fuzzy rule set without a significant loss of test performance, as well as a more detailed research of other fuzzy knowledge representations.

Chapter 9

Summary, Conclusions, and Further Work

This thesis has investigated *learning classifier systems* as competitive methods for machine learning, addressing two important challenges not only for LCSs, but also for different machine learning techniques: extracting key knowledge from rare classes and building models that are comprehensible to human experts. To address the first challenge, we started a journey departing from a decomposition of the key elements that we would require for any LCS to detect rare classes and finishing with the improvement and application of LCSs to real-world domains with rare classes. To approach the second challenge, we took an inventiveness approach to hybridize the best practices of LCSs, GAs, and fuzzy systems; this resulted in the first Michigan-style learning fuzzy-classifier system for classification tasks.

In this chapter, we first summarize the results and critical observations provided along the consecution of each objective. Thereafter, in addition to the particular conclusions derived and contributions provided under each case, we abstract the work and make an effort to highlight the lessons learned on the way. Finally, we briefly discuss future work that will be made in light of the insights and results provided by this thesis.

9.1 Summary and Conclusions

The present work started with the identification of Michigan-style LCSs as mature machine learning techniques for which we had (1) competent implementations, (2) theory for design, and (3) applications in important domains. In addition, we discussed three main characteristics that make Michigan-style LCS an appealing alternative for machine learning. That is, Michigan-style LCSs (1) evolve a distributed solution in parallel, searching at different pace in different regions of the search space, (2) create a set of independent classifiers, and (3) learn the model online from a stream of examples. Among them, we highlighted the added value provided by their online learning architecture, which makes them suitable to deal with current scientific and industrial applications in which data are usually generated online and models need to be learned on the fly. With these potential advantages of Michigan-style LCSs in mind, we proposed to address the following two critical challenges in machine learning with LCSs:

1. Learning from domains that contain rare classes.
2. Building more understandable models and bringing reasoning mechanisms closer to human ones.

These challenges were studied in the context of two particular LCSs: XCS and UCS. We selected XCS since it is, by far, the most influential Michigan-style LCS. We also included UCS as this thesis was especially concerned with classification problems, and UCS is a specialization of XCS for supervised learning. The inclusion of the two LCSs set the first objective of this work. That is, UCS was identified as a young system that had received no research since its initial design in 2003, and some open issues such as the suitability of the fitness computation scheme, which did not share resources, remained unaddressed. In addition, the advantages of the new architecture of UCS with respect to XCS in classification problems needed further investigation. For this reason, we first proposed to update UCS and compare it with XCS on a set of boundedly difficult classification problems. Then, taking XCS and the new version of UCS as starting point, we focused on the challenges of modeling rare classes and building more comprehensible classification models. More specifically, we articulated the following four objectives:

1. Revise and update UCS and compare it with XCS.
2. Analyze and improve LCSs for mining rarities.
3. Apply LCSs for extracting models from real-world classification problems with rarities.
4. Design and implement an LCS with fuzzy logic reasoning for supervised learning.

Below, we summarize the work done under each objective and provide the key conclusions extracted from each point.

Revise and update UCS and compare it with XCS. The first objective of this thesis proposed the design of a fitness-sharing scheme for UCS, since resource sharing schemes have been shown to provide benefits to both GAs and LCSs. Therefore, we followed a creative analysis to design a fitness-sharing scheme for UCS based on those of XCS and other Michigan-style LCSs. With the introduction of the fitness-sharing scheme, the parameter update procedure of UCS was slightly modified. To evaluate whether the new scheme was beneficial to UCS, we compared the original UCS with UCS with fitness sharing on a collection of four boundedly difficult problems. Later, XCS was included in the comparison with the aim of highlighting the differences between both LCSs in classification tasks.

The empirical analysis provided several insights into the advantages of fitness sharing and into the differences between XCS and UCS. On the one hand, fitness sharing, while not being prejudicial in any problem, appeared to be crucial in domains where there was a tendency to create over-general classifiers. In these problems, the new fitness-sharing scheme enabled UCS to make a stronger pressure toward the removal of over-general classifiers when the first accurate classifiers were discovered. On the other hand, the analysis also made evident that the specialized architecture of UCS enabled the system to solve the tested problems with less computational resources than those required by XCS. And finally, as UCS computes classifiers' accuracy as the true proportion of correct predictions of the classifier instead of as the accuracy

of the payoff prediction, the system produced a more reliable fitness pressure toward the optimal population than XCS. These results were expected as UCS is specialized for supervised learning tasks, whereas XCS is a more general architecture that can be applied to reinforcement learning. Thence, the experimental results confirmed that the architecture of UCS is better suited than the XCS's one for classification problems.

Analyze and improve LCSs for mining rarities. With XCS and the updated version of UCS, we faced the problem of mining imbalanced domains with LCSs. Instead of developing new approaches that may improve the system performance in particular domains—but provide little understanding of the real problems that may affect LCSs—we took a more general approach. We abstracted the problem and considered any Michigan-style LCS as a system that evolves a distributed collection of sub-solutions—or niches, which, in turn, contain classifiers—that are evaluated and created online. Thence, we considered the problem of having different distributed sub-solutions—some of which are activated with a lower frequency—that compete for the environmental resources. Then, we followed a design decomposition approach and defined five elements, concerning the creation and evaluation procedures, that needed to be guaranteed to efficiently and scalably solve class-imbalanced problems.

The elements of the design decomposition were analyzed for the particular cases of XCS and UCS. Facetwise models were derived to explain the different elements, and critical conditions for convergence were detected for both systems. More specifically, the study included the following five points:

- The parameter update procedures were examined, providing key recommendations on how they should be configured to ensure an accurate estimation of the parameters of over-general classifiers.
- The capabilities of the covering operator to generate classifiers representing rare classes were analyzed.
- Population size bounds were derived to guarantee that both XCS and UCS could sustain niches that are infrequently activated.
- The effect of the period of activation of the GA on the preservation of infrequent solutions was studied.
- Takeover time expressions for proportionate and tournament selection were deduced, and critical conditions of niche extinction were developed in the same analysis.

In summary, for XCS, we theoretically demonstrated and empirically validated (1) that the parameter update procedure needs to be set inversely proportional to the imbalance ratio to ensure accurate estimates of classifier parameters¹ and (2) that either the population size or the period of application of the GA needs to increase linearly with the imbalance ratio to warrant that an accurate model can be extracted from the under-sampled class. Also, conditions where no convergence can be guaranteed were predicted by the niche extinction models. For UCS, the same conclusions could be extracted, with two key differences. First, the parameter update procedure of UCS was more robust than that of XCS and did not need any especial configuration to deal with domains with large imbalance ratios. Second, the theory developed

¹In particular, we showed that, with the Widrow-Hoff rule, β needs to decrease linearly with the imbalance ratio to ensure reliable estimates of over-general classifier parameters

for UCS predicted an upper bound on the system behavior. All these analyses resulted in key insights, which were articulated as configuration recommendations, enabling both XCS and UCS to solve highly imbalanced problems that previously eluded solution.

Overall, the main conclusion derived from this objective is that LCSs are competitive machine learning techniques that can effectively solve problems with rare classes, scaling linearly with the imbalance ratio. Therefore, this supports that LCSs are ready to tackle real-world problems with rarities.

Apply LCSs for extracting models from real-world classification problems with rarities.

Facetwise analysis provided key insights about LCSs behavior on domains with class imbalances, pointing toward several recommendations that need to be followed to learn accurate models from rare classes efficiently. These models proposed to configure both XCS and UCS according to the imbalance ratio of the training data set, which was assumed to map the imbalance ratio among starved and nourished niches. Nonetheless, this information is not available in real-world domains with unknown characteristics, which opens a gap between the recommendations provided by the theory and its application.

Therefore, under the current objective, we first started to bridge the gap between theory and application in real-world domains. For this purpose, we redefined the concepts of niche, representative classifier, and over-general classifier for domains with continuous attributes. This also led to the redefinition of the problem of mining rare classes. In the interval-based representation, we detected that the niche formation depended on the combination of the expressiveness of the knowledge representation and distribution of examples in the feature space. In brief, we showed that starved niches—or small disjuncts, as termed in the machine learning community—could appear in completely balanced data sets if the form defined by the conditions of the classifiers—in our case, hyper rectangles—did not match the shape of the class boundary. For example, the hyper rectangular representation required the creation of starved niches to accurately approximate oblique class boundaries, regardless of the imbalance ratio of the training data set. Thence, this observation evidenced the relevance and broadness of the problem with starved niches, which can appear in any real-world problem.

Therefore, we identified the new problem of estimating the ratio of the frequency of activation of nourished niches to the frequency of activation of starved niches to enable the application of the recommendations extracted from the theoretical analysis. For this purpose, we designed a heuristic procedure that automatically computes the niche imbalance ratio and self-adapts both XCS and UCS. Empirical results showed that the self-adaptation mechanisms enabled XCS and UCS to solve problems with rare classes without being informed of the actual imbalance ratio.

With the new heuristic procedure, XCS and UCS were ready to face real-world problems, self-adapting themselves according to the information gathered during the evolution. Then, the objective was to analyze whether the two LCSs were competitive with respect to other machine learning techniques in extracting classification models from domains with rare classes. For this purpose, we compared the accuracy of XCS's and UCS's models with those created by three of the most influential machine learning techniques: C4.5, SMO, and IBk. The empirical results showed that both LCSs provided the most accurate models on average. In addition, we extended the comparison by considering four of the most-competent re-sampling techniques: random over-sampling, under-sampling based on Tomek links, SMOTE, and cSMOTE. In

general, random over-sampling and SMOTE enabled the learners to create more accurate models of the minority class.

In addition to the observations pointed out in the experimental analysis, the whole comparison came with a crucial conclusion: the performance of each learning system—and each re-sampling technique if used—depended on each particular problem. That is, although general conclusions could be extracted, such as that XCS was the classifier that obtained the most accurate models in the majority of the problems, this does not guarantee that the best method on average will be the best performer for a new real-world, unknown problem. Actually, along the analysis, in several particular cases, the worst method of the comparison provided the most accurate models or vice versa, or re-sampling techniques that resulted in poorer results than those obtained with the original data sets were identified. Therefore, this conclusion demands further studying the characteristics that make real-world problems difficult to learn for each specific learner. This future line work is discussed in more detail in the following sections.

Design and implement an LCS with fuzzy logic reasoning for supervised learning. After analyzing and improving LCSs with the aim of evolving more accurate models of rare classes, the last objective of this work pointed toward increasing the interpretability of the models evolved by LCSs and using reasoning mechanisms that are similar to the human ones. Providing understandable models is a key aspect in supervised learning, especially in critical domains where human experts may require an explanation of the prediction to contrast it with their knowledge, thoughts, or beliefs. For example, in medical domains, human experts may require an explanation of a given diagnosis to see if it confirms or refutes their initial diagnosis.

While in the previous objectives we took an analytic approach to studying existing systems, we followed a creative analysis to achieve the last objective of the thesis. That is, we mixed the ideas that come from LCSs—which provide an accurate online evaluation system—, GAs—which represent a robust discovery component—and fuzzy logic—which supplies human-like representations and reasoning mechanisms. The combination of the three ideas was not novel itself, but the way in which the elements were combined was. The result was the first Michigan-style fuzzy-classifier system that evolves a linguistic fuzzy rule set online for classification tasks. It is worth noting that the system was not a result of a trial/error process, but was derived from a careful analysis of the different ways in which fuzzy logic could be introduced in both the representation and the reasoning mechanisms of UCS. The presented learning method resulted from few iterations on the initial design.

The system was provided with two important characteristics that came from the crossbreeding between LCSs and fuzzy logics, which prepare the system to deal with new challenging problems. Inherited from Michigan-style LCSs, Fuzzy-UCS consists of an online learning architecture; therefore, it can learn from data streams (Aggarwal, 2007; Gama and Gaber, 2007). Thence, the system can deal with applications in which a continuous flow of labeled data is generated, which is usual in many current industrial applications. On the other hand, due to the integration of fuzzy logic into the system, the evolved linguistic fuzzy rules are more legible, and the reasoning mechanisms are closer to human reasoning. Moreover, rule reduction mechanisms were designed for each fuzzy inference. Therefore, the system is prepared to deal, effectively and scalably, with problems with vague and uncertain data, which is very common in many real-world problems (Sánchez and Couso, 2007; Sánchez et al., 2007).

Fuzzy-UCS was extensively analyzed. As Fuzzy-UCS was applied to real-world problems with

unknown characteristics, we evaluated the quality of the evolved models by comparing them with the models created by several of the most influential machine learning techniques for pattern recognition. The experimental results showed that Fuzzy-UCS statistically outperformed all the fuzzy learners included in the comparison; also, Fuzzy-UCS was one of the best methods when compared to non-fuzzy learners. Besides, the models evolved by Fuzzy-UCS were, by far, more interpretable than the correspondent models in UCS. Finally, the advantages of the hybridization of LCSs and fuzzy logics was shown by using Fuzzy-UCS to solve a very large problem, the 1999 KDD Cup intrusion detection data set.

In general, the contributions of this work emphasize that Michigan-style LCSs represent a general-purpose, highly-competitive framework for the evolution and the discovery of independent classifiers online. The results provided along this work showed that this framework resulted in competent implementations, that is XCS and UCS, that can capture and accurately model rare classes on the fly. Furthermore, we also illustrated that LCSs easily permit the crossbreeding of ideas, enabling the integration of the best practices that come from several fields. All this makes LCSs one of the most appealing alternatives for facing new challenging applications, which usually require the discovery of knowledge from rarities, the integration of complex representations, and the combination of new techniques. For this reason, LCSs probably have much to say in the future of machine learning.

In addition to the conclusions extracted from the contributions of this work, the development of the present thesis also resulted in valuable lessons for facing new engineering and machine learning problems. The next section discusses these lessons in more detail.

9.2 Lessons from LCSs Design and Application

On our way toward the thesis objectives, two notably different strategies have been followed. While we employed an analytical approach to study LCSs in domains with rare classes, we took an inventiveness methodology and creative analysis to build a machine that mixed the ideas coming from the LCSs, the GAs, and the fuzzy logic fields. Here, we raise the analysis one notch and emphasize the general lessons learned from the application of the two approaches to solve the problems that were defined in the beginning of this document. More specifically, we want to emphasize the following two key lessons:

1. The importance of design decomposition.
2. The relevance of ideas crossbreeding for successfully dealing with complex applications.

These two lessons are further discussed in what follows.

The importance of design decomposition. When we started to approach the challenge of learning from imbalance domains with LCSs, we faced the problem of improving a complex existing learning architecture in a complicated problem. At that point of the thesis, we initially thought about the following three main methodologies to address the problem:

1. Follow intuition to design new modifications that may help discover rare classes.

2. Develop models of the whole system.
3. Decompose the problem in critical elements and analyze them separately.

The first two approaches provide two completely opposite ways of focusing the problem. Using intuition to develop new approaches may lead to some modifications of a particular system that may help get more accurate models of rare classes. Actually, this approach was initially followed in (Orriols-Puig and Bernadó-Mansilla, 2005a,b, 2007). Although this resulted in some improvements on LCSs behavior, these works could not explain the real problems that LCSs may need to face when learning from domains with rare classes. On the other end of the spectrum, we have the option of building global models of the system and then using these models to study the impact of rare classes. The main difficulty of this approach is that the models have to consider all the interactions among the system components. Thence, this may require a high cost to derive complex mathematical equations, especially in LCSs, where several components interact during learning. Moreover, the complexity of the equations themselves may hamper some key insights.

In this thesis, we took the third approach and followed a design decomposition methodology. That is, we sought an effective design decomposition for the problem of creating accurate models of rare classes with LCSs. This led us to a decomposition in five elements that focused on the problems that LCSs may have in discovering starved niches, and the study of each element permitted us to derive simple models that provided key insights into the system behavior and served as a design tool to improve LCSs' ability to solve highly imbalanced domains. The benefits of this approach have been soundly evidenced along the present work.

Therefore, the lesson extracted here is that design decomposition is a powerful tool not only for analyzing GAs and LCSs, but also for studying any complex system for which models of the whole system are too complex or costly. That is, design decomposition proposes a novel engineering methodology in which complex problems are analyzed by decomposing them into simpler subproblems and, with little algebraic effort, facetwise or "little" models are derived for each subproblem assuming that all the other facets behave in an ideal manner. The models of different facets provide key insights on the system behavior and can be used as a tool for designing new competent systems that satisfy the requirements identified by the models. Furthermore, this approach enables us to save efforts in computing more complex models that, sometimes, may hide some important insights in their complexity. This thesis has provided a good example of how "little" models can help us increase our understanding of complex systems, solving new challenging problems that previously eluded solution. Therefore, the results of this work provide another example of the power of design decomposition, which we hope help encourage other researchers to follow this approach—or even utilize some of the derived models—to analyze complex systems and apply LCSs to new challenging problems.

The relevance of ideas crossbreeding for successfully dealing with complex applications. The second important conclusion of the present work is the need of mixing ideas that come from different learning paradigms to tackle, scalably and efficiently, increasingly complex real-world problems and to build more robust, intelligent, and practical machine learning techniques. That is, current real-world applications usually consist of a large number of examples with complex structures and, sometimes, with vagueness and uncertainty. Most of the typical learning methods are not ready to deal with these types of data or, simply, are not scalable and so not applicable to large data sets. Therefore, this situation demands gathering the best

practices of different machine learning fields and building scalable learners that take the best characteristics of each area. Actually, fields such as *soft computing* were born with the aim of crossbreeding the ideas that come from different machine learning areas.

Fuzzy-UCS is an example of fruitful crossbreeding of LCSs, GAs, and fuzzy logics which resulted in a system that can evolve fuzzy rule sets online from a stream of data that may contain vagueness or uncertainty. In addition to the benefits shown in the application of Fuzzy-UCS across several problems, its design comes with the second key lesson of this thesis. That is, the design of Fuzzy-UCS makes evident that LCSs are a “*friendly*” framework for ideas crossbreeding, probably because they were initially designed as a general framework to create artificial intelligence itself. If we regard the general model proposed by the several Michigan-style LCSs studied in the present work, we can realize that LCSs propose a largely general learning model that permits easily incorporating new knowledge representations, apportionment of credit algorithms, or even new search procedures. Therefore, in the current machine learning era, where the field is constantly faced with increasingly challenging problems and ideas crossbreeding appear as the best way to tackle these problems, LCSs turn to be one of the most appealing methods to build next generation machine learning techniques.

All the results, the conclusions, and the lessons learned in this work have also served to fix new objectives that will be approached in further work. In the next section, we take a glance ahead and define two major future research lines that derive from this thesis.

9.3 Further Work

Along the consecution of the objectives of this thesis, several open issues that demand further research have been identified in the end of each chapter. Here, we discuss two important future work lines that come directly or indirectly from the work of this thesis. These two new goals that will guide our immediate future work are:

1. Design measures to characterize real-world classification problems and relate this characterization to the theory.
2. Adapt LCSs to extract association rules online.

The former, the need for a better characterization of real-world classification problems, was already emphasized in the comparison of several machine learning techniques performed in chapter 7. The later, the adaptation of LCSs to extract association rules online, comes motivated by the types of problems in industry that currently face machine learning techniques. The motivation for each of these two lines is elaborated in more detail in what follows.

Design measures to characterize real-world classification problems and relate this characterization to the theory. The study of the class-imbalance problem in LCSs indirectly evidenced the increased difficulty of dealing with real-world problems. That is, in the first stage of the study, we took an analytical approach, derived facetwise models, and validated these models with artificial problems in which we could control the different dimensions of problem difficulty. The experiments emphasized that, given the particular complexities of the artificial

problems, the theory could explain the behavior of LCSs on these problems. Nonetheless, the same analysis could not be directly applied to real-world problems. That is, the fact that the characteristics of real-world problems were unknown opened a gap between the theory and its application.

Therefore, to evaluate LCSs performance, we compared them with a collection of top-notch machine learning techniques and analyzed which learners resulted in the most accurate results on average. This type of comparison permitted us to extract conclusions on the average performance of the learners; however, it provided poor information about whether a given learner would be the best performer in a new real-world problem. This was an inevitable effect of having no information about the intrinsic complexities of the new problem. Therefore, in general, we could barely predict whether a given learner can efficiently solve a new problem. To do this, we need to (1) identify the sources of complexity that affect each particular learner and (2) create some methodology to characterize real-world problems. While the first types of analyses are common in machine learning (the facetwise analysis provided in this thesis is a clear example), the characterization of real-world problems is still a young field.

Consequently, the first future work line aims at advancing in the characterization of real-world problems. For supervised learning, [Ho and Basu \(2002\)](#) designed a collection of metrics that provide some indicators about the geometrical characteristics of the class distributions in the training data set. Some analyses that relate these metrics with the error of XCS have been already conducted ([Bernadó-Mansilla and Ho, 2005](#); [Bernadó-Mansilla et al., 2006](#)). In spite of these first successful analyses, these complexity metrics were not enough to fully capture all the sources of difficulty of classification problems. Thence, as further work, we will follow up these works, taking the defined metrics as starting point, with the aim of defining more metrics and using them to characterize real-world problems. This characterization would permit us to approach the theory to the actual difficulties of real-world problems.

Moreover, it appears appealing to use these metrics to enrich the study of different learning systems by comparing them on collections of data sets with a certain given complexity. That is, the characterization of learning domains would permit identifying the problem characteristics for which each learner is better suited than the others, thence, identifying the sweet spot where each learning algorithm is the best in the comparison. Furthermore, the definition of complexity metrics would enable us not only to assess the difficulty of existing data sets, but also to create new data sets with certain complexities to evaluate learning methods. The first steps toward this promising research area have been taken in ([Macià et al., 2008a,b,c](#)).

Adapt LCSs to extract association rules online. Under the fourth objective of this thesis, we designed Fuzzy-UCS, preparing the system to deal with streams of labeled data with vagueness and uncertainty. This resulted in a powerful tool that was shown to be able to mine large data sets online, which were made available as data streams. We already argued that the online learning architecture made Fuzzy-UCS an appealing alternative to extract classification models from data streams, which is increasingly demanded in industrial applications.

In addition, there are many industrial applications which generate streams of unlabeled data ([Aggarwal, 2007](#); [Gama and Gaber, 2007](#)). This defines an unsupervised learning problem, in which learning examples need to be processed on the fly to extract useful information. To solve these types of problems, we propose to use the same ideas of XCS, UCS, and Fuzzy-UCS and prepare LCSs to extract key information from these streams of data in form of association rules.

For this purpose, concepts of association rule mining would be incorporated to a system similar to Fuzzy-UCS, and the affected mechanisms of the system would be modified to deal with the new representation. Note that this future work line also puts emphasis on the crossbreeding of ideas to solve a new challenging problem in machine learning. That is, we propose to define an unsupervised learning technique by combining the best practices in LCSs, GAs, fuzzy logic, and association rule mining.

Recently, the first steps toward this direction have been taken. A Michigan-style LCS for association rule mining, which incorporates many of the mechanisms discussed in this thesis, was designed in (Orriols-Puig et al., 2008g). Later, another version of the same system, which included the fuzzy representation of Fuzzy-UCS, was used to model the consumer behavior in a web-based trust model (Orriols-Puig et al., 2008i,h). New applications as well as more analyses of the system behavior will be conducted as further work.

After all the conclusions and future work lines provided through all the chapters of this thesis and collected in the present and previous sections, little extra motivation can be given to highlight the many future lines of this work. To conclude with the thesis, I would only recall to the reader the innovation and creativity capabilities of evolution, which by means of basic genetic information (building blocks) mixing, random changes, and selection-of-the-fittest principle, has been able to provide accurate, adapted, and diverse solutions to life. Therefore, the power of competent GAs, LCSs, and GBML systems encourages their application to solve increasingly challenging problems that require a dose of innovation and creativity. Without GAs, many real-world problems would not have been solved. Surely, GAs and GBML have a lot to contribute in the following decades.

Appendix A

Description of the Artificial Problems

This appendix describes the artificial problems used along the chapters of this thesis. For each problem, we provide a concise description, specify and give a particular example of the best action map $[B]$ (Bernadó-Mansilla and Garrell, 2003) and the complete action map $[O]$ (Wilson, 1995; Kovacs and Kerber, 2001) of the problem, and describe the dimensions along which the problem difficulty can be moved.

A.1 Parity

The parity is a problem that has widely been used as a benchmark in LCS since it was originally introduced by Kovacs (2001) to show the dependence of XCS's performance on the optimal population size. The problem is defined as follows. Given a binary string of length ℓ , where there are k relevant bits ($0 < k \leq \ell$), the output is the number of one-valued bits in the k relevant bits modulo two. The complexity of the problem can be moved along the building block length, which is controlled with the parameter k . That is, larger values of k poses more complexity to the learner, which needs to discover larger building blocks, and so, larger populations, without

Table A.1: Best action map (first and second columns) and complete action map (all columns) of the parity problem with $\ell = k = 4$.

0000:0	1000:1	0000:1	1000:0
0001:1	1001:0	0001:0	1001:1
0010:1	1010:0	0010:0	1010:1
0011:0	1011:1	0011:1	1011:0
0100:1	1100:0	0100:0	1100:1
0101:0	1101:1	0101:1	1101:0
0110:0	1110:1	0110:1	1110:0
0111:1	1111:0	0111:0	1111:1

Table A.2: Best action map (first column) and complete action map (all columns) of the decoder problem with $\ell = k = 4$.

0000:0	###:0	#1##:0	##1#:0	###1:0
0001:1	###:1	#1##:1	##1#:1	###0:1
0010:2	###:2	#1##:2	##0#:2	###1:2
0011:3	###:3	#1##:3	##0#:3	###0:3
0100:4	###:4	#0##:4	##1#:4	###1:4
0101:5	###:5	#0##:5	##1#:5	###0:5
0110:6	###:6	#0##:6	##0#:6	###1:6
0111:7	###:7	#0##:7	##0#:7	###0:7
1000:8	###:8	#1##:8	##1#:8	###1:8
...
1111:15	###:15	#0##:15	##0#:15	###0:15

fitness guidance.

The best action map size consists of 2^ℓ rules, each one with all the k relevant bits specified and predicting the correct class. The complete action map doubles the best action map, as it also maintains specific rules predicting the wrong class; thence, the size of the complete action map is $||O|| = 2^{\ell+1}$. Table A.1 shows the best action map and the correct action map for a parity problem with $\ell = 4$ and $k = 4$.

A.2 Decoder

The decoder problem is an artificial problem with binary inputs and multiple classes. Given an input of length ℓ , where there are k relevant bits ($0 < k \leq \ell$), the output is determined by the decimal value of k relevant bits. Therefore, the k relevant bits form a building block. The number of classes increases exponentially with the condition length, that is, $num_{classes} = 2^\ell$. Thus, k controls three dimensions of complexity: the number of classes, the length of the building blocks of the problem, and the size of the optimal population.

The best action map is formed by 2^ℓ classifiers, each one with the k relevant variables specified and the class set to the decimal value of the k relevant bits. The complete action map adds ℓ consistently incorrect rules per each consistently correct rule of the best action map; thus, $||O|| = 2^\ell \cdot (\ell + 1)$. Table A.2 shows the best action map and the correct action map for a decoder problem with $\ell = 4$ and $k = 4$. Note that whilst the best action map contains classifiers with all k bits specified, the correct action map contains consistently incorrect classifiers that have all bits general but one.

A.3 Position

Position is an imbalanced multi-class problem defined as follows. Given a binary-input instance of length ℓ , the output is the position of the left-most one-valued bit. The best action map

Table A.3: Best action map (first column) and complete action map (all columns) of position with $\ell=6$.

000000:0	1#####:0	#1#####:0	##1####:0	###1###:0	####1#:0	#####1:0
000001:1	1#####:1	#1#####:1	##1####:1	###1###:1	####1#:1	#####0:1
00001#:2	1#####:2	#1#####:2	##1####:2	###1###:2	####1#:2	
0001##:3	1#####:3	#1#####:3	##1####:3	###0##:3		
001###:4	1#####:4	#1#####:4	##0###:4			
01####:5	1#####:5	#0####:5				
1#####:6	0#####:6					

Table A.4: Best action map (first column) and complete action map (all columns) of the multiplexer problem with $\ell = 6$.

000###:0	000###:1
001###:1	001###:0
01#0##:0	01#0##:1
01#1##:1	01#1##:0
10##0#:0	10##0#:1
10##1#:1	10##1#:0
11###0:0	11###0:1
11###1:1	11###1:0

consists of $\ell + 1$ rules with different levels of generalization. The complete action map needs to maintain $\frac{\ell^2+3\ell}{2}$ consistently incorrect rules; thence, the size of the complete action map is $||[O]|| = \ell + 1 + \frac{\ell^2+3\ell}{2}$. Table A.3 shows the best action map and the complete action map for position with $\ell = 6$. Notice that k controls four dimensions of complexity: the number of classes, the length of the building block, the size of the optimal population, and the imbalance ratio between the most general and the most specific optimal classifier.

A.4 Multiplexer

The multiplexer problem is one of the most used benchmarks in accuracy-based learning classifier systems (Wilson, 1995). The multiplexer is defined for binary strings of size ℓ , where the first $\log_2 \ell$ bits are the address bits and the remaining bits are the position bits. Then, the output is the value of the position bit referred by the decimal value of the address bits. Note that k controls two dimensions of problem difficulty: the length of the building block and the size of the optimal population.

The best action map consists of $2^{\log_2 \ell + 1}$ classifiers with all the address bits and the corresponding position bit specified and all the other bits set to ‘#’. The complete action adds $2^{\log_2 \ell + 1}$ classifiers with the same conditions as those in the best action map, but the opposite class; therefore, $||[O]|| = 2^{\log_2 \ell + 2}$. Table A.4 shows an example of the best action map and the

complete action map for the multiplexer problem with $\ell = 6$.

A.4.1 Imbalanced Multiplexer

The imbalanced multiplexer was introduced in (Orriols-Puig and Bernadó-Mansilla, 2006a) to analyze the effects of under-sampled classes in XCS. Departing from the original multiplexer problem, the imbalanced multiplexer under-samples one of the classes—labeled as the minority class—so that the ratio of the number of instances of class 0 to the number of instances of class 1 sampled to the system equals to the parameter ir , which is specified by the user. The best and the complete action map are the same than those of the original multiplexer problem. Therefore, the system has to generalize the same optimal population regardless of the fact that the instances of one of the classes are under-sampled. Note that the imbalanced multiplexer enables moving the complexity of the problem along three dimensions: the length of the building block, the size of the optimal population, and the imbalance ratio.

A.4.2 Multiplexer with Alternating Noise

The multiplexer with alternating noise was firstly used in (Butz, 2006) to show that XCS with tournament selection is able to handle data sets with inconsistent data. The problem modifies the multiplexer by adding alternating noise. That is, when a new input instance corresponding to the multiplexer problem is sampled, its action is flipped with probability P_x . Again, the best and the complete action map are the same than those of the original multiplexer problem. Note that the multiplexer with alternating noise permits moving the complexity of the problem along three dimensions: the length of the building block, the size of the optimal population, and the proportion of noise in the class of the learning instances.

Appendix B

Statistical Tests

This appendix describes the statistical tests employed along this thesis. We first briefly motivate the use of non-parametric statistical tests for multiple and pairwise comparisons. Then, we describe the methodology used to statistically analyze the results in the experimental comparisons of this thesis. This methodology contemplates comparisons among multiple learners and comparisons between pairs of learners. The procedures followed and the particular tests used in each of the two types of comparisons are explained in detail.

B.1 Statistical Tests for Contrasting Hypotheses

The strong research on machine learning has resulted in the design of several learning algorithms whose behavior is typically compared with existing learning systems on a collection of data sets. For this purpose, the machine learning community has agreed in some common procedures to set up the experiments. Currently, most of the published papers use validation procedures such as k-fold cross validation (Dietterich, 1998) to obtain reliable estimates of the metric employed to assess the quality of the learners. In turn, this quality can be measured with different metrics; for example, in this thesis, we have used the test accuracy, the product of TP rate and TN rate, and the size of the models as some of these indicators. This procedure yields large tables of results in which, for each data set and method, we have n measures of performance ($n \geq 1$) that have been obtained by applying the method in each validation set and repeating this procedure for multiple seeds in case that either the validation process or the learning algorithm is stochastic.

The purpose of this chapter is to provide detail on the statistical tests employed to analyze these tables of results. The underlying idea is to apply statistical methods to contrast our initial hypotheses. Some examples of these hypotheses could be

- “method A outperforms method B”, for pairwise comparisons, or
- “method A is the best of the comparison, surpassing the results obtained with methods B, C, and D”, for multiple comparisons.

All the statistical methodology presented herein follows the recommendations by Demšar (2006), who emphasizes the importance of using non-parametric statistical tests for the types of comparisons usually performed in machine learning. The reason for this is clearly stated by the author;

in general, parametric tests require that the input data—in our case, the tables of results, which consist of the performance of the compared methods—satisfy strong conditions, and the tests to check these conditions need large amounts of data—that is, a large number of data sets—to be effective (for further details, please see (Demšar, 2006)). This is not usually the case in machine learning. For this reason, all the statistical analyses conducted in this thesis have been based on non-parametric tests.

In the following sections, we describe the methodology used to compare (1) pairs of learners (section B.2), and (2) multiple learners (section B.3). All these tests are computed on tables of results in which, for each data set and learner, a single performance measure is provided; in case of multiple runs, the average performance is supplied. For each statistical test, we provide a general description of which type of null hypotheses the statistical tests check and carefully describe the process followed by the test, illustrating the procedure with an example.

B.2 Comparisons of Two Learning Systems

When two algorithms are compared, we aim at contrasting the null hypothesis of whether “*algorithm A significantly outperforms algorithm B on a collection of problems*”. Following the recommendations by Demšar (2006), in our statistical comparisons we avoided using the *paired t-test* parametric test (Sheskin, 2000), probably the most used test for pairwise comparisons in machine learning in the last decade. Although this common use, the main drawback of the paired t-test is that it requires three conditions on the data: commensurability of the data, normal distribution of the differences between the two random variables, and lack of outliers. Since these three conditions are hard to satisfy in the typical machine learning comparisons, our statistical analysis used the non-parametric counterpart of the paired t-test, that is, the *Wilcoxon signed-ranks test* (Wilcoxon, 1945). The Wilcoxon signed-ranks test can be reliably applied to the average of the performance measure computed for a machine learning technique and imposes no additional restrictions to the data. As proceeds, this test is explained in more detail.

B.2.1 The Wilcoxon Signed-Ranks Test

The Wilcoxon signed-ranks test ranks the differences in the performance of two classifiers for each data set, ignoring the signs, and compares the ranks for the positive and the negative differences. As proceeds, we explicate the procedure to compute the statistic for this test. Besides, we exemplify the procedure by applying the statistical test on the results of table B.1, in which the performance of two methods, let us say method M1 and method M2, are compared on a collection of 20 data sets. For each method and data set, the table provides an average value of performance.

The first step of the test is to compute the differences of the performance measures obtained by each method in each data set. These differences are calculated in the fourth column of table B.1. Then, the absolute values of these differences are ranked, considering that the smallest difference holds the first position of the ranking, and the largest difference gets the last position of the ranking. In case of ties, the average rank is assigned to all the elements that have the same performance.

Table B.1: Comparison of the performance of methods M1 and M2 (second and third column). The fourth column provides the performance difference, and the fifth column supplies the rank of the differences.

	M1	M2	difference	rank
<i>ann</i>	97.39	98.61	1.22	11
<i>aut</i>	67.42	69.32	1.90	14
<i>bal</i>	84.40	83.40	-1.00	9
<i>bpa</i>	59.42	58.93	-0.49	7
<i>cmc</i>	49.67	49.42	-0.25	5
<i>col</i>	82.46	78.50	-3.96	18
<i>gls</i>	57.21	57.43	0.22	4
<i>h-c</i>	82.62	82.05	-0.57	8
<i>h-s</i>	80.78	78.11	-2.67	16
<i>irs</i>	95.47	93.73	-1.74	12
<i>pim</i>	74.11	74.32	0.21	3
<i>son</i>	73.71	71.66	-2.05	15
<i>tao</i>	83.02	87.53	4.51	20
<i>thy</i>	89.49	91.25	1.76	13
<i>veh</i>	65.35	65.34	-0.01	1
<i>wbcd</i>	95.73	95.29	-0.44	6
<i>wdbc</i>	94.61	94.51	-0.10	2
<i>wne</i>	94.86	91.82	-3.04	17
<i>wdbc</i>	76.05	71.69	-4.36	19
<i>zoo</i>	94.78	95.90	1.12	10

Thereafter, the method computes R^+ as the sum of ranks for the data sets on which M2 outperformed M1, and R^- as the sum of ranks for the data sets on which M1 outperformed M2. Ranks for which the difference is 0 are split evenly between R^+ and R^- . If there is an odd number of them, one is ignored. In the example, we obtain $R^+ = 75$ and $R^- = 135$.

Then, we assign to T the smaller value between R^+ and R^- , that is, $T = \min(R^+, R^-)$. With T , we can compute the z statistic as

$$z = \frac{T - \frac{1}{4}N(N + 1)}{\sqrt{\frac{1}{24}N(N + 1)(2N + 1)}}, \tag{B.1}$$

where z is distributed normally and N is the number of data sets of the comparison. With $\alpha = 0.05$, the null hypothesis can be rejected if z is smaller than -1.96. The exact p -value can be extracted from the table of the normal distribution.

Let us apply this final step to our example. Replacing $T = 75$ and $N = 20$ into equation B.1, we obtain that $z = -1.11$. As $z > -1.96$, we cannot reject the hypothesis that both learners perform the same, on average, with $\alpha = 0.05$. Besides, by consulting the table of the normal distribution, we can compute the exact p -value by checking the probability whose value approaches the z statistic the most; in this case, we obtained $p = 0.26$.

Table B.2: Comparison of the performance of methods M1, M2, and M3. For each method and data set, the average rank is supplied in parentheses. The last column provides the rank of each learning algorithm for each data set.

	M1	M2	M3
<i>ann</i>	98.85 (1)	97.39 (3)	98.61 (2)
<i>aut</i>	74.42 (1)	67.42 (3)	69.32 (2)
<i>bal</i>	88.65 (1)	84.40 (2)	83.40 (3)
<i>bpa</i>	59.82 (1)	59.42 (2)	58.93 (3)
<i>cmc</i>	51.72 (1)	49.67 (2)	49.42 (3)
<i>col</i>	85.01 (1)	82.46 (2)	78.50 (3)
<i>gls</i>	60.65 (1)	57.21 (3)	57.43 (2)
<i>h-c</i>	84.39 (1)	82.62 (2)	82.05 (3)
<i>h-s</i>	81.33 (1)	80.78 (2)	78.11 (3)
<i>irs</i>	95.67 (1)	95.47 (2)	93.73 (3)
<i>pim</i>	74.88 (1)	74.11 (3)	74.32 (2)
<i>son</i>	80.78 (1)	73.71 (2)	71.66 (3)
<i>tao</i>	81.71 (3)	83.02 (2)	87.53 (1)
<i>thy</i>	88.18 (3)	89.49 (2)	91.25 (1)
<i>veh</i>	67.68 (1)	65.35 (2)	65.34 (3)
<i>wbcd</i>	96.01 (1)	95.73 (2)	95.29 (3)
<i>wdbc</i>	95.20 (1)	94.61 (2)	94.51 (3)
<i>wne</i>	94.12 (2)	94.86 (1)	91.82 (3)
<i>wpbc</i>	76.06 (1)	76.05 (2)	71.69 (3)
<i>zoo</i>	96.50 (1)	94.78 (3)	95.90 (2)
Avg	1.25	2.20	2.55

B.3 Comparisons of Multiple Classifiers

In a multiple classifier comparison, the machine learning practitioner may think of using pairwise comparisons repeatedly, so that all possible pairs of algorithms are compared against each other. Although this has been a common practice in machine learning, statisticians have warned that this procedure causes a certain proportion of the null hypotheses to be rejected due to random chance (Sheskin, 2000; Demšar, 2006). In fact, testing multiple hypotheses is a well-known problem in statistics. Along this thesis, we have followed the methodology proposed by Demšar (2006), which is based on non-parametric statistical tests. This methodology proposes to first apply a multiple-comparison test to analyze whether all the algorithms performed the same on average. If this is the case, no further actions can be taken. Otherwise, different post-hoc tests can be applied depending on the null hypothesis to be tested.

As proceeds, we describe the three multiple-comparison tests used in this thesis. We first describe the Friedman’s tests (Friedman, 1937, 1940), which contrasts the null hypothesis of whether all the algorithms perform the same on average. Then, we explain the Nemenyi test (Nemenyi, 1963), a post-hoc test that compares all learning algorithms with each other. Finally,

we describe the Bonferroni-Dunn procedure (Dunn, 1961), which compares all the learning algorithms with another one that is selected as the control classifier. To exemplify the procedures, we apply each test on the results of the comparison of three learners provided in table B.2.

B.3.1 Friedman’s Test

The Friedman’s test (Friedman, 1937, 1940) is a non-parametric multiple-comparison test that is used to detect significant differences across multiple methods. That is, the procedure checks the null hypothesis of “whether all algorithms perform the same on average”. The procedure is similar to its parametric counter part, the ANOVA test (Fisher, 1959). The key difference between them is that the Friedman’s test is based on the ranks of the algorithms and does not require further assumptions of normality and sphericity of the data (Demšar, 2006). As proceeds, the statistical procedure is detailed and applied to the example in table B.2.

The procedure starts ranking the algorithms for each data set (see the ranks in parentheses in table B.2). In case of ties, the average rank is assigned to each learner. Then, the procedure computes the average rank R_i of each algorithm i , which is provided in the last row of table B.2. Next, the Friedman statistic is computed as

$$\chi_F^2 = \frac{12N}{k(k+1)} \left[\sum_i R_i^2 - \frac{k(k+1)^2}{4} \right] \tag{B.2}$$

where N is the number of data sets, and k is the number of learning algorithms in the comparison. The Friedman statistic is distributed according to the χ^2 distribution with $k - 1$ degrees of freedom.

Let us now calculate the Friedman statistic for the example in table B.2. Replacing $N = 20$ and $k = 3$ in equation B.2 we obtain that

$$\chi_F^2 = \frac{12 \cdot 20}{3 \cdot 4} \left[(1.25^2 + 2.20^2 + 2.55^2) - \frac{3 \cdot 4^2}{4} \right] = 18.1 \tag{B.3}$$

With three algorithms, the statistic behaves as the χ^2 distribution with 2 degrees of freedom. Hence, the critical value of $\chi^2(2)$ for $\alpha = 0.05$ is 10.60. As the computed Friedman statistic is greater than 10.60, we can reject the null hypothesis that all the learners perform the same on average. Besides, we can use the same table to provide the p-value by checking the probability whose value is the closest to the obtained Friedman statistic; in this example, $p=0.0001$.

When the Friedman procedure rejects the null hypothesis, a post-hoc test is applied to detect further differences. The next subsections explicate two of these procedures: the Nemenyi test and the Bonferroni-Dunn test.

B.3.2 Post-hoc Nemenyi Test

The post-hoc Nemenyi test aims at comparing all classifiers with each other. The method is based on the critical distance among learners, that is, the minimum distance between two methods to consider that they are statistically different. The procedure is detailed as follows.

Table B.3: Critical values for the two-tailed Nemenyi test.

#classifiers	2	3	4	5	6	7	8	9	10
$q_{0.05}$	1.960	2.343	2.569	2.728	2.850	2.949	3.031	3.102	3.164
$q_{0.10}$	1.645	2.052	2.291	2.459	2.589	2.693	2.780	2.855	2.920

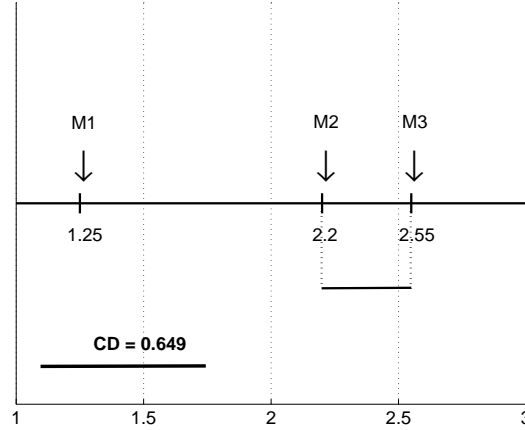


Figure B.1: Comparison of the performance of all classifiers against each other with the Nemenyi test. Groups of classifiers that are not significantly different (at $\alpha = 0.10$) are connected.

The Nemenyi test considers that the performance of two classifiers is different if the distance between their ranks is larger than the critical distance CD computed as

$$CD = q_{\alpha} \sqrt{\frac{k(k+1)}{6N}} \quad (\text{B.4})$$

where N and k are the number of learners and the number of data sets respectively, and q_{α} is the critical value based on the Studentized range statistic (Sheskin, 2000). Table B.3 provides the critical values for the Nemenyi test for $\alpha = \{0.05, 0.01\}$ and from $k = 2$ to $k = 10$.

The critical distance for the example in table B.2 is calculated as follows. Recognizing that $k = 3$, we can extract, from table B.3, that $q_{0.10} = 2.052$ at $\alpha = 0.10$. Thence, the critical distance is

$$CD_{\alpha=0.10} = 2.052 \sqrt{\frac{3 \cdot 4}{6 \cdot 20}} = 0.649 \quad (\text{B.5})$$

Therefore, any pair of algorithms whose rank differs by more than 0.649 perform significantly different. These results are illustrated in figure B.1 in which each learner is depicted according to its rank, and all the algorithms that perform equivalently are connected with a line.

Table B.4: Critical values for the two-tailed Bonferroni-Dunn test.

<i>#classifiers</i>	2	3	4	5	6	7	8	9	10
$q_{0.05}$	1.960	2.241	2.394	2.498	2.576	2.638	2.690	2.724	2.773
$q_{0.10}$	1.645	1.960	2.128	2.241	2.326	2.394	2.450	2.498	2.539

B.3.3 Post-hoc Bonferroni-Dunn Test

If we want to compare all learning systems with respect to a control learner, we can use the Bonferroni-Dunn (Dunn, 1961) test instead of the Nemenyi test. As proceeds, this statistical procedure is explained in detail.

The Bonferroni-Dunn test provides a general procedure to control the family-wise error in multiple hypotheses tests by dividing α by the number of comparisons that have to be performed; in our case, we perform $k - 1$ comparisons since each learner is compared with the control algorithm. The statistical procedure can be computed in two different ways. Firstly, the statistic for comparing the i th and the j th classifiers can be calculated as

$$z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6N}}} \tag{B.6}$$

where R_i and R_j are the ranks of the i th and the j th learners, k is the number of learning systems in the comparison, and N is the number of data sets. Once computed, the z value is used to find the corresponding probability from the table of normal distribution, which can be compared with the desired α . As mentioned, the significance level needs to be adjusted as $\alpha/(k - 1)$.

The second equivalent procedure to compute the Bonferroni-Dunn test is by using the concept of critical distance, as done by the Nemenyi test. The critical distance is computed as indicated in equation B.4, where the values of q_α are calculated from the Studentized range statistic, but with the difference that $\alpha/(k - 1)$ instead of α is considered to obtain these values. Table B.4 provides the critical values for the Bonferroni-Dunn test for $\alpha = \{0.05, 0.01\}$ and from $k = 2$ to $k = 10$. The advantage of applying this second procedure to compute the Bonferroni-Dunn test is that the results can be visually illustrated, as done for the Bonferroni-Dunn test.

Let us now compute the Bonferroni-Dunn test for the example in table B.2. The first step is to identify the control learner with which the classifiers will be compared. Let us assume that we want to compare the different classifiers with the best rank method, that is, M1. Then, having that $k = 3$, table B.3 indicates that $q_{0.10} = 1.960$ at $\alpha = 0.10$. Thence, the critical distance is

$$CD = 1.960 \sqrt{\frac{3 \cdot 4}{6 \cdot 20}} = 0.619 \tag{B.7}$$

Then, we compare the differences of ranks between each classifier and the control learner. The difference between the ranks of M1 and M2 is 0.95 which is greater than 0.619; therefore, M2 significantly degrades the results of M1. Similarly, the difference between the ranks of M1 and M3 is 1.30, which in turn is greater than 0.619; thence, M1 also significantly outperforms M3

according to a Bonferroni-Dunn test at $\alpha = 0.10$. The same graphical representation as the one done for the Nemenyi test can be used here. Nonetheless, note that, in this case, M1 significantly outperforms all the other methods, so no graphic representation is necessary.

B.4 Summary

This appendix has described the statistical tests used along the thesis. We first briefly discussed the robustness of non-parametric tests with respect to parametric tests and draw the methodology used in this thesis to compare pairs of classifiers and multiple (more than two) learning systems. Then, each of these two types of comparisons got a different section where the particular statistical methods used in our analyses were described in detail, providing a detailed example of use for each one.

Appendix C

Full Results of the Comparison of the Re-sampling Techniques

Section 7.5 analyzed whether the application of re-sampling methods improved the accuracy of the models extracted by XCS, UCS, C4.5, SMO, and IBk on imbalanced domains. For compactness, the analysis only gathered the statistical analysis and extracted conclusions from it. The purpose of this appendix is to provide the full results of the comparison, which involved the following four re-sampling techniques: (1) random over-sampling, (2) under-sampling based on Tomek links, (3) SMOTE, and (4) cSMOTE. As proceeds, we describe and provide the tables of results.

Description of the Tables of Results

Tables C.1, C.2, C.3, C.4, and C.5 supply the average of the product of the TP rate and the TN rate obtained in each data set by C4.5, SMO, IBk, XCS, and UCS respectively. Moreover, to let a more detailed analysis of each problem, we also provide pair-wise comparisons for each particular combination of learner and re-sampling technique per data set. The ● and ○ symbols denote a significant degradation/improvement of the method in the corresponding data set with respect to the same data set but with another re-sampling method (or without re-sampling). We acknowledge in advance that pair-wise comparisons may be taken with a grain of salt; nevertheless, they also help draw good insights about the performance of each learner. Finally, the last four rows of the table summarize (i) the average performance, (ii) the average rank, (iii) the position of each learner in the ranking, and (iv) the number of times that the results obtained with the learner are surpassed/degraded by another learner.

Table C.1: Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **C4.5** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 data sets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one.

	Original	Ovs	UnsTL	SMOTE	cSMOTE
<i>bald1</i>	0.00 ± 0.00 \bullet	19.91 ± 37.27 $\circ\circ\circ\circ$	0.00 ± 0.00 \bullet	0.00 ± 0.00 \bullet	1.90 ± 6.32 \bullet
<i>bald2</i>	69.30 ± 6.83 \bullet	67.57 ± 6.85 $\bullet\bullet$	68.03 ± 6.40 \bullet	72.73 ± 8.28 \circ	75.35 ± 6.88 $\circ\circ\circ$
<i>bald3</i>	71.20 ± 6.04 $\bullet\bullet$	68.31 ± 5.55 \bullet	65.78 ± 4.91 $\bullet\bullet\bullet$	72.82 ± 4.21 \circ	77.75 ± 6.87 $\circ\circ\circ$
<i>bpa</i>	33.08 ± 14.09	36.81 ± 11.78	31.67 ± 17.31	33.31 ± 7.49	29.56 ± 15.74
<i>glsd1</i>	79.50 ± 42.16	89.50 ± 31.62	89.81 ± 0.00 \bullet	99.50 ± 0.00 $\circ\circ$	59.00 ± 51.64 \bullet
<i>glsd2</i>	34.50 ± 47.43 \bullet	82.50 ± 33.75 \circ	58.50 ± 48.30	63.50 ± 47.43	68.00 ± 42.16
<i>glsd3</i>	28.97 ± 42.16 \bullet	46.45 ± 47.14	45.96 ± 35.36	64.95 ± 42.16 \circ	24.22 ± 35.36
<i>glsd4</i>	73.55 ± 32.63	80.70 ± 25.40	71.73 ± 19.95 \bullet	84.93 ± 19.33 \circ	74.29 ± 32.63
<i>glsd5</i>	66.52 ± 16.77 \circ	70.29 ± 17.10 \circ	52.12 ± 15.06 $\bullet\bullet\bullet\bullet$	64.77 ± 16.56 \circ	66.68 ± 15.43 \circ
<i>glsd6</i>	52.54 ± 15.13	53.80 ± 9.75	49.78 ± 15.69	54.54 ± 24.12	46.34 ± 25.53
<i>h-s</i>	63.33 ± 13.29	58.06 ± 9.98	61.17 ± 15.84	59.44 ± 14.80	56.67 ± 13.72
<i>pim</i>	43.87 ± 13.27 \bullet	54.68 ± 7.25 $\circ\circ$	50.50 ± 12.19	52.38 ± 8.98	46.76 ± 7.87 \bullet
<i>tao</i>	90.98 ± 2.14 \bullet	90.98 ± 2.14 \bullet	91.10 ± 1.30	92.71 ± 1.53 $\circ\circ$	91.86 ± 3.96
<i>thyd1</i>	87.61 ± 16.10	84.63 ± 17.21	87.31 ± 14.05	82.09 ± 17.21	76.53 ± 17.21
<i>thyd2</i>	93.24 ± 12.45 \circ	91.94 ± 12.45	85.32 ± 13.61 \bullet	91.11 ± 13.61	88.06 ± 16.87
<i>thyd3</i>	87.65 ± 10.34	88.13 ± 8.08	86.06 ± 7.81	83.25 ± 11.99	84.79 ± 8.05
<i>wavd1</i>	67.79 ± 4.06 $\bullet\circ$	67.99 ± 2.93 \circ	70.25 ± 3.36 $\circ\circ$	70.75 ± 4.02 \circ	64.04 ± 3.50 $\bullet\bullet\bullet\bullet$
<i>wavd2</i>	62.54 ± 3.89	65.05 ± 3.62 \circ	64.02 ± 3.19	64.41 ± 2.91 \circ	61.33 ± 3.56 $\bullet\bullet$
<i>wavd3</i>	68.60 ± 2.38 $\bullet\circ$	69.35 ± 3.35 $\bullet\circ$	70.54 ± 2.18 $\circ\circ$	71.45 ± 3.50 $\circ\circ$	65.66 ± 3.54 $\bullet\bullet\bullet\bullet$
<i>wbcd</i>	89.12 ± 3.42	89.63 ± 4.21	90.70 ± 3.29	91.94 ± 2.18	91.94 ± 5.17
<i>wdbc</i>	88.79 ± 5.09	86.51 ± 6.88	85.95 ± 5.01	87.67 ± 3.30	86.53 ± 5.04
<i>wined1</i>	85.15 ± 16.63	89.62 ± 16.63	79.92 ± 16.36	89.92 ± 16.36	89.69 ± 13.98
<i>wined2</i>	91.81 ± 8.05	89.38 ± 8.05	87.71 ± 8.05 \bullet	91.74 ± 8.05 \circ	88.47 ± 8.78
<i>wined3</i>	87.62 ± 11.70	84.46 ± 11.68	84.03 ± 9.64	84.61 ± 6.90	83.36 ± 11.76
<i>wpbc</i>	33.55 ± 12.87	33.30 ± 21.92	30.19 ± 13.29	38.36 ± 21.66	30.66 ± 28.36
Avg	66.03	70.38	66.33	70.52	65.18
Rank	3.00	2.60	3.68	2.14	3.58
Pos	3	2	5	1	4
Inf/Sup	9/5	5/11	13/4	1/14	13/7

Table C.2: Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **SMO** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 data sets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one.

	Original	Ovs	UnsTL	SMOTE	cSMOTE
<i>bald1</i>	0.00 \pm 0.00 \bullet	16.38 \pm 13.78 $\circ\circ\circ\circ$	0.00 \pm 0.00 \bullet	0.00 \pm 0.00 \bullet	0.00 \pm 0.00 \bullet
<i>bald2</i>	84.03 \pm 7.30 $\circ\circ$	84.28 \pm 7.11 $\circ\circ$	85.09 \pm 6.50 $\circ\circ$	76.94 \pm 8.26 $\bullet\bullet\bullet$	77.88 \pm 9.00 $\bullet\bullet\bullet$
<i>bald3</i>	85.81 \pm 8.40 $\bullet\bullet$	85.43 \pm 9.16 $\bullet\bullet$	85.24 \pm 6.11 $\bullet\bullet$	78.06 \pm 9.85 $\bullet\bullet\bullet\bullet$	90.13 \pm 7.76 $\circ\circ\circ\circ$
<i>bpa</i>	0.00 \pm 0.00 $\bullet\bullet\bullet$	36.33 \pm 6.75 $\circ\circ\circ\circ$	11.99 \pm 4.16 $\bullet\bullet\circ\circ$	0.00 \pm 0.00 $\bullet\bullet\bullet$	28.18 \pm 11.16 $\bullet\circ\circ\circ$
<i>glsd1</i>	0.00 \pm 0.00 $\bullet\bullet$	87.83 \pm 7.60 $\circ\circ\circ$	71.71 \pm 24.59 $\circ\circ\circ$	10.00 \pm 31.62 $\bullet\bullet$	0.00 \pm 0.00 $\bullet\bullet$
<i>glsd2</i>	15.00 \pm 33.75 $\bullet\bullet\bullet$	82.50 \pm 29.74 $\circ\circ$	67.00 \pm 36.61 $\bullet\circ\circ$	74.50 \pm 39.61 $\circ\circ\circ$	15.00 \pm 33.75 $\bullet\bullet\bullet$
<i>glsd3</i>	0.00 \pm 0.00 $\bullet\bullet$	30.36 \pm 11.39 $\circ\circ\circ$	21.03 \pm 16.84 $\circ\circ\circ$	0.00 \pm 0.00 $\bullet\bullet$	0.00 \pm 0.00 $\bullet\bullet$
<i>glsd4</i>	80.03 \pm 24.33	85.09 \pm 17.44 \circ	82.03 \pm 16.81 $\bullet\bullet$	85.61 \pm 17.83 \circ	82.99 \pm 19.30
<i>glsd5</i>	9.50 \pm 9.42 $\bullet\bullet\bullet\bullet$	44.67 \pm 16.83 $\circ\bullet$	38.33 \pm 13.33 $\bullet\bullet$	42.57 \pm 14.42 \circ	28.65 \pm 22.27 \circ
<i>glsd6</i>	0.00 \pm 0.00 $\bullet\bullet\bullet\bullet$	26.89 \pm 13.24 $\circ\circ$	27.36 \pm 11.27 $\circ\circ$	17.71 \pm 8.40 $\bullet\bullet\circ$	22.28 \pm 14.58 \circ
<i>h-s</i>	68.83 \pm 8.87 \circ	66.89 \pm 9.24	67.61 \pm 7.32	65.83 \pm 12.08	63.78 \pm 8.70 \bullet
<i>pim</i>	48.31 \pm 5.60 $\bullet\bullet\bullet$	55.75 \pm 7.12 \circ	56.25 \pm 6.88 $\circ\circ$	49.79 \pm 8.14 \bullet	53.44 \pm 8.05 \circ
<i>tao</i>	70.59 \pm 6.45 $\circ\circ$	70.59 \pm 6.45 $\circ\circ$	70.49 \pm 6.15 \circ	62.44 \pm 5.91 $\bullet\bullet\bullet\bullet$	69.28 \pm 6.92 $\bullet\bullet\circ$
<i>thyd1</i>	76.67 \pm 22.50 \bullet	90.00 \pm 16.10 $\circ\circ$	80.00 \pm 23.31	80.00 \pm 23.31	76.67 \pm 22.50 \bullet
<i>thyd2</i>	54.17 \pm 24.92 $\bullet\bullet\bullet\bullet$	98.33 \pm 2.68 $\circ\circ$	93.61 \pm 12.29 \circ	96.39 \pm 7.86 $\circ\circ$	80.83 \pm 18.02 $\bullet\bullet\circ$
<i>thyd3</i>	33.81 \pm 21.35 $\bullet\bullet\bullet$	52.38 \pm 19.76 $\circ\circ\circ$	41.67 \pm 21.68 $\bullet\bullet$	62.13 \pm 18.65 $\circ\circ\circ$	39.76 \pm 24.49 $\bullet\bullet\circ$
<i>wavd1</i>	78.68 \pm 4.27 $\bullet\bullet$	80.98 \pm 3.00 $\circ\circ$	80.81 \pm 2.90 $\circ\circ$	80.35 \pm 2.37	78.84 \pm 3.73 $\bullet\bullet$
<i>wavd2</i>	72.30 \pm 2.71 $\bullet\bullet$	75.15 \pm 2.18 $\circ\circ$	74.67 \pm 1.69 \circ	74.40 \pm 2.56	73.75 \pm 1.80 \bullet
<i>wavd3</i>	79.57 \pm 2.04	81.09 \pm 1.35 \circ	80.84 \pm 1.49	80.55 \pm 1.57 \bullet	79.75 \pm 2.58
<i>wbcd</i>	92.70 \pm 5.32 \bullet	93.70 \pm 5.06	93.31 \pm 5.64 \bullet	95.30 \pm 4.68 $\circ\circ$	93.12 \pm 6.19
<i>wdbc</i>	94.28 \pm 3.28	93.64 \pm 4.66	93.60 \pm 3.04	93.63 \pm 4.82	92.80 \pm 4.71
<i>wined1</i>	98.46 \pm 3.24	98.46 \pm 3.24	96.15 \pm 4.05	98.46 \pm 3.24	96.46 \pm 6.61
<i>wined2</i>	97.50 \pm 5.62	97.50 \pm 4.03	95.00 \pm 4.30	96.67 \pm 4.30	96.67 \pm 5.83
<i>wined3</i>	97.14 \pm 6.02	95.71 \pm 6.90	92.94 \pm 8.14	95.23 \pm 6.36	94.45 \pm 7.93
<i>wpsc</i>	9.37 \pm 16.98 $\bullet\bullet\bullet\bullet$	43.76 \pm 16.91 \circ	38.92 \pm 19.79 \circ	42.35 \pm 18.95 \circ	43.85 \pm 21.27 \circ
Avg	53.87	70.95	65.83	62.36	59.14
Rank	3.74	1.60	2.92	3.08	3.66
Pos	5	1	2	3	4
Inf/Sup	6/40	1/40	11/24	23/14	23/14

Table C.3: Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **IBk** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 datasets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one.

	Original	Ovs	UnsTL	SMOTE	cSMOTE
<i>bald1</i>	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	4.06 ± 9.56	0.00 ± 0.00
<i>bald2</i>	81.16 ± 5.54 $\circ\circ$	79.66 ± 5.09 \circ	73.83 ± 3.26 $\bullet\bullet\bullet$	78.60 ± 6.09 \circ	75.25 ± 6.45 \bullet
<i>bald3</i>	82.11 ± 8.67 $\circ\circ$	80.95 ± 7.66 \circ	75.25 ± 6.21 $\bullet\bullet$	78.32 ± 8.46 \bullet	79.61 ± 10.01
<i>bpa</i>	32.40 ± 9.44	36.10 ± 13.80 \circ	32.68 ± 11.87	28.97 ± 8.57 \bullet	32.35 ± 13.56
<i>glsd1</i>	69.32 ± 48.30	86.94 ± 31.62	65.07 ± 0.00	87.81 ± 31.62	68.28 ± 48.30
<i>glsd2</i>	24.13 ± 35.36 $\bullet\bullet\bullet$	79.94 ± 33.75 \circ	68.80 ± 42.16 \circ	84.62 ± 31.62 \circ	72.75 ± 42.49
<i>glsd3</i>	0.00 ± 0.00 $\bullet\bullet$	38.64 ± 43.78 \circ	37.01 ± 35.36 \circ	44.70 ± 33.33	28.03 ± 34.96
<i>glsd4</i>	77.07 ± 24.98	76.23 ± 24.91	76.84 ± 25.40	76.64 ± 24.91	77.92 ± 24.91
<i>glsd5</i>	62.26 ± 21.14 \circ	62.37 ± 15.72	58.37 ± 14.75 \bullet	62.57 ± 15.06	62.52 ± 17.88
<i>glsd6</i>	61.74 ± 18.23	61.70 ± 15.76	59.84 ± 24.83	63.19 ± 12.46	60.99 ± 17.86
<i>h-s</i>	64.40 ± 14.65	61.52 ± 10.24	59.11 ± 11.95	60.63 ± 7.91	61.50 ± 13.03
<i>pim</i>	46.91 ± 4.84	50.27 ± 11.30 \circ	51.50 ± 9.82 \circ	49.65 ± 6.03 \circ	44.05 ± 11.48 $\bullet\bullet\bullet$
<i>tao</i>	94.25 ± 2.10 $\circ\circ$	92.61 ± 2.00	92.61 ± 2.17 \bullet	93.02 ± 2.29	91.92 ± 2.15 \bullet
<i>thyd1</i>	76.67 ± 22.50	91.26 ± 14.05	76.09 ± 23.31	84.28 ± 23.31	81.51 ± 23.57
<i>thyd2</i>	77.90 ± 21.40 $\bullet\bullet\bullet\bullet$	98.33 ± 0.00 \circ	95.88 ± 7.91 \circ	98.33 ± 0.00 \circ	93.98 ± 10.54 \circ
<i>thyd3</i>	81.12 ± 16.16	92.38 ± 6.55	87.31 ± 7.84	88.81 ± 8.03	87.84 ± 11.45
<i>wavd1</i>	72.28 ± 3.97 $\circ\circ$	71.62 ± 2.14 $\circ\circ$	72.34 ± 2.42 $\circ\circ$	66.67 ± 0.77 $\bullet\bullet\bullet$	65.71 ± 2.93 $\bullet\bullet\bullet$
<i>wavd2</i>	67.49 ± 1.75 $\circ\circ\circ$	65.62 ± 1.79 $\bullet\circ\circ$	66.69 ± 2.50 $\circ\circ$	57.51 ± 0.76 $\bullet\bullet\bullet\bullet$	61.08 ± 3.58 $\bullet\bullet\bullet\circ$
<i>wavd3</i>	74.14 ± 2.86 $\circ\circ$	73.71 ± 1.98 $\bullet\circ\circ$	74.81 ± 2.70 $\circ\circ\circ$	68.32 ± 0.95 $\bullet\bullet\bullet\bullet$	65.53 ± 2.96 $\bullet\bullet\bullet\bullet$
<i>wbcd</i>	92.72 ± 5.36	94.91 ± 2.13	93.53 ± 3.83	95.02 ± 1.32	92.34 ± 4.49
<i>wdbc</i>	93.47 ± 3.64	91.70 ± 3.51	93.59 ± 3.51	91.45 ± 2.70	91.21 ± 4.53
<i>wined1</i>	94.98 ± 8.29	96.15 ± 0.00	93.85 ± 0.00	97.69 ± 0.00	96.92 ± 0.00
<i>wined2</i>	97.50 ± 4.03 \circ	95.76 ± 0.00 \circ	92.35 ± 0.00 $\bullet\bullet\bullet\bullet$	95.76 ± 0.00 \circ	96.59 ± 0.00 \circ
<i>wined3</i>	87.94 ± 12.53	91.43 ± 9.99	92.01 ± 7.53	95.71 ± 6.90	93.34 ± 7.38
<i>wpbc</i>	28.98 ± 16.49	29.36 ± 20.88 \bullet	27.70 ± 22.66 $\bullet\bullet$	37.39 ± 19.73 $\circ\circ$	37.10 ± 21.96 \circ
Avg	65.64	71.97	68.68	71.59	68.73
Rank	2.98	2.56	3.52	2.44	3.50
Pos	3	2	5	1	4
Inf/Sup	9/15	3/14	13/11	12/8	15/4

Table C.4: Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **XCS** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 data sets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one.

	Original	Ovs	UnsTL	SMOTE	cSMOTE
<i>bald1</i>	0.00 ± 0.00	1.98 ± 6.27	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
<i>bald2</i>	71.14 ± 5.02	70.06 ± 7.81	71.58 ± 5.03	72.79 ± 9.49	73.10 ± 6.56
<i>bald3</i>	69.98 ± 7.23 $\bullet\bullet\bullet$	73.95 ± 4.75	73.47 ± 6.03 \circ	72.78 ± 6.71 $\bullet\bullet$	76.15 ± 6.58 $\circ\circ$
<i>bpa</i>	47.58 ± 10.92 $\circ\circ$	45.05 ± 12.20 \circ	38.61 ± 9.89 \circ	22.40 ± 14.39 $\bullet\bullet\bullet\bullet$	40.69 ± 11.47 $\bullet\circ$
<i>glsd1</i>	20.00 ± 42.16 $\bullet\bullet$	69.50 ± 47.98 \circ	73.00 ± 27.01 \circ	58.50 ± 50.45	59.50 ± 51.23
<i>glsd2</i>	59.00 ± 45.02	59.50 ± 45.49	62.75 ± 35.72	73.50 ± 41.57	63.00 ± 45.90
<i>glsd3</i>	0.00 ± 0.00 $\bullet\bullet\bullet$	47.99 ± 38.80 $\circ\circ$	42.11 ± 17.78 $\circ\circ$	42.22 ± 40.93 $\circ\circ$	9.74 ± 20.54 $\bullet\bullet\bullet$
<i>glsd4</i>	80.03 ± 24.33	87.25 ± 18.72	84.68 ± 15.68	80.59 ± 25.66	79.66 ± 24.59
<i>glsd5</i>	68.67 ± 18.71	66.17 ± 15.41 \bullet	68.28 ± 19.81	73.07 ± 17.37 $\circ\circ$	62.44 ± 16.52 \bullet
<i>glsd6</i>	60.53 ± 11.21	63.48 ± 14.17	64.18 ± 12.62	64.50 ± 14.51	62.47 ± 12.64
<i>h-s</i>	59.89 ± 15.59	58.00 ± 11.45 \bullet	58.61 ± 14.68	65.61 ± 15.12 \circ	60.22 ± 15.51
<i>pim</i>	45.85 ± 6.37 $\bullet\bullet$	50.53 ± 4.89 $\circ\circ$	51.24 ± 7.64 \bullet	55.41 ± 8.76 $\circ\circ\circ$	48.36 ± 8.97
<i>tao</i>	82.89 ± 5.42 $\bullet\circ$	83.60 ± 6.04 \circ	83.39 ± 5.91 \circ	58.01 ± 31.57 $\bullet\bullet\bullet\bullet$	84.45 ± 6.34 $\circ\circ$
<i>thyd1</i>	78.36 ± 22.01 \bullet	87.96 ± 15.98	81.29 ± 20.83 \bullet	91.70 ± 13.68 $\circ\circ$	89.07 ± 16.44
<i>thyd2</i>	82.50 ± 24.98	95.56 ± 10.41	92.78 ± 11.43	91.94 ± 11.60	93.06 ± 12.09
<i>thyd3</i>	89.84 ± 11.75	87.25 ± 10.81 \bullet	88.43 ± 11.84	93.11 ± 8.73 \circ	87.51 ± 10.38
<i>wavd1</i>	80.44 ± 2.97	81.91 ± 3.24 \circ	80.87 ± 3.58	80.24 ± 2.00	80.11 ± 2.97 \bullet
<i>wavd2</i>	73.48 ± 2.88	76.03 ± 2.24 $\circ\circ$	75.60 ± 1.52 \circ	71.96 ± 2.78 $\bullet\bullet$	73.84 ± 2.89 \bullet
<i>wavd3</i>	81.01 ± 3.99	82.01 ± 2.05 $\circ\circ$	81.16 ± 2.49	80.23 ± 2.19 \bullet	79.46 ± 2.59 \bullet
<i>wbcd</i>	92.31 ± 5.50	92.72 ± 6.01	92.49 ± 5.63	94.42 ± 4.41	92.70 ± 6.13
<i>wdbc</i>	90.27 ± 4.61	88.16 ± 6.33	90.48 ± 4.13	92.17 ± 4.95	91.08 ± 6.24
<i>wined1</i>	99.23 ± 2.43	95.69 ± 6.60	96.15 ± 5.44	97.69 ± 3.72	96.62 ± 8.36
<i>wined2</i>	99.17 ± 2.64	95.76 ± 5.96	96.67 ± 4.30	98.33 ± 3.51	97.50 ± 5.62
<i>wined3</i>	93.38 ± 7.15	91.86 ± 9.81	92.29 ± 6.84	94.11 ± 8.13	92.12 ± 7.74
<i>wpbc</i>	20.33 ± 16.38	25.84 ± 19.03	25.32 ± 17.21	31.65 ± 18.85	21.35 ± 11.96
Avg	65.83	71.11	70.62	70.28	68.57
Rank	3.60	2.86	2.94	2.42	3.18
Pos	5	2	3	1	4
Inf/Sup	12/3	4/11	2/7	12/12	8/5

Table C.5: Comparison of the performance, measured as the product of TP rate and TN rate, achieved by **UCS** with the original and re-sampled data sets. For each method and data set, the \bullet and \circ symbols indicate that the method is statistically inferior/superior than another of the learners according to a Wilcoxon signed-ranks test at $\alpha = 0.05$. *Avg* provides the performance average of each method over the 25 data sets. Rows *Rank* and *Pos* show the average rank of each learning algorithm and its position in the ranking respectively. The last row provides *Inf/Sup*, where *Inf* is the number of times that the learner has been surpassed by another one, and *Sup* is the number of times that the method has outperformed another one.

	Original	Ovs	UnsTL	SMOTE	cSMOTE
<i>bald1</i>	0.00 ± 0.00	3.23 ± 6.82	0.00 ± 0.00	3.55 ± 7.48	0.00 ± 0.00
<i>bald2</i>	69.75 ± 8.19	72.07 ± 6.79	72.35 ± 5.60	72.73 ± 8.03	73.22 ± 5.32
<i>bald3</i>	73.61 ± 6.66 \bullet	72.18 ± 5.14 \bullet	74.01 ± 7.05 \bullet	73.15 ± 6.58 \bullet	78.40 ± 6.64 $\circ\circ\circ$
<i>bpa</i>	47.59 ± 11.22	41.72 ± 10.60	48.29 ± 9.68	41.20 ± 7.60	40.74 ± 9.18
<i>glsd1</i>	59.00 ± 50.87	59.50 ± 51.23	72.19 ± 18.80	68.52 ± 47.33	58.52 ± 50.46
<i>glsd2</i>	74.00 ± 41.89 \circ	63.50 ± 46.25	77.12 ± 27.40 \circ	82.50 ± 32.68 \circ	38.50 ± 49.78 $\bullet\bullet\bullet$
<i>glsd3</i>	19.49 ± 25.17	33.50 ± 45.22	28.93 ± 22.94	45.24 ± 42.50	23.25 ± 24.64
<i>glsd4</i>	83.54 ± 19.53	87.25 ± 18.72 \circ	74.41 ± 24.70 \bullet	82.67 ± 19.50	77.25 ± 28.64
<i>glsd5</i>	65.63 ± 21.46	68.54 ± 16.54	64.50 ± 14.10	62.54 ± 23.57	70.44 ± 17.42
<i>glsd6</i>	57.06 ± 14.20	61.64 ± 18.57	69.26 ± 21.48	62.70 ± 12.96	59.05 ± 15.39
<i>h-s</i>	55.00 ± 13.61	54.17 ± 16.60	55.61 ± 14.45	57.00 ± 15.95	52.83 ± 17.96
<i>pim</i>	47.82 ± 6.60	49.38 ± 5.11	52.45 ± 6.93 \circ	51.89 ± 8.05 \circ	46.74 ± 6.71 $\bullet\bullet$
<i>tao</i>	78.81 ± 7.18	80.65 ± 6.64	78.21 ± 4.27	75.72 ± 7.23	78.53 ± 7.51
<i>thyd1</i>	92.25 ± 13.66	88.89 ± 15.71	88.92 ± 15.51	91.88 ± 14.46	88.89 ± 15.49
<i>thyd2</i>	93.06 ± 12.09	91.94 ± 11.60	84.81 ± 15.61	93.33 ± 9.99	94.44 ± 10.14
<i>thyd3</i>	88.08 ± 14.89	84.98 ± 10.52	86.79 ± 9.80	87.95 ± 8.94	85.03 ± 10.16
<i>wavd1</i>	76.33 ± 2.10 $\bullet\bullet$	78.18 ± 3.10 $\circ\circ$	78.99 ± 3.77 $\circ\circ$	78.66 ± 3.35 \circ	75.56 ± 3.75 $\bullet\bullet\bullet$
<i>wavd2</i>	71.49 ± 3.83	73.08 ± 2.93 \circ	70.57 ± 2.67 \bullet	74.46 ± 2.76 $\circ\circ$	69.66 ± 1.60 $\bullet\bullet$
<i>wavd3</i>	76.60 ± 4.14 $\bullet\bullet$	80.60 ± 2.21 $\circ\circ$	78.32 ± 2.78 \circ	79.56 ± 1.84 $\circ\circ$	75.10 ± 2.75 $\bullet\bullet\bullet$
<i>wbcd</i>	94.06 ± 4.23	93.10 ± 4.97	93.10 ± 4.35	94.25 ± 4.81	94.28 ± 4.36
<i>wdbc</i>	89.68 ± 5.61	90.54 ± 4.08	89.84 ± 4.43	90.38 ± 7.75	87.81 ± 6.17
<i>wined1</i>	99.23 ± 2.43	100.00 ± 0.00 $\circ\circ$	94.92 ± 6.49 \bullet	96.92 ± 3.97 \bullet	93.96 ± 9.30
<i>wined2</i>	91.88 ± 10.02	95.83 ± 7.08 \circ	92.56 ± 7.98 \bullet	94.92 ± 7.07	95.83 ± 7.08
<i>wined3</i>	85.33 ± 9.55 $\bullet\bullet\bullet\bullet$	94.11 ± 8.17 \circ	93.86 ± 8.52 \circ	94.71 ± 6.94 \circ	91.90 ± 5.97 \circ
<i>wpbc</i>	17.17 ± 21.63	21.55 ± 14.96	25.55 ± 18.72	30.70 ± 15.04	21.34 ± 13.56
Avg	68.26	69.61	69.82	71.49	66.85
Rank	3.44	2.78	2.86	2.24	3.68
Pos	4	2	3	1	5
Inf/Sup	9/1	1/10	5/6	2/8	13/5

Appendix D

Empirical Analysis of the Sensitivity of Fuzzy-UCS to Configuration Parameters

As many competitive Michigan-style LCSs, Fuzzy-UCS has several configuration parameters, which permit adjusting the behavior of the system to evolve models with maximum quality for particular problems. At a first glance, choosing a correct configuration may seem a crucial task only suitable to expert users. Nonetheless, several analyses identified the robustness of Michigan-style LCSs to the majority of configuration parameters. Actually, most of the applications of Michigan-style LCSs used the same default parameters to solve pattern recognition problems (Bernadó-Mansilla et al., 2002; Bernadó-Mansilla and Garrell, 2003; Butz, 2006; Orriols-Puig and Bernadó-Mansilla, 2008b; Dixon et al., 2002, 2004; Fu et al., 2001; Wilson, 2000). We consider that this robustness is also present in Fuzzy-UCS. For this reason, we used the same default configuration to solve the collection of real-world problems in all the experiments conducted in chapter 8.

The aim of this appendix is to empirically show the robustness of Fuzzy-UCS to configuration parameters. For this purpose, we systematically analyze the impact of the parameter settings on the quality of the final solution on a set of real-world problems; besides, we relate the results to theoretical and empirical studies of the sensitivity of LCSs—particularly XCS and UCS—to configuration parameters. It is worth highlighting that the following study does not pretend to establish guidelines to configure Fuzzy-UCS, but to intuitively show the effect of the different parameters.

The remainder of this appendix is organized as follows. Section D.1 gathers and provides a brief description all the parameters of Fuzzy-UCS. Section D.2 details the experimental methodology followed in the analysis. Section D.3 compares different configurations of Fuzzy-UCS to the default configuration used in chapter 8 and studies the impact of changing the configuration of the different parameters. Finally, section D.4 summarizes and concludes.

D.1 Configuration Parameters of Fuzzy-UCS

The configuration parameters of Fuzzy-UCS are:

1. N : Maximum population size.
2. $P_{\#}$: Generalization probability in covering.
3. ν : Fitness pressure.
4. F_0 : Minimum fitness required for subsumption.
5. θ_{GA} : Threshold that controls the application period of the genetic algorithm on each niche.
6. θ_{sub} : Minimum experience required to be a candidate subsumer.
7. $\theta_{exploit}$: Minimum experience required to participate in the class inference of a new example.
8. θ_{del} : Minimum experience required to use classifier's fitness to calculate its deletion vote.
9. δ : Fraction of mean fitness below which the deletion probability of a classifier is further decreased according to the ratio of its fitness to the average fitness of the population.
10. χ : Probability of crossover.
11. μ : Probability of mutation.

D.2 Experimental Methodology

For the sake of clarity, we analyzed the effect of different parameters or groups of related parameters separately. Specifically, we examined the sensitivity of Fuzzy-UCS to:

1. Rule initialization (parameter $P_{\#}$). That is, we studied how the generalization degree in the initial population affected the quality of the models.
2. Fitness pressure (parameter ν). We analyzed to which extend the selection pressure toward highly accurate classifiers affected the learning process.
3. Genetic algorithm. We empirically showed the effect of changing a set of parameters related to the genetic algorithm: θ_{GA} , θ_{del} , and θ_{sub} .
4. Deletion (parameter δ). We examined the effects of changing the pressure toward deletion of classifiers with fitness below the average fitness.

We compared modifications on these configuration parameters with the default configuration C_P used in chapter 8, that is: $N=6400$, $F_0 = 0.99$, $\nu = 10$, $\{\theta_{GA}, \theta_{del}, \theta_{sub}\} = 50$, $\theta_{exploit} = 10$, $\chi = 0.8$, $\mu = 0.6$, $\delta=0.1$, and $P_{\#} = 0.6$. As done in chapter 8, we used the test accuracy and the size of the final rule set to evaluate the quality of the models. The results were statistically

analyzed as follows. First, we applied the multiple-comparison Friedman's test (Friedman, 1937, 1940) to contrast the hypothesis that the results of all learners were equivalent on average. If significant differences were found, the post-hoc Bonferroni-Dunn test (Dunn, 1961) was used to compare a control method against the others. Moreover, pairwise comparisons were applied according to the Wilcoxon signed-ranks test (Wilcoxon, 1945).

All results provided through this appendix are averages over ten runs with different random seeds. Due to the large number of configurations run for this analysis, we restricted the data set collection to twelve of the twenty real-world problems used in chapter 8: *bal*, *bpa*, *gls*, *h-s*, *irs*, *pim*, *tao*, *thy*, *veh*, *wbcd*, *wdbc*, and *wne*.

D.3 Fuzzy-UCS's Sensitivity to Configuration Parameters

This section analyzes in detail the effect of changing the parameters related to (1) rule initialization, (2) fitness pressure, (3) genetic algorithm, and (4) deletion. Each one of these analyses gets one of the subsequent subsections.

D.3.1 Sensitivity to Rule Initialization

Population initialization was identified as a crucial aspect for the success of LCSs and evolutionary algorithms in general (Goldberg, 2002). Butz et al. (2001) theoretically derived a covering bound for XCS indicating that the initial population should be general enough to cover all the training instances and permit the genetic algorithm to take place. The theoretical study resulted in practical guidelines suggesting that the initial generalization level (i.e., the parameter $P_{\#}$ in Fuzzy-UCS) be set to a high value. The same study showed that the best results on a set of artificial problems were obtained with $P_{\#} \approx 0.6$. For this reason, as many LCS practitioners have done during the last few years, we set $P_{\#} = 0.6$ in our experiments.

Herein, we empirically analyze the effect of decreasing the generalization in the initial population of Fuzzy-UCS. For this purpose, we ran Fuzzy-UCS with the default configuration C_P , but changing $P_{\#} = 0.2$ (C_1) and $P_{\#} = 0.4$ (C_2). Tables D.1 and D.2 show the performance and the rule set size achieved by Fuzzy-UCS with the three inference schemes and the three configurations.

The multiple-comparison Friedman's test did not permit rejecting the null hypothesis that the three configurations performed equivalently on average for each inference technique of Fuzzy-UCS at 95% confidence level. Nonetheless, note that configurations C_P and C_2 , that is, the configurations that use $P_{\#} = \{0.6, 0.4\}$ respectively, were the two best ranked configurations in all the inference schemes. A more detailed analysis on each particular problem permitted detecting in which problems a higher specificity on the initial population yielded more accurate models. For example, for the problems *thy*, and especially *gls*, Fuzzy-UCS obtained better results when the initial population was more specific (that is, $P_{\#}$ took a low value). On the other hand, for other problems such as *bal*, *h-s*, *pim*, *veh*, *wdbc*, and *wne*, Fuzzy-UCS presented better results with higher values of $P_{\#}$. Thus, we acknowledge that different settings of $P_{\#}$ may be used for different problems with particular characteristics and that further analysis has to be done to detect correlations between problem complexity and the setting of $P_{\#}$. Nevertheless, these results also evidence that it is safer to set $P_{\#}$ to higher values as a general rule of thumb.

Table D.1: Comparison of the test accuracy obtained with the three types of inference and the three configurations which vary $P_{\#}$. *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.

	wavg			awin			nfit		
	Cp	C1	C2	Cp	C1	C2	Cp	C1	C2
<i>bal</i>	88.65	86.90	88.63	84.40	74.24	82.66	83.40	73.53	82.89
<i>bpa</i>	59.82	59.19	60.82	59.42	57.16	58.54	58.93	56.72	56.67
<i>gls</i>	60.65	68.77	66.10	57.21	60.53	56.60	57.43	62.76	61.81
<i>h-s</i>	81.33	79.26	79.89	80.78	74.81	80.63	78.11	67.07	76.32
<i>irs</i>	95.67	95.00	94.93	95.47	95.53	95.73	93.73	94.67	93.40
<i>pim</i>	74.88	73.93	74.70	74.11	73.77	74.01	74.32	71.15	72.41
<i>tao</i>	81.71	81.10	81.31	83.02	82.91	83.08	87.53	89.00	88.46
<i>thy</i>	88.18	93.93	91.28	89.49	90.66	90.09	91.25	92.75	92.23
<i>veh</i>	67.68	67.29	68.67	65.35	66.65	67.02	65.34	63.81	65.65
<i>wbcd</i>	96.01	95.35	96.11	95.73	94.61	95.78	95.29	93.97	95.58
<i>wdbc</i>	95.20	93.55	95.31	94.61	91.40	93.64	94.51	90.05	93.62
<i>wne</i>	94.12	95.34	95.04	94.86	93.51	96.06	91.82	89.74	93.40
Rank	1.83	2.42	1.75	1.83	2.50	1.67	1.83	2.25	1.92
Pos	2	3	1	2	3	1	1	3	2
Frd	0.2053			0.097			0.558		

Table D.2: Comparison of the model sizes obtained with the three types of inference and the three configurations which vary $P_{\#}$. *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.

	wavg			awin			nfit		
	Cp	C1	C2	Cp	C1	C2	Cp	C1	C2
<i>bal</i>	1212	827	1580	114	128	122	75	105	60
<i>bpa</i>	1440	1714	1623	73	103	64	39	63	30
<i>gls</i>	2799	2551	3484	62	81	60	36	58	27
<i>h-s</i>	3415	2456	4002	117	139	122	62	93	62
<i>irs</i>	480	1217	634	18	26	19	7	10	6
<i>pim</i>	2841	1976	3407	192	251	183	62	141	42
<i>tao</i>	111	153	117	19	19	19	14	15	12
<i>thy</i>	1283	1838	1487	37	49	36	11	14	9
<i>veh</i>	3732	1776	4717	332	428	431	147	221	136
<i>wbcd</i>	3130	2305	4299	138	161	145	28	70	31
<i>wdbc</i>	5412	2724	5243	276	406	271	101	211	102
<i>wne</i>	3686	3695	4529	95	137	103	26	58	27
Rank	1.67	1.75	2.58	1.50	2.92	1.58	1.75	3.00	1.25
Pos	1	2	3	1	3	2	2	3	1
Frd	0.0458			0.0010			0.00006		

The rule set sizes evolved with the different configurations were not statistically equivalent according to the Friedman's test at a significance level of 0.05. To detect the significant differences among configurations, we applied the Nemenyi test with $\alpha = 0.1$ (that is, the critical difference is $CD=0.83$). The Nemenyi test identified the following three significant differences: (i) with weighted average inference, $C2$ resulted in significantly larger rule sets than Cp ; (ii) with action winner inference, $C1$ created significantly larger rule sets than the other two configurations; (iii) with most numerous and fittest rules inference, $C1$ built significantly larger models than the other two configurations. The last two points can be easily explained as follows. As $C1$ used a low value of $P_{\#}$, final populations contained more specific classifiers than populations created with Cp and $C2$. These two inference schemes only kept the classifiers that maximally matched an input instance in the final population. Thus, if classifiers were more specific, a larger number of them were set to the final population. On the other hand, with weighted average inference, the largest populations were obtained with $C2$. We hypothesize that this is because $C2$ created slightly general and accurate classifiers that coexisted in the population and partially covered the same training instances. Although the learning process pressured to obtain a minimum set of these classifiers, the evolved populations of $C2$ were bigger than those obtained with the other two inference schemes since these classifiers with partially overlapped conditions were maintained in the final population as they covered some training instances with maximum degree.

D.3.2 Sensitivity to Fitness Pressure

In Fuzzy-UCS, fitness pressure is determined by the parameter ν . This parameter biases the selection pressure toward the fittest classifiers. A few analyses have been conducted on the correct setting of this parameter in LCSs. Kharbat et al. (2005) showed that ν had a strong effect if proportionate selection was used in XCS and recommended to use values around 10 for this parameter. Similarly, Brown et al. (2007) empirically showed that $\nu = 10$ was an optimal setting for UCS in a set of artificial problems. Thus, in our experiments, we used $\nu = 10$.

As proceeds, we analyze the impact of decreasing ν on the quality of the models. To achieve this, tables D.3 and D.4 show the accuracies and sizes of the evolved models for $\nu = 1$ ($C3$) and $\nu = 5$ ($C4$). The statistical analysis indicated that the accuracy of the models was not equivalent on average according to the multiple-comparison Friedman's test. Therefore, we applied the Nemenyi test (at $\alpha = 0.10$) to detect significant differences among learners (the critical distance is $CD = 0.83$). The test identified that, for all inference schemes, Fuzzy-UCS evolved more accurate models with Cp than with $C3$. Besides, the models created with Cp held the first position of the ranking, and the models built with $C4$ held the second position of the ranking for any inference level. Thus, these results evidenced that larger values of ν yielded more accurate models. We applied a pairwise comparison between Cp and $C4$ according to a Wilcoxon signed-ranks test, which detected, with $p = 0.02$, that the models evolved with Cp were significantly more accurate than those created with $C4$.

Conclusions on the model sizes depend on the used inference scheme. For weighted average inference, $C3$ created significantly larger models than the other configurations, and $C4$ evolved significantly larger models than Cp according to a Bonferroni-Dunn test at $\alpha = 0.1$. So, the higher the fitness pressure was, the smaller the final models were. This evidenced that setting high values of ν is crucial to remove over-general classifiers in favor of highly-fit classifiers.

Table D.3: Comparison of the test accuracy obtained with the three types of inference and the three configurations which vary the fitness pressure ν . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.

	wavg			awin			nfit		
	Cp	C3	C4	Cp	C3	C4	Cp	C3	C4
<i>bal</i>	88.65	83.31	87.87	84.40	80.11	83.90	83.40	63.33	82.49
<i>bpa</i>	59.82	57.94	59.08	59.42	58.42	58.50	58.93	56.45	60.23
<i>gls</i>	60.65	55.55	58.23	57.21	52.60	56.92	57.43	48.45	56.06
<i>h-s</i>	81.33	83.52	82.03	80.78	81.89	81.53	78.11	78.59	78.53
<i>irs</i>	95.67	93.13	95.07	95.47	94.80	95.07	93.73	82.60	92.27
<i>pim</i>	74.88	71.30	74.61	74.11	72.07	73.38	74.32	71.81	73.80
<i>tao</i>	81.71	79.18	80.90	83.02	82.57	83.36	87.53	79.05	86.68
<i>thy</i>	88.18	78.67	85.88	89.49	87.42	89.02	91.25	86.18	89.11
<i>veh</i>	67.68	61.12	67.23	65.35	61.22	64.77	65.34	57.59	65.79
<i>wbcd</i>	96.01	95.33	95.50	95.73	94.90	95.42	95.29	92.39	94.40
<i>wdbc</i>	95.20	94.68	94.96	94.61	93.99	94.11	94.51	92.88	94.00
<i>wne</i>	94.12	93.27	94.49	94.86	94.98	95.20	91.82	85.29	93.92
Rank	1.25	2.83	1.92	1.42	2.75	1.83	1.42	2.83	1.75
Pos	1	3	2	1	3	2	1	3	2
Frd	0.00050			0.00380			0.00140		

Table D.4: Comparison of the model sizes obtained with the three types of inference and the three configurations which vary the fitness pressure ν . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.

	wavg			awin			nfit		
	Cp	C3	C4	Cp	C3	C4	Cp	C3	C4
<i>bal</i>	1212	2371	1580	114	63	122	75	8	60
<i>bpa</i>	1440	2960	1623	73	47	64	39	12	30
<i>gls</i>	2799	4484	3484	62	52	60	36	17	27
<i>h-s</i>	3415	5469	4002	117	115	122	62	31	62
<i>irs</i>	480	1334	634	18	20	19	7	5	6
<i>pim</i>	2841	4166	3407	192	132	183	62	22	42
<i>tao</i>	111	149	117	19	16	19	14	5	12
<i>thy</i>	1283	2122	1487	37	32	36	11	7	9
<i>veh</i>	3732	5709	4717	332	242	431	147	84	136
<i>wbcd</i>	3130	4992	4299	138	137	145	28	26	31
<i>wdbc</i>	5412	5839	5243	276	255	271	101	92	102
<i>wne</i>	3686	5514	4529	95	97	103	26	29	27
Rank	1.08	3.00	1.92	2.33	1.25	2.42	2.67	1.17	2.17
Pos	1	3	2	2	1	3	3	1	2
Frd	0.00001			0.0052			0.0010		

For action winner inference, the statistical analysis only detected that the models evolved with *C3* were significantly smaller than the models created with the other two configurations. This might be due to the presence of over-general classifiers with moderate fitness in the final populations, which had not been removed due to the poor genetic pressure toward highly fit classifiers. These classifiers replaced more specific but fitter ones in the final population. The same behavior could be observed for most numerous and fittest rules inference, where Fuzzy-UCS with configuration *C3* created significantly smaller models than with the other two configurations.

D.3.3 Sensitivity to the GA

In this section, we analyze the sensitivity of Fuzzy-UCS to the parameters concerning the genetic algorithm application: θ_{GA} , θ_{del} , and θ_{sub} . The parameter θ_{GA} is a threshold that controls the application period of the GA on the different correct sets [C] of the system. That is, a correct set will receive a genetic event if the average time since the last application of the GA on this correct set exceeds θ_{GA} . If we want to maximize the genetic discovery, and so, the learning rate, θ_{GA} should be set to zero. In this case, a correct set would receive a genetic event every time it is activated. However, note that the parameters of new classifiers are incrementally updated as these classifiers participate in successive correct sets. So, if we apply a genetic algorithm to each correct set, selection would be biased since there would be poorly evaluated classifiers in the niches. Moreover, this would also imply the generation of a large number of new classifiers with poorly evaluated parameters. For this reason, θ_{GA} should be set to a higher value in real-world problems. In the configuration used in the paper, we set $\theta_{GA} = 50$, since this corresponds to the standard value used for the equivalent parameter in XCS and UCS (Bernadó-Mansilla et al., 2002; Bernadó-Mansilla and Garrell, 2003; Butz, 2006; Orriols-Puig and Bernadó-Mansilla, 2008b; Dixon et al., 2002, 2004; Fu et al., 2001; Wilson, 2000).

The values of the θ_{del} and θ_{sub} parameters are usually determined by θ_{GA} . If we consider that the classifiers in a niche need to receive an average of θ_{GA} parameter updates before going through a genetic event, intuitively this should also apply for deletion and subsumption. For this reason, in the configuration used in chapter 8, we set $\theta_{del} = \theta_{sub} = \theta_{GA}$.

To analyze the sensitivity of Fuzzy-UCS to this set of parameters, we performed the following experiments. In configurations *C5* and *C6*, we incremented θ_{GA} , θ_{del} , and θ_{sub} to 100 and 200 respectively. As we decreased the application rate of the genetic algorithm, we expected to obtain a lower accuracy in the final models. Moreover, to confirm the stability of the system, we ran *C7* and *C8*, two other configurations of Fuzzy-UCS. *C7* equaled *C5* except for the number of iterations, $numIter=200\,000$. *C8* equaled *C6* except for the number of iterations, $numIter=400\,000$. Therefore, we guaranteed that the number of genetic events received by the niches with configurations *C7* and *C8* was equivalent to the number of genetic events received with configuration *Cp*. Thus, we expected to obtain similar results.

Tables D.5 and D.6 show the accuracies and sizes of the models for the different configurations. The multiple-comparison Friedman's test rejected the null hypothesis that all the configurations performed the same on average for a particular inference scheme at 95% confidence level. As configuration *Cp* is the best ranked in all cases, we used the Bonferroni-Dunn test to detect which configurations performed worse than *Cp* at $\alpha = 0.10$ (the critical difference is $CD = 1.44$). The statistical test detected that: (i) with weighted average and most numerous and fittest rules, inferences *Cp*, *C7*, and *C8* performed equivalently, whilst *C5* and

Table D.5: Comparison of the test accuracy obtained with the three types of inference and the five configurations varying θ_{GA} , θ_{del} , and θ_{sub} . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.

	wavg					awin					nfit				
	Rp	C5	C6	C7	C8	Rp	C5	C6	C7	C8	Rp	C5	C6	C7	C8
<i>bal</i>	88.65	86.88	86.35	88.52	88.50	84.40	83.95	82.82	84.25	83.33	83.40	82.46	80.85	83.59	83.42
<i>bpa</i>	59.82	58.38	59.18	59.76	59.17	59.42	57.99	57.57	58.64	57.16	58.93	58.76	60.75	57.32	57.07
<i>gls</i>	60.65	56.78	55.75	61.79	64.17	57.21	56.57	55.59	57.30	56.63	57.43	54.77	53.50	57.91	58.96
<i>h-s</i>	81.33	81.04	81.81	80.75	79.79	80.78	80.89	80.78	80.40	80.71	78.11	78.52	79.30	77.57	77.08
<i>irs</i>	95.67	94.73	94.40	94.87	95.00	95.47	95.73	94.73	95.20	95.67	93.73	93.40	92.60	93.73	94.40
<i>pim</i>	74.88	74.00	72.17	74.75	74.74	74.11	73.17	71.81	73.83	74.00	74.32	73.71	72.78	73.25	72.62
<i>tao</i>	81.71	81.67	80.98	81.79	82.05	83.02	82.91	82.17	82.95	82.97	87.53	84.96	82.13	87.54	86.97
<i>thy</i>	88.18	86.89	85.41	88.18	89.65	89.49	89.45	87.93	89.81	90.00	91.25	89.34	88.28	90.20	91.74
<i>veh</i>	67.68	65.64	64.10	67.89	67.41	65.35	64.75	63.40	65.92	65.23	65.34	64.11	62.13	65.82	64.69
<i>wbcd</i>	96.01	95.65	95.14	95.82	95.74	95.73	95.43	94.96	95.76	95.92	95.29	94.73	94.26	95.29	95.14
<i>wdbc</i>	95.20	95.12	94.96	95.04	95.28	94.61	94.41	93.99	94.87	94.71	94.51	94.25	94.09	94.49	94.15
<i>wne</i>	94.12	93.50	94.73	95.12	94.91	94.86	94.08	94.46	95.24	94.93	91.82	90.33	90.88	92.09	92.86
Rnk	1.92	4.00	4.33	2.33	2.42	2.13	3.42	4.63	2.25	2.58	2.13	3.58	4.17	2.29	2.83
Pos	1	4	5	2	3	1	4	5	2	3	1	4	5	2	3
Frd	0.00014					0.00035					0.006				

C6 presented significantly poorer results; and (ii) with action winner, Cp, C5, C7, and C8 had the same accuracy on average, while C6 showed the poorest results. Further statistical analysis by means of pairwise comparisons supported these conclusions and, moreover, detected that C5 degraded the results obtained with Cp, C7, and C8 with action winner inference (see table D.7). Therefore, all this statistical study supported the initial hypothesis: as the number of genetic events decreases, the evolved models are less accurate.

The statistical analysis on the model sizes only identified significant differences for weighted average inference. In this case, the post-hoc Bonferroni-Dunn test detected that configuration C6 evolved larger models than the other configurations. This is because, with configuration C6, the correct sets received the lowest number of genetic events; therefore, the population had less diversity.

D.3.4 Sensitivity to Deletion

The deletion mechanism designed for Fuzzy-UCS was inspired by the initial deletion procedure of XCS (Kovacs, 1999). This schema increases the probability of deletion of experienced classifiers whose fitness is less than δ times the average fitness of the population. So, varying δ results in changing the pressure toward deletion of classifiers with low fitness. Nonetheless, recent studies have shown that XCS is not sensitive to the settings of δ (Kovacs and Bull, 2007). To confirm this statement, we ran XCS with $\delta = 1$ (configuration C9).

Table D.8 and D.9 compare the models accuracies and sizes of Fuzzy-UCS with configurations Cp and C9. We applied a pairwise comparison between the two configurations for each inference

D.3. FUZZY-UCS'S SENSITIVITY TO CONFIGURATION PARAMETERS

Table D.6: Comparison of the model sizes obtained with the three types of inference and the five configurations varying θ_{GA} , θ_{del} , and θ_{sub} . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *Frd* reports the p-value obtained with the multiple-comparison Friedman test performed for each inference methodology.

	wavg					awin					nfit				
	Rp	C5	C6	C7	C8	Rp	C5	C6	C7	C8	Rp	C5	C6	C7	C8
<i>bal</i>	1212	1233	1164	1096	1002	114	117	119	115	114	75	71	63	80	84
<i>bpa</i>	1440	1320	934	1519	1607	73	60	52	74	73	39	24	20	43	43
<i>gls</i>	2799	2684	2528	2835	2926	62	62	60	59	59	36	31	28	39	41
<i>h-s</i>	3415	3505	3273	3449	3396	117	133	137	113	107	62	68	67	59	58
<i>irs</i>	480	540	378	495	482	18	18	18	17	17	7	8	8	7	7
<i>pim</i>	2841	2539	2072	2707	2686	192	179	161	193	181	62	48	41	79	87
<i>tao</i>	111	143	117	107	101	19	18	17	19	19	14	13	10	14	14
<i>thy</i>	1283	1166	780	1266	1259	37	37	34	38	36	11	11	11	10	10
<i>veh</i>	3732	3859	3981	3581	3498	332	319	310	326	317	147	143	116	169	199
<i>wbcd</i>	3130	3184	2477	3097	3111	138	148	150	138	140	28	33	36	28	28
<i>wdbc</i>	5412	5343	5037	5415	5412	276	279	272	275	269	101	88	83	112	115
<i>wne</i>	3686	3568	3241	3746	3764	95	101	100	96	95	26	29	30	25	24
Rank	3.42	3.42	1.75	3.42	3.00	3.17	3.50	2.83	3.33	2.17	3.17	3.50	2.83	3.33	2.17
Pos	4	4	1	4	2	3	5	2	4	1	3	5	2	4	1
Frd	0.0368					0.1466					0.6289				

Table D.7: Pairwise comparison of the test accuracy of Fuzzy-UCS obtained with the three types of inference and the five configurations varying θ_{GA} , θ_{del} , and θ_{sub} by means of a Wilcoxon signed-ranks test.

	wavg					awin					nfit				
	Cp	C5	C6	C7	C8	Cp	C5	C6	C7	C8	Cp	C5	C6	C7	C8
Cp		.002	.008	.695	.938		.012	.003	.875	.480		.006	.034	.424	.530
C5	⊖		.182	.010	.012	⊖		.004	.041	.347	⊖		.100	.084	.182
C6	⊖	−		.012	.015	⊖	⊖		.003	.005	⊖	−		.060	.100
C7	−	⊕	⊕		.938	+	⊕	⊕		.327	−	+	+		1.00
C8	−	⊕	⊕	−		−	+	⊕	+		−	+	+	−	

scheme according to a Wilcoxon signed-ranks test (the approximate p-value is provided in the last row of the tables). The null hypothesis that the results obtained with both configurations were equal on average could not be rejected. This supported the empirical conclusions extracted by Kovacs and Bull (2007), which highlighted the robustness of XCS (and Fuzzy-UCS in our case) to the parameter δ .

Table D.8: Comparison of the test accuracy obtained with the three types of inference and the two configurations which vary the deletion pressure δ . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *PW* reports the p-value obtained with the pairwise Wilcoxon signed-ranks test performed for each inference methodology.

	wavg		awin		nfit	
	Cp	C9	Cp	C9	Cp	C9
<i>bal</i>	88.65	88.68	84.40	84.30	83.40	82.91
<i>bpa</i>	59.82	59.61	59.42	58.37	58.93	58.32
<i>gls</i>	60.65	59.85	57.21	56.70	57.43	58.21
<i>h-s</i>	81.33	80.81	80.78	80.78	78.11	79.44
<i>irs</i>	95.67	95.13	95.47	95.60	93.73	94.40
<i>pim</i>	74.88	74.70	74.11	74.19	74.32	74.11
<i>tao</i>	81.71	81.64	83.02	83.11	87.53	87.79
<i>thy</i>	88.18	89.04	89.49	90.29	91.25	89.47
<i>veh</i>	67.68	66.50	65.35	65.84	65.34	65.59
<i>wbcd</i>	96.01	95.59	95.73	95.52	95.29	95.03
<i>wdbc</i>	95.20	95.09	94.61	94.43	94.51	94.19
<i>wne</i>	94.12	94.58	94.86	94.97	91.82	92.94
Rank	1.25	1.75	1.54	1.46	1.5	1.5
Pos	1	2	2	1	1.5	1.5
PW	0.1099		0.8311		0.7334	

Table D.9: Comparison of the model sizes obtained with the three types of inference and the three configurations which vary the deletion pressure δ . *Rank* gives the average rank of each configuration for each one of the three inference schemes. *Pos* shows the absolute position in the ranking. *PW* reports the p-value obtained with the pairwise Wilcoxon signed-ranks test performed for each inference methodology.

	wavg		awin		nfit	
	Cp	C9	Cp	C9	Cp	C9
<i>bal</i>	1212	1310	114	109	75	67
<i>bpa</i>	1440	1437	73	74	39	40
<i>gls</i>	2799	2869	62	60	36	35
<i>h-s</i>	3415	3450	117	116	62	60
<i>irs</i>	480	492	18	17	7	7
<i>pim</i>	2841	2765	192	188	62	66
<i>tao</i>	111	107	19	19	14	14
<i>thy</i>	1283	1276	37	37	11	11
<i>veh</i>	3732	3741	332	321	147	139
<i>wbcd</i>	3130	3385	138	135	28	28
<i>wdbc</i>	5412	5439	276	277	101	99
<i>wne</i>	3686	3808	95	97	26	25
Rank	1.33	1.67	1.75	1.25	1.75	1.25
Pos	1	2	2	1	2	1
PW	0.064		0.0977		0.1719	

D.4 Summary and Conclusions

The study performed in this chapter empirically showed that there are two key parameters to guarantee the success of Fuzzy-UCS: generalization in initialization ($P_{\#}$) and fitness pressure (ν). Specifically, the generality in the initial population has to be high enough to let the genetic algorithm take place, as suggested by Butz et al. (2001). Moreover, the fitness pressure should be high enough to ensure a strong and reliable pressure toward the fittest classifiers in the population. On the other hand, changing the setting of the other parameters appears to have little effect on the model's quality.

We acknowledge that better results could be individually obtained if we tuned Fuzzy-UCS for each particular problem. Nonetheless, we are interested in robust systems that perform well on average. For this reason we did not consider to tune the system for each problem in the experiments conducted in chapter 8.

Bibliography

- C. Aggarwal, editor. *Data streams: Models and algorithms*. Springer, 2007.
- J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro. Evolutionary learning of hierarchical decision rules. *IEEE Transactions on Systems, Man, and Cybernetics—Part B*, 33(2):324–331, 2003.
- J. S. Aguilar-Ruiz, R. Giráldez, and J. C. Riquelme. Natural encoding for evolutionary supervised learning. *IEEE Transactions on Evolutionary Computation*, 11(4):466–479, 2007.
- D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, 1991.
- R. Alcalá, J. Casillas, O. Cordón, and F. Herrera. Building fuzzy graphs: Features and taxonomy of learning for non-grid-oriented fuzzy rule-based systems. *Journal of Intelligent and Fuzzy Systems*, 11(3-4):99–119, 2001.
- J. Alcalá-Fdez, F. Herrera, F. Márquez, and A. Peregrín. Increasing fuzzy rules cooperation based on evolutionary adaptive inference systems: Research articles. *International Journal in Intelligent Systems*, 22(9):1035–1064, 2007. ISSN 0884-8173. doi: <http://dx.doi.org/10.1002/int.v22:9>.
- J. Alcalá-Fdez, L. Sánchez, S. García, M. J. del Jesus, S. Ventura, J. M. Garrell, J. Otero, C. Romero, J. Bacardit, V. M. Rivas, J. C. Fernández, and F. Herrera. KEEL: A software tool to assess evolutionary algorithms for data mining problems. *Soft Computing*, doi=10.1007/s00500-008-0323-y, 2008.
- D. L. Applegate, R. E. Bixby, V. Chvátal, and W. J. Cook. *The traveling salesman problem: A computational Study*. Princeton University Press, Princeton, NJ, USA, 2006.
- A. Arasu, B. Babcock, S. Babu, M. Datar, K. Ito, I. Nishizawa, J. Rosenstein, and J. Widom. STREAM: The stanford stream data manager demonstration description - short overview of system status and plans. In *SIGMOD'03: Proceedings of the ACM International Conference on Management of Data*, 2003.
- A. Asuncion and D. J. Newman. *UCI Machine Learning Repository: [<http://www.ics.uci.edu/~mllearn/MLRepository.html>]*. University of California, 2007.
- J. Bacardit. *GAssist Source Code: <http://www.asap.cs.nott.ac.uk/~jqb/PSP/GAssist-Java.tar.gz>*, 2007.

- J. Bacardit. *Pittsburgh genetic-based machine learning in the data mining era: Representations, generalization and run-time*. PhD thesis, Ramon Llull University, Barcelona, Catalonia, Spain, Barcelona, 2004.
- J. Bacardit and M. V. Butz. Data mining in learning classifier systems: Comparing XCS with GAssist. In *Proceedings of the 7th International Workshop on Learning Classifier Systems*. Springer-Verlag, 2004.
- T. Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 57–62, 1994.
- T. Bäck. Generalized convergence models for tournament- and (μ , λ)-selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 2–8. Morgan Kaufmann Publishers Inc., 1995. ISBN 1-55860-370-0.
- T. Bäck. *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.
- J. E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pages 101–111, 1985.
- A. Bardossy and L. Duckstein. *Fuzzy rule-based modeling with applications to geophysical, biological, and engineering systems*. CRC Press, Inc., Boca Raton, FL, USA, 1995. ISBN 0849378338.
- G. Batista, R. C. Prati, and M. C. Monrad. A study of the behavior of several methods for balancing machine learning training data. *SIGKDD Explorations Newsletter, special issue on learning from imbalanced datasets*, 6(1):20–29, 2004.
- E. Bernadó-Mansilla and J. M. Garrell. Accuracy-based learning classifier systems: Models, analysis and applications to classification tasks. *Evolutionary Computation*, 11(3):209–238, 2003.
- E. Bernadó-Mansilla and T. Ho. Domain of competence of XCS classifier system in complexity measurement space. *IEEE Transactions on Evolutionary Computation*, 9(1):1–23, 2005.
- E. Bernadó-Mansilla, X. Llorà, and J. Garrell. XCS and GALE: A comparative study of two learning classifier systems on data mining. In *Advances in Learning Classifier Systems*, volume 2321 of *LNAI*, pages 115–132. Springer, 2002.
- E. Bernadó-Mansilla, T. Ho, and A. Orriols-Puig. *Data Complexity and Evolutionary Learning: Classifier’s Behavior and Domain of Competence*, pages 115–134. Springer, 2006.
- H. G. Beyer. Toward a theory of evolution strategies: Self-adaptation. *Evolutionary Computation*, 3(3):311–347, 1996.
- C. M. Bishop. *Pattern recognition and machine learning*. Springer, 2007. ISBN 0-387-31073-8.
- T. Blickle and L. Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, 1995.

- T. Blickle and L. Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996.
- A. Bonarini. Evolutionary learning of fuzzy rules: Competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Norwell, MA: Kluwer Academic Press, 1996.
- A. Bonarini and V. Trianni. Learning fuzzy classifier systems for multi-agent coordination. *Information Sciences: an International Journal*, 136(1-4):215–239, 2001. ISSN 0020-0255.
- P. Bonelli and A. Parodi. An efficient classifier system and its experimental comparison with two representative learning methods on three medical domains. In *4th International Conference on Genetic Algorithms*, pages 288–295, 1991.
- L. Booker. *Intelligent behavior as an adaptation to the task environment*. PhD thesis, University of Michigan, 1982.
- G. E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6:81–101, 1957.
- G. E. P. Box and N. P. Draper. *Evolutionary operation. A method for increasing industrial productivity*. New York: Wiley, 1969.
- H. J. Bremermann. Optimization through evolution and recombination. In *Self-Organizing Systems*. Pergamon Press, Oxford, U.K., 1962.
- R. A. Brooks. Elephants don't play chess. *Robotics and Autonomous Systems*, 6:3–15, 1990.
- G. Brown, T. Kovacs, and J. Marshall. UCSpv: Principled voting in UCS rule populations. In *GECCO'07: Proceedings of the 2007 Conference on Genetic and Evolutionary Computation*, pages 1774–1781, New York, NY, USA, 2007. ACM.
- B. G. Buchanan. A (very) brief history of artificial intelligence. *AI Magazine*, 26(4):53–60, 2005.
- J. J. Buckley and Y. Hayashi. Fuzzy neural networks: A survey. *Fuzzy Sets and Systems*, 66: 1–13, 1994.
- J. J. Buckley and Y. Hayashi. Neural networks for fuzzy systems. *Fuzzy Sets and Systems*, 71: 265–276, 1995.
- L. Bull. *Applications of learning classifier systems*. Springer Verlag, 2004.
- L. Bull. Two simple learning classifier systems. In *Foundations of Learning Classifier Systems*, volume 183/2005, pages 63–89. Springer Berlin, 2005.
- L. Bull and J. Hurst. ZCS redux. *Evolutionary Computation*, 10(2):185–205, 2002.
- L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *GECCO'02: Proceedings of the 2002 Genetic and Evolutionary Computation Conference*, pages 905–911, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-878-8.

- L. Bull, E. Bernadó-Mansilla, and J. Holmes, editors. *Learning classifier systems in data mining*. Studies in Computational Intelligence. Springer, 2008.
- M. Butz, D. E. Goldberg, P. L. Lanzi, and K. Sastry. Problem solution sustenance in XCS: Markov chain analysis of niche support distributions and the impact on computational complexity. *Genetic Programming and Evolvable Machines*, 8(1):5–37, 2007.
- M. V. Butz. *Rule-based evolutionary online learning systems: A principled approach to LCS analysis and design*, volume 109 of *Studies in Fuzziness and Soft Computing*. Springer, 2006.
- M. V. Butz and M. Pelikan. Analyzing the evolutionary pressures in XCS. In *GECCO'01: Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 935–942. San Francisco, CA: Morgan Kaufmann, 2001.
- M. V. Butz and S. W. Wilson. An algorithmic description of XCS. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems: Proceedings of the Third International Workshop*, volume 1996 of *Lecture Notes in Artificial Intelligence*, pages 253–272. Springer, 2001.
- M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. How XCS evolves accurate classifiers. In *GECCO'01: Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 927–934. San Francisco, CA: Morgan Kaufmann, 2001.
- M. V. Butz, D. E. Goldberg, and T. Tharankunnel. Analysis and improvement of fitness exploration in XCS: Bounding models, tournament selection, and bilateral accuracy. *Evolutionary Computation*, 11(3):239–277, 2003.
- M. V. Butz, D. E. Goldberg, and P. L. Lanzi. Bounding learning time in XCS. In *GECCO'2004: Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, LNCS, pages 739–750. Springer, 2004a.
- M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson. Toward a theory of generalization and learning in XCS. *IEEE Transactions on Evolutionary Computation*, 8(1):28–46, 2004b.
- M. V. Butz, D. E. Goldberg, and P. L. Lanzi. Gradient descent methods in learning classifier systems: Improving XCS performance in multistep problems. *IEEE Transactions on Evolutionary Computation*, 9(5):452–473, 2005a.
- M. V. Butz, D. E. Goldberg, and P. L. Lanzi. Extracted global structure makes local building block processing effective in XCS. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'2005)*, pages 655–662. ACM Press, 2005b.
- M. V. Butz, K. Sastry, and D. E. Goldberg. Strong, stable, and reliable fitness pressure in XCS due to tournament selection. *Genetic Programming and Evolvable Machines*, 6(1):53–77, 2005c.
- M. V. Butz, P. L. Lanzi, and S. W. Wilson. Hyper-ellipsoidal conditions in XCS: Rotation, linear approximation, and solution structure. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1457–1464, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: <http://doi.acm.org/10.1145/1143997.1144237>.

- M. V. Butz, P. L. Lanzi, and S. W. Wilson. Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. *IEEE Transactions on Evolutionary Computation*, 12(3):355–376, 2008. doi: 10.1109/TEVC.2007.903551.
- E. Cantú-Paz. Selection intensity in genetic algorithms with generation gaps. In *GECCO'00: Proceedings of the 2000 Genetic and Evolutionary Computation Conference*, pages 911–918, 1999a.
- E. Cantú-Paz. Migration policies and takeover times in parallel genetic algorithms. In *GECCO'99: Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 775–775, 1999b. also IlliGAL Report No. 99008.
- B. Carse, T. Fogarty, and A. Munro. Evolving fuzzy rule based controllers using genetic algorithms. *Fuzzy Sets and Systems*, 80(3):273–293, 1996. ISSN 0165-0114.
- D. R. Carvalho and A. A. Freitas. A genetic-algorithm for discovering small-disjunct rules in data mining. *Applied Soft Computing*, 2(2):75–88, 2002.
- J. Casillas, O. Cordon, M. J. del Jesus, and F. Herrera. Genetic tuning of fuzzy rule deep structures preserving interpretability and its interaction with fuzzy rule set reduction. *IEEE Transactions on Fuzzy Systems*, 13(1):13–29, 2005. ISSN 1063-6706. doi: 10.1109/TFUZZ.2004.839670.
- J. Casillas, B. Carse, and L. Bull. Fuzzy-XCS: A Michigan genetic fuzzy system. *IEEE Transactions on Fuzzy Systems*, 15(4):536–550, 2007.
- P. K. Chan and S. J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Knowledge Discovery and Data Mining*, pages 164–168, 1998.
- N. V. Chawla, K. Bowyer, L. O. Hall, and W. Kegelmeyer. SMOTE: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.
- N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer. SMOTEBoost: Improving prediction of the minority class in boosting. In *VIIth European Conference on Principles and Practice of Knowledge Discovery in Databases(PKDD03)*, pages 107–119. Springer-Verlag, 2003.
- N. V. Chawla, N. Japkowicz, and A. Kolcz, editors. *Special issue on learning from imbalanced datasets*, volume 6. 2004.
- O. Cordon and F. Herrera. A three-stage evolutionary process for learning descriptive and approximate fuzzy-logic-controller knowledge bases from examples. *International Journal of Approximate Reasoning*, 17(4):369–407, 1997.
- O. Cordon, M. del Jesús, and F. Herrera. A proposal on reasoning methods in fuzzy rule-based classification systems. *International Journal of Approximate Reasoning*, 20(1):21–45, 1999.
- O. Cordon, F. Herrera, F. Hoffmann, and L. Magdalena. *Genetic fuzzy systems: Evolutionary tuning and learning of fuzzy knowledge bases*, volume 19 of *Advances in Fuzzy Systems—Applications and Theory*. World Scientific, 2001a.

- O. Cordón, F. Herrera, and P. Villar. Generating the knowledge base of a fuzzy rule-based system by the genetic learning of the data base. *IEEE Transactions on Fuzzy Systems*, 9(4): 667–674, August 2001b.
- K. Crockett, Z. Bandar, and D. Mclean. On the optimization of t-norm parameters within fuzzy decision trees. In *IEEE International Conference on Fuzzy Systems (FUZZ-IEEE 2007)*, pages 103–108, 2007. doi: 10.1109/FUZZY.2007.4295348.
- K. A. Crockett, Z. Bandar, J. Fowdar, and J. O’Shea. Genetic tuning of fuzzy inference within fuzzy classifier systems. *Expert Systems*, 23:63–82, 2006. doi: doi:10.1111/j.1468-0394.2006.00325.x.
- H. H. Dam, H. A. Abbass, and C. Lokan. Be real! XCS with continuous-valued inputs. In *GECCO’05: In Proceedings of the 2005 Genetic and Evolutionary Computation Conference workshop program*, pages 85–87, Washington, D.C., USA, 2005. ACM Press.
- C. Darwin. *The origin of species*. see online version at www.literature.org, 1859.
- K. Deb. Genetic algorithms in multimodal function optimization. In *Master Thesis and TCGA Report No. 89002. Tuscaloosa, AL: Department of Engineering Mechanics, University of Alabama*, 1989.
- M. J. del Jesus, F. Hoffmann, L. J. Navascués, and L. Sánchez. Induction of fuzzy-rule-based classifiers with evolutionary boosting algorithms. *IEEE Transactions on Fuzzy Systems*, 12(3):296–308, 2004.
- J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.
- A. V. den Bosch, T. Weijters, and J. V. den Herik. When small disjuncts abound, try lazy learning: A case study. In *Proceedings Seventh BENELEARN Conference*, pages 109–118, 1997.
- T. G. Dietterich. Approximate statistical tests for comparing supervised classification learning algorithms. *Neural Computation*, 10(7):1895–1924, 1998.
- P. W. Dixon, D. W. Corne, and M. J. Oates. A preliminary investigation of modified XCS as a generic data mining tool. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems, 4th International Workshop*, volume 2321 of *Lecture Notes in Computer Science*, pages 133–150. Springer, 2002.
- P. W. Dixon, D. W. Corne, and M. J. Oates. *A ruleset reduction algorithm for the XCSI learning classifier system*, volume 2661/2003 of *Lecture Notes in Computer Science*, pages 20–29. Springer, 2004. ISBN 978-3-540-20544-9.
- J. Drugowitsch. *Design and analysis of learning classifier systems: A probabilistic approach*. Springer, 2008.
- J. Drugowitsch and A. M. Barry. A formal framework and extensions for function approximation in learning classifier systems. *Machine Learning*, 70(1):45–88, 2008.

- R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification*. Wiley-Interscience, 2nd edition, 2000.
- O. Dunn. Multiple comparisons among means. *Journal of the American Statistical Association*, 56:52–64, 1961.
- T. Fawcett. PRIE: A system for generating rulelists to maximize ROC performance. *Data Mining and Knowledge Discovery*, 17(2):207–224, 2008. doi: 10.1007/s10618-008-0089-y.
- E. A. Feigenbaum and J. Feldman, editors. *Computers and Thought*. AAAI press, 1995.
- R. Fisher. *Statistical methods and scientific inference*. Hafner Publishing Co, New York, 2nd edition, 1959.
- D. B. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.
- D. B. Fogel. *On the organization of intellect*. PhD thesis, 1964.
- D. B. Fogel, A. J. Owens, and M. J. Walsh. *Artificial intelligence through simulated evolution*. Wiley, New York, USA, 1966.
- E. Frank and I. H. Witten. Generating accurate rule sets without global optimization. In *Proceedings of the 15th International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann, San Francisco, CA, 1998.
- W. Frawley, G. Piatetsky-Shapiro, and C. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 213–228, 1992.
- A. Freitas. *Data mining and knowledge discovery with evolutionary algorithms*. Springer-Verlag, 2002.
- Y. Freund and R. E. Schapire. Experiments with a new boosting algorithm. In *International Conference on Machine Learning*, pages 148–156, 1996.
- R. M. Friedberg. A learning machine: part I. *IBM Journal*, 2:2–13, 1958.
- R. M. Friedberg, B. Dunham, and J. H. North. A learning machine: part II. *IBM Journal*, 3: 282–287, 1959.
- M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32:675–701, 1937.
- M. Friedman. A comparison of alternative tests of significance for the problem of m rankings. *Annals of Mathematical Statistics*, 11:86–92, 1940.
- C. Fu, S. W. Wilson, and L. Davis. Studies of the XCSI classifier system on a data mining problem. In *GECCO’01: Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 985–993, San Francisco, CA, USA, 2001. Morgan Kaufmann.
- T. Furuhashi, K. Nakaoka, and Y. Uchikawa. Suppression of excess fuzziness using multiple fuzzy classifier systems. In *Proceedings of the 3th IEEE International Conference on Fuzzy Systems*, pages 411–414. Morgan Kaufmann, 1994.

- J. Gama and M. M. Gaber, editors. *Learning from data streams*. Springer, 2007.
- S. García and F. Herrera. Evolutionary under-sampling for classification with imbalanced data sets: Proposals and taxonomy. *Evolutionary Computation*, 2008.
- A. Giordana and F. Neri. Search-intensive concept induction. *Evolutionary Computation*, 3(4): 375–419, 1995.
- H. Goksu, P. Pigg, and V. Dixit. Music Composition Using Genetic Algorithms (GA) and Multilayer Perceptrons (MLP). In *Advances in Natural Computation*, volume 3612 of *Lectures Notes in Computer Science*, pages 1242–1250, 2005.
- D. E. Goldberg. *The design of innovation: Lessons from and for competent genetic algorithms*. Kluwer Academic Publishers, 1 edition, 2002.
- D. E. Goldberg. Controlling dynamic systems with genetic algorithms and rule learning. In *Proceedings of the 4th Yale Workshop on Applications and Adaptive Systems Theory*, pages 91–97, 1985a.
- D. E. Goldberg. Dynamic system control using rule learning and genetic algorithms. In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, volume 1, pages 588–592, 1985b.
- D. E. Goldberg. Computer-aided gas pipeline operation using genetic algorithms and rule learning. part I: Genetic algorithms in pipeline optimization. *Engineering with Computers*, pages 35–45, 1987a.
- D. E. Goldberg. Computer-aided gas pipeline operation using genetic algorithms and rule learning. part II: Rule learning control of a pipeline under normal and abnormal conditions. *Engineering with Computers*, pages 47–58, 1987b.
- D. E. Goldberg. *Genetic algorithms in search, optimization & machine learning*. Addison Wesley, 1 edition, 1989a.
- D. E. Goldberg. Sizing populations for serial and parallel genetic algorithms. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 70–79, 1989b.
- D. E. Goldberg. *Computer-aided gas pipeline operation using genetic algorithms and rule learning*. PhD thesis, University of Michigan, Ann Arbor, MI, 1983.
- D. E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. *Foundations of Genetic Algorithms*, pages 69–93, 2003.
- D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, 1987.
- D. E. Goldberg and M. Rudnick. Genetic algorithms and the variance of fitness. *Complex Systems*, 5(3):265–278, 1991.
- D. E. Goldberg, K. Deb, and B. Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.

- D. E. Goldberg, K. Deb, and J. H. Clark. Genetic algorithms, noise, and the sizing of populations. *Complex Systems*, 6:333–362, 1992.
- D. E. Goldberg, D. Thierens, and K. Deb. Toward a better understanding of mixing in genetic algorithms. *Journal of the Society of Instrument and Control Engineers*, 32(1):10–16, 1993.
- D. E. Goldberg, K. Sastry, and T. Latoza. On the supply of building blocks. In *GECCO'01: Proceedings of the 2001 Genetic and Evolutionary Computation Conference*, pages 336–342. Springer, 2001.
- D. E. Goldberg, K. Sastry, and X. Llorà. Toward routine billion-variable optimization using genetic algorithms: Short communication. *Complexity*, 12(3):27–29, 2007. ISSN 1076-2787. doi: <http://dx.doi.org/10.1002/cplx.v12:3>.
- A. González and R. Pérez. SLAVE: A genetic learning system based on an iterative approach. *IEEE Transactions on Fuzzy Systems*, 7(2):176–191, 1999.
- D. P. Greene. Automated knowledge acquisition: Overcoming the expert systems bottleneck. In *Proceedings of the Seventh International Conference on Information Systems*, pages 107–117, Pittsburgh, PA, 1987. Lawrence Erlbaum Assoc.
- D. P. Greene. *Inductive knowledge acquisition using genetic adaptive search*. PhD thesis, Pittsburgh, PA, 1992.
- D. P. Greene and S. E. Smith. Competition-based induction of decision models from examples. *Machine Learning*, 13:229–257, 1993.
- D. P. Greene and S. F. Smith. A genetic system for learning models of consumer choice. In *Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, pages 217–223, Boston, MA, 1987. Morgan Kaufmann.
- J. J. Grefenstette and J. E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20–27, 1989.
- J. J. Grefenstette and J. Fitzpatrick. Genetic search with approximate function evaluations. In *International Conference on Genetic Algorithms and their Applications*, pages 112–120, 1992.
- J. W. Grzymala-Busse, L. K. Goodwin, and W. J. Grzymala-Busse. An approach to imbalanced data sets based on changing rule strength. In *Learning from Imbalanced Data Sets: Papers from the AAAI Workshop*, pages 69–74, 2000.
- H. Han, W. Y. Wang, and B. H. Mao. Borderline-SMOTE: A new over-sampling method in imbalanced data sets learning. In *ICIC'05: Proceedings of the 2005 International Conference on Intelligent Computing*, pages 878–887. Springer-Verlag, 2005.
- G. Harik. Linkage learning via probabilistic modeling in the ECGA. Technical report, Illinois Genetic Algorithm Laboratory, University of Illinois at Urbana-Champaign (IlliGAL Report No. 99010), 1999.

- G. Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, Ann Arbor, 1997. Also available as IlliGAL Report 97005.
- G. Harik, E. Cantú-Paz, D. E. Goldberg, and B. L. Miller. The gambler's ruin problem, genetic algorithms, and the sizing of populations. *Evolutionary Computation*, 7(3):231–253, 1999. ISSN 1063-6560. doi: <http://dx.doi.org/10.1162/evco.1999.7.3.231>.
- F. Herrera. Genetic fuzzy systems: Taxonomy and current research trends and prospects. *Evolutionary Intelligence*, 1:27–46, 2008. doi: 10.1007/s12065-007-0001-5.
- S. Hettich and S. D. Bay. *The UCI KDD Archive* [<http://kdd.ics.uci.edu>]. Irvine, CA: University of California, Department of Information and Computer Science, 1999.
- T. K. Ho and M. Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):289–300, 2002. ISSN 0162-8828.
- J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the Association for Computing Machinery*, 3:297–314, 1962.
- J. H. Holland. Nonlinear environments permitting efficient adaptation. In *Computer and Information Sciences II*. New York: Academic, 1967.
- J. H. Holland. Processing and processors for schemata. In E. L. Jacks, editor, *Associative information processing*, pages 127–146. New York: American Elsevier, 1971.
- J. H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, 1975.
- J. H. Holland. Adaptation. In R. Rosen and F. Snell, editors, *Progress in Theoretical Biology*, volume 4, pages 263–293. New York: Academic Press, 1976.
- J. H. Holland. *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA., 2nd edition, 1992.
- J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. In D. Waterman and F. Hayes-Roth, editors, *Pattern-directed inference systems*, pages 313–329. Academic Press, San Diego, USA, 1978.
- J. Holmes. Differential negative reinforcement improves classifier system learning rate in two-class problems with unequal base rates. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 635–642. Morgan Kaufmann, 1998.
- R. C. Holte, L. E. Acker, and B. W. Porter. Concept learning and the problem of small disjuncts. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 813–818, 1989.
- A. Homaifar and E. McCormick. Simultaneous design of membership functions and rule sets for fuzzy controllers using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(2): 129–139, 1995. ISSN 1063-6706. doi: 10.1109/91.388168.

- J. Horn. *The nature of niching: Genetic algorithms and the evolution of optimal, cooperative populations*. PhD thesis, Illinois Genetic Algorithms Laboratory (IlligAL), University of Illinois at Urbana Champaign, Urbana Champaign 117, 1997.
- H. Ishibuchi and T. Yamamoto. Rule weight specification in fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, 13(4):428–435, 2005.
- H. Ishibuchi, K. Nozaki, N. Yamamoto, and H. Tanaka. Selection fuzzy if-then rules for classification problems using genetic algorithms. *IEEE Transactions on Fuzzy Systems*, 3(3):260–270, 1995.
- H. Ishibuchi, T. Murata, and I. B. Türkşen. Single-objective and two-objective genetic algorithms for selecting linguistic rules for pattern classification problems. *Fuzzy Sets and Systems*, 89(2):135–150, 1997. ISSN 0165-0114. doi: [http://dx.doi.org/10.1016/S0165-0114\(96\)00098-X](http://dx.doi.org/10.1016/S0165-0114(96)00098-X).
- H. Ishibuchi, T. Nakashima, and T. Morisawa. Voting in fuzzy rule-based systems for pattern classification problems. *Fuzzy Sets and Systems*, 103(2):223–238, 1999a.
- H. Ishibuchi, T. Nakashima, and T. Murata. Performance evaluation of fuzzy classifier systems for multidimensional pattern classification problems. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 29(5):601–618, 1999b.
- H. Ishibuchi, T. Yamamoto, and T. Murata. Hybridization of fuzzy GBML approaches for pattern classification problems. *IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics*, 35(2):359–365, 2005.
- B. Jahne, H. Haussecker, and P. Geissler, editors. *Handbook of Computer Vision and Applications*, volume 1-3. Academic Press, 1999.
- C. Z. Janikow. A knowledge-intensive genetic algorithm for supervised learning. *Machine Learning*, 13(2-3):189–228, 1993. ISSN 0885-6125.
- N. Japkowicz and S. Stephen. The class imbalance problem: Significance and strategies. In *International Conference on Artificial Intelligence (IC-AI'00)*, volume 1, pages 111–117, 2000.
- N. Japkowicz and S. Stephen. The class imbalance problem: A systematic study. *Intelligent Data Analysis*, 6(5):429–450, November 2002.
- N. Japkowicz and J. Taeho. Class imbalances versus small disjuncts. *SIGKDD Explorations Newsletter, special issue on learning from imbalanced datasets*, 6(1):40–49, June 2004.
- J. H. Jo and T. H. Ahn. Reviewing the use of genetic algorithms in a real-time computer game. In *Artificial Intelligence and Applications*, 2002.
- G. H. John and P. Langley. Estimating continuous distributions in bayesian classifiers. In *11th Conference on Uncertainty in Artificial Intelligence*, pages 338–345. Morgan Kaufmann, 1995.
- K. A. D. Jong. *Evolutionary computation: A unified approach*. MIT Press, Cambridge MA, 2006.

- K. A. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, University of Michigan, 1975.
- K. A. D. Jong and W. M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 651–656. Sidney, Australia, 1991.
- K. A. D. Jong, W. M. Spears, and D. Gordon. Using genetic algorithms for concept learning. *Genetic Algorithms for Machine Learning, A Special Issue of Machine Learning*, 13(2–3): 161–188, 1993.
- C. M. Karat, J. Vergo, and D. Nahamoo. Conversational interface technologies. In *The human-computer interaction handbook: fundamentals, evolving technologies and emerging applications*, pages 169–186. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA, 2003. ISBN 0-8058-3838-4.
- C. Karr. Genetic algorithms for fuzzy controllers. *AI Expert*, 6(2):26–33, 1991. ISSN 0888-3785.
- F. Kharbat, L. Bull, and M. Odeh. Revisiting genetic selection in the XCS learning classifier system. In *Congress on Evolutionary Computation*, pages 2061–2068, Edinburgh, UK, 2–5 September 2005. IEEE.
- D. Kim, Y. S. Choi, and S. Y. Lee. An accurate cog defuzzifier design using lamarckian co-adaptation of learning and evolution. *Fuzzy Sets and Systems*, 130:207–225, 2002. doi: doi:10.1016/S0165-0114(01)00167-1.
- H. Kitano. Empirical studies of the speed of convergence of neural networks training using genetic algorithms. In *In Proceedings of the Eight National Conference in Artificial Intelligence*, pages 789–796, 1990.
- G. J. Klir and B. Yuan. *Fuzzy sets and fuzzy logic—Theory and applications*. Prentice-Hall, 1995.
- J. Korst and E. Aarts. *Simulated annealing and boltzmann machines*. Wiley-Interscience, New York, 1997.
- T. Kovacs. Towards a theory of strong overgeneral classifiers. In W. Martin and W. M. Spears, editors, *Foundations of Genetic Algorithms*, volume 6, pages 165–184. Morgan Kaufmann, 2001.
- T. Kovacs. XCS classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In Roy, Chawdhry, and Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, pages 59–68, London, UK, 1997. Springer-Verlag.
- T. Kovacs. Deletion schemes for classifier systems. In *GECCO'99: Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 329–336. Morgan Kaufmann, 1999.
- T. Kovacs and L. Bull. Toward a better understanding of rule initialisation and deletion. In *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2777–2780, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-698-1.

- T. Kovacs and M. Kerber. What makes a problem hard for XCS? In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems: Third International Workshop*, pages 80–99. Springer-Verlag, 2001.
- T. Kovacs and M. Kerber. High classification accuracy does not imply effective genetic search. In *GECCO'04: 2004 Genetic and Evolutionary Computation Conference*, pages 785–796, Seattle, WA, USA. 26-30 June, 2004. Springer, LNCS 3103.
- J. R. Koza. Hierarchical genetic algorithms operation on populations of computer programs. In *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 1, pages 768–774, 1989.
- J. R. Koza. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press, Cambridge, Massachusetts, 1992.
- J. R. Koza, M. A. Keane, M. J. Streeter, W. Myrdlowec, J. Yu, and G. Lanza. *Genetic programming IV: Routine human-competitive machine intelligence*. Kluwer Academic Publishers, 2003.
- M. Kubat, R. Holte, and S. Matwin. Machine learning for the detection of oil spills in satellite radar images. *Machine Learning*, 30(2-3):195–215, 1998.
- P. L. Lanzi. Learning classifier systems from a reinforcement learning perspective. *Soft Computing — A Fusion of Foundations, Methodologies and Applications*, 6(3):162–170, 2002.
- P. L. Lanzi. Extending the representation of classifier conditions part I: From binary to messy coding. In *GECCO'99: Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, pages 337–344, Orlando (FL), 1999a. Morgan Kaufmann.
- P. L. Lanzi. *Reinforcement Learning with Classifier Systems*. PhD thesis, Politecnico di Milano, 1999b.
- P. L. Lanzi and A. Perrucci. Extending the representation of classifier conditions part II: From messy coding to S-expressions. In *GECCO'99: Proceedings of the 1999 Genetic and Evolutionary Computation Conference*, volume 1, pages 345–352. Morgan Kaufmann, 1999.
- P. L. Lanzi and S. W. Wilson. Using convex hulls to represent classifier conditions. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1481–1488, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: <http://doi.acm.org/10.1145/1143997.1144240>.
- P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. XCS with computed prediction in continuous multistep environments. In *The 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2032–2039, 2005. doi: 10.1109/CEC.2005.1554945.
- P. Larrañaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation*. Springer, 2002.
- C. F. Lima, M. Pelikan, K. Sastry, M. V. Butz, D. E. Goldberg, and F. Lobo. Substructural neighborhoods for local search in the bayesian optimization algorithm. In *PPSN IX: Parallel Problem Solving from Nature*, pages 232–241, 2006.

- C. Ling and C. Li. Data mining for direct marketing: Problems and solutions. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 1998.
- Z. Liu, A. Liu, C. Wang, and Z. Niu. Evolving neural network using real coded genetic algorithm (GA) for multispectral image classification. *Future Generation Computer Systems*, 20(7): 1119–1129, 2004. ISSN 0167-739X.
- X. Llorà and J. M. Garrell. Evolution of decision trees. In *CCIA'01: Fourth Catalan Conference on Artificial Intelligence*, pages 115–122. ACIA Press, 2001.
- X. Llorà and S. W. Wilson. Mixed decision trees: Minimizing knowledge representation bias in LCS. In *GECCO'04: Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, pages 797–809. Springer-Verlag, LNCS 3103, 2004.
- X. Llorà, R. Reddy, B. Matesic, and R. Bhargava. Towards better than human capability in diagnosing prostate cancer using infrared spectroscopic imaging. In *GECCO'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 2098–2105, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: <http://doi.acm.org/10.1145/1276958.1277366>.
- N. Macià, E. Bernadó-Mansilla, and A. Orriols-Puig. Preliminary approach on synthetic datasets generation for classification. In *In 2008 International Conference on Pattern Recognition*, 2008a. in press.
- N. Macià, E. Bernadó-Mansilla, and A. Orriols-Puig. On the dimensions of data complexity through synthetic data sets. In *In Recent Advances in Artificial Intelligence Research and Development*. IOS Press, 2008b. in press.
- N. Macià, A. Orriols-Puig, and E. Bernadó-Mansilla. Genetic-based synthetic data sets for the analysis of classifiers' behavior. In *HIS'08: Proceedings of the 2008 Hybrid Intelligent Systems Conference*, 2008c. in press.
- S. M. Mahfoud. *Niching methods for genetic algorithms*. PhD thesis, Illinois Genetic Algorithms Laboratory (IlligAL), University of Illinois at Urbana Champaign, Urbana Champaign 117, 1995.
- E. H. Mamdani. Applications of fuzzy algorithm for control a simple dynamic plant. In *Proceedings IEEE*, pages 1585–1588, 1974.
- F. A. Marquez, A. Peregrin, and F. Herrera. Cooperative evolutionary learning of linguistic fuzzy rules and parametric aggregation connectors for mamdani fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 15(6):1162–1178, 2007. ISSN 1063-6706. doi: 10.1109/TFUZZ.2007.904121.
- J. McCharthy. What is artificial intelligence. Technical report, Stanford University Report, Stanford, CA, USA, 2007.
- M. McInerney and A. P. Dhawan. Use of genetic algorithms with backpropagation in training of feedforward neural networks. In *In Proceedings of the IEEE International Conference on Neural Networks (ICNN'93)*, pages 203–208, San Francisco, USA, 1993.

- D. Mellor. A first order logic classifier system. In *GECCO'05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 1819–1826, New York, NY, USA, 2005. ACM. ISBN 1-59593-010-8. doi: <http://doi.acm.org/10.1145/1068009.1068318>.
- R. S. Michalski. On the quasi-minimal solution of the covering problem. In *FCIP'69: Proceedings of the 5th International Symposium on Information Processing*, volume A3, pages 125–128, Bled, Yugoslavia, 1969.
- R. S. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The AQ15 inductive learning system: An overview and experiments. Technical report, Intelligent Systems Group, ISG 86-20, UIUCDCS-R-86-1260, Department of Computer Science, University of Illinois, Urbana, 1986.
- D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, 1994.
- I. Mierswa. Controlling overfitting with multi-objective support vector machines. In *GECCO'07: Proceedings of the 2007 Conference on Genetic and Evolutionary Computation*, pages 1830–1837, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: <http://doi.acm.org/10.1145/1276958.1277323>.
- B. L. Miller and D. E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995.
- B. L. Miller and D. E. Goldberg. Optimal sampling for genetic algorithms. *Intelligent Engineering Systems through Artificial Neural Networks*, 6:291–297, 1996.
- T. M. Mitchell. The discipline of machine learning. Technical report, Machine Learning Department, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 2006.
- T. M. Mitchell. *Machine learning*. McGraw Hill, 1997.
- S. Morales-Ortigosa, A. Orriols-Puig, and E. Bernadó-Mansilla. Can evolution strategies improve learning guidance in XCS? Design and comparison with genetic algorithms based XCS. In *Advances in Artificial Intelligence*, volume in press. IOS press, 2008a.
- S. Morales-Ortigosa, A. Orriols-Puig, and E. Bernadó-Mansilla. New crossover operator for evolutionary rule discovery in XCS. In *Proceedings of the 2008 Hybrid and Intelligent Systems Conference*, volume in press. IEEE, 2008b.
- H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm I. Continuous parameter optimization. *Evolutionary Computation*, 1(1):25–49, 1993.
- K. Nakaoka, T. Furuhashi, and Y. Uchikawa. A study on apportionment of credits of fuzzy classifier system for knowledge acquisition in large scale systems. In *Proceedings of the 3th IEEE International Conference on Fuzzy Systems*, pages 1797–1800. Morgan Kaufmann, 1994.
- P. B. Nemenyi. *Distribution-free multiple comparisons*. PhD thesis, Princeton University, New Jersey, USA, 1963.

- N. J. Nilson. *Introduction to Machine Learning. Draft of Incomplete Notes*. Electronic version, draft edition, 2005.
- Y. Nomura, K. Ikebukuro, K. Yokoyama, T. Takeuchi, Y. Arikawa, S. Ohno, I. Karube, and M. Valenzuela-Rendón. Reinforcement learning in the fuzzy classifier system. *Expert Systems with Applications*, 14:237–247, 1998.
- A. Nurnberger, C. Borgelt, and A. Klose. Improving naive bayes classifiers using neuro-fuzzy learning. In *Proceedings of the 1999 Conference on Neural Information Processing*, volume 1, pages 154–159, Perth, WA, Australia, 1999.
- A. Orriols-Puig and E. Bernadó-Mansilla. Analysis of reduction algorithms for XCS classifier system. In *Recent Advances in Artificial Intelligence Research and Development*, number 113 in 1, pages 383–390. IOS Press, October 2004.
- A. Orriols-Puig and E. Bernadó-Mansilla. The class imbalance problem in learning classifier systems: A preliminary study. In *GECCO'05: Proceedings of the 2005 Genetic and Evolutionary Computation Conference Workshop Program*, pages 74–78, Washington, D.C., USA, 25-29 June 2005a. ACM Press.
- A. Orriols-Puig and E. Bernadó-Mansilla. The class imbalance problem in UCS classifier system: Fitness adaptation. In *Congress on Evolutionary Computation*, volume 1, pages 604–611, Edinburgh, UK, 2-5 September 2005b. IEEE.
- A. Orriols-Puig and E. Bernadó-Mansilla. Bounding XCS parameters for unbalanced datasets. In *GECCO'06: Proceedings of the 2006 Genetic and Evolutionary Computation Conference*, pages 1561–1568. ACM Press, 2006a.
- A. Orriols-Puig and E. Bernadó-Mansilla. The class imbalance problem in UCS classifier system: A preliminary study. In *Advances at the frontier of LCS*, pages 164–183. Springer, 2007.
- A. Orriols-Puig and E. Bernadó-Mansilla. A further look at UCS classifier system. In *GECCO'06: Proceedings of the 2006 Genetic and Evolutionary Computation Conference Workshop Program*, page to appear, Seattle, W.A., USA, 08–12 July 2006b. ACM Press.
- A. Orriols-Puig and E. Bernadó-Mansilla. Revisiting UCS: Description, fitness sharing and comparison with XCS. In *Advances at the Frontier of LCSs*. Springer, 2008.
- A. Orriols-Puig and E. Bernadó-Mansilla. Mining imbalanced data with learning classifier systems. In *Learning Classifier Systems in Data Mining*, pages 123–145. Springer, 2008a.
- A. Orriols-Puig and E. Bernadó-Mansilla. Evolutionary rule-based systems for imbalanced datasets. *Soft Computing Journal*, doi=10.1007/s00500-008-0319-7, 2008b.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. Fuzzy-UCS: Preliminary results. In *GECCO'07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference Workshop Program*, volume 3, pages 2871–2874. ACM Press, 2007a.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. Aprendizaje supervisado de reglas difusas mediante un sistema clasificador evolutivo estilo Michigan. In *Proceedings of the II Congreso Espa nol de Informática (CEDI 2007). I Jornadas sobre Algoritmos Evolutivos y Metaheurísticas(JAEM07)*, pages 171–178, 2007b.

- A. Orriols-Puig, D. E. Goldberg, K. Sastry, and E. Bernadó-Mansilla. Modeling XCS in class imbalances: Population size and parameters' settings. In *GECCO'07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, volume 2, pages 1838–1845. ACM Press, 2007c.
- A. Orriols-Puig, K. Sastry, P. Lanzi, D. Goldberg, and E. Bernadó-Mansilla. Modeling selection pressure in XCS for proportionate and tournament selection. In *GECCO'07: Proceedings of the 2007 Genetic and Evolutionary Computation Conference*, volume 2, pages 1846–1853. ACM Press, 2007d.
- A. Orriols-Puig, E. Bernadó-Mansilla, D. E. Goldberg, K. Sastry, and P. L. Lanzi. Facetwise analysis of XCS for problems with class imbalances. *IEEE Transactions on Evolutionary Computation*, submitted, 2008a.
- A. Orriols-Puig, E. Bernadó-Mansilla, N. Macià, and T. K. Ho. CMAPI: A complexity metrics API. *Journal of Machine Learning Research (submitted)*, 2008b.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. Evolving fuzzy rules with UCS. In *Advances at the Frontier of LCSs*. Springer, 2008c.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. Genetic-based machine learning systems are competitive for pattern recognition. *Evolutionary Intelligence*, 2008d. doi: 10.1007/s12065-008-0013-9.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. A comparative study of several classifiers in supervised learning. In *Learning Classifier Systems in Data Mining*, pages 205–230. Springer, 2008e.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. Fuzzy-UCS: a Michigan-style learning fuzzy-classifier system for supervised learning. *IEEE Transactions on Evolutionary Computation*, 2008f. doi: 10.1109/TEVC.2008.925144.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. First approach toward on-line evolution of association rules with learning classifier systems. In *GECCO'08: Proceedings of the 2008 Genetic and Evolutionary Computation Conference Workshop Program*, pages 2031–2038, New York, NY, USA, 2008g. ACM. ISBN 978-1-60558-131-6. doi: <http://doi.acm.org/10.1145/1388969.1389017>.
- A. Orriols-Puig, J. Casillas, and E. Bernadó-Mansilla. Approximate versus linguistic representation in fuzzy-ucs. In *H AIS'08: Proceedings of the 2008 International Workshop on Hybrid Artificial Intelligence Systems*, LNAI, (in press), 2008h.
- A. Orriols-Puig, J. Casillas, and F. Martínez-López. Modelado causal en marketing mediante aprendizaje no supervisado de reglas de asociación difusas. In *ESTYLF'08: Proceedings of the XIV Congreso Español sobre Tecnologías y Lógica Fuzzy*, 2008i. Spanish version only.
- J. Otero and L. Sánchez. Induction of descriptive fuzzy classifiers with the logitboost algorithm. *Soft Computing*, 10(9):825–835, 2006. ISSN 1432-7643.
- G. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99, 1990.

- A. Parodi and P. Bonelli. A new approach to fuzzy classifier systems. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 223–230. Morgan Kaufmann, 1993.
- M. Pazzani, C. Merz, P. Murphy, K. Ali, T. Hume, and C. Brunk. Reducing the misclassification costs. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 217–225, 1994.
- W. Pedrycz. Fuzzy sets technology in knowledge discovery. *Fuzzy Sets and Systems*, 98(3): 279290, 1998.
- M. Pelikan. *Hierarchical bayesian optimization algorithm: Toward a new generation of evolutionary algorithms*, volume 170 of *Studies in Computational Intelligence*. Springer, 2005.
- M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. BOA: The bayesian optimization algorithm. In *GECCO'09: Genetic and Evolutionary Computation Conference*, pages 525–532, 1999.
- M. Pelikan, D. E. Goldberg, and E. Cantú-Paz. Linkage learning, estimation distribution, and bayesian networks. *Evolutionary Computation*, 8(3):314–341, 2000a.
- M. Pelikan, D. E. Goldberg, and F. Lobo. A survey of optimization by building and using probabilistic models. In *Proceedings of the American Control Conference*, pages 3289–3293, 2000b.
- M. Pelikan, K. Sastry, and E. Cantú-Paz. *Scalable optimization via probabilistic modeling*, volume 33 of *Studies in Computational Intelligence*. Springer, 2006. ISBN 978-3-540-34953-2.
- J. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods - Support Vector Learning*. MIT Press, 1998.
- J. R. Quinlan. Discovering rules by induction from large collections of examples. In D. Michie, editor, *Expert Systems in the Micro Electronic Age*, Edinburgh, UK, 1979. Edinburgh University Press.
- J. R. Quinlan. *C4.5: Programs for machine learning*. Morgan Kaufmann Publishers, San Mateo, California, 1995.
- I. Rechenberg. *Cybernetic solution path of an experimental problem*, volume 1122. 1965.
- I. Rechenberg. *Evolution strategie: Optimierung technischer systeme nach prinzipien der biologischen evolution*. Frommann-Holzboog, 1973.
- R. L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.
- M. Robnik-Sikonja, D. Cukjati, and I. Kononenko. Comprehensible evaluation of prognostic factors and prediction of wound healing. *Artificial Intelligence in Medicine*, 29(1-2):25–38, 2003.
- D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. In D. E. Rumelhart and J. L. McClelland, editors, *Parallel Distributed Processing*, pages 318–364, Cambridge, MA, 1986. MIT Press.

- S. Russell and P. Norvig. *Artificial intelligence: A modern approach*. Series on Artificial Intelligence. Prentice Hall, 2nd edition, 2002.
- L. Sánchez and I. Couso. Fuzzy random variables-based modeling with GA-P algorithms. In R. Yager and B. Bouchon-Menier, editors, *Information, Uncertainty and Fusion*, pages 245–256. Kluwer, 2000.
- L. Sánchez and I. Couso. Advocating the use of imprecisely observed data in genetic fuzzy systems. *IEEE Transactions on Fuzzy Systems*, 15(4):551–562, 2007.
- L. Sánchez and I. Couso. Learning with imprecise examples with GA-P algorithms. *Soft Computing*, 5(1–4):305–319, 1998.
- L. Sánchez and J. Otero. Boosting fuzzy rules in classification problems under single-winner inference. *International Journal of Intelligent Systems*, 22(9):1021–1034, 2007. ISSN 0884-8173. doi: <http://dx.doi.org/10.1002/int.v22:9>.
- L. Sánchez, I. Couso, and J. A. Corrales. Combining GP operators with SA search to evolve fuzzy rule based classifiers. *Information Sciences*, 136(1–4):175–191, 2001. ISSN 0020-0255.
- L. Sánchez, I. Couso, and J. Casillas. Modeling vague data with genetic fuzzy systems under a combination of crisp and imprecise criteria. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Multicriteria Decision Making*, pages 346–353, 2007.
- K. Sastry and D. E. Goldberg. Modeling tournament selection with replacement using apparent added noise. *Intelligent Engineering Systems Through Artificial Neural Networks*, 11:129–134, 2001.
- K. Sastry and D. E. Goldberg. Analysis of mixing in genetic algorithms: A survey. Technical report, IlliGAL Report No. 2002012, Urbana, IL: University of Illinois at Urbana-Champaign, 2002.
- K. Sastry and D. E. Goldberg. Probabilistic model building and competent genetic programming. In R. L. Riolo and B. Worzel, editors, *Genetic Programming Theory and Practice*, pages 205–220. Kluwer, 2003a.
- K. Sastry and D. E. Goldberg. Scalability of selectorecombinative genetic algorithms for problems with tight linkage. In *GECCO'03: Proceedings of the 2003 Genetic and Evolutionary Computation Conference*, pages 1332–1344, 2003b.
- K. Sastry and D. E. Goldberg. Designing competent mutation operators via probabilistic model building of neighborhoods. In *GECCO'04: Proceedings of the 2004 Genetic and Evolutionary Computation Conference*, volume 2, pages 114–125, 2004.
- K. Sastry and A. Orriols-Puig. Extended compact genetic algorithm in matlab. Technical report, IlliGAL Report No. 2007009, Urbana-Champaign IL 61801, USA, 2007.
- H. P. Schwefel. *Numerical optimization of computer models*. John Wiley & Sons, 1981.
- D. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman & Hall, 2000.

- R. E. Smith and M. Valenzuela-Rendón. A study of rule set development in learning classifier system. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 340–346, San Mateo, California, 1989. Morgan Kaufmann. ISBN 1-55860-006-3.
- S. F. Smith. Flexible learning of problem solving heuristics through adaptive search. In *In Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, pages 421–425, Los Altos, CA, 1983. Morgan Kaufmann.
- S. F. Smith. Adaptive learning systems. In R. Forsyth, editor, *Expert Systems: Principles and Case Studies*, pages 168–189. Chapman and Hall, London, U.K., 1984.
- S. F. Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, University of Pittsburgh, USA, 1980.
- S. Y. Sohn. Meta analysis of classification algorithms for pattern recognition. *IEEE Transactions of Pattern Analysis and Machine Learning*, 21(11):1137–1144, 1999.
- C. Stone and L. Bull. For real! XCS with continuous-valued inputs. *Evolutionary Computation*, 11(3):299–336, 2003.
- R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. Cambridge, MA: MIT Press, 1998.
- K. Tamee, L. Bull, and O. Pinngern. A learning classifier system approach to clustering. In *ISDA'06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications*, pages 621–626, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2528-8. doi: <http://dx.doi.org/10.1109/ISDA.2006.62>.
- K. Tamee, L. Bull, and O. Pinngern. Towards clustering with XCS. In *GECCO'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1854–1860, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: <http://doi.acm.org/10.1145/1276958.1277326>.
- F. Teixidó-Navarro, A. Orriols-Puig, and E. Bernadó-Mansilla. Hierarchical evolution of linear regressors. In *GECCO'08: Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1413–1420, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-130-9. doi: <http://doi.acm.org/10.1145/1389095.1389367>.
- S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. Elsevier, 3rd edition, 2006.
- D. Thierens and D. E. Goldberg. Convergence models of genetic algorithm selection schemes. In *Parallel Problem Solving from Nature*, volume 3, pages 116–121, 1994a.
- D. Thierens and D. E. Goldberg. Elitist recombination: An integrated selection recombination ga. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 508–512, 1994b.
- D. Thierens, D. E. Goldberg, and A. G. Pereira. Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, page 535540, 1998.

- P. Thrift. Fuzzy logic synthesis with genetic algorithms. In R. K. Belew and L. B. Booker, editors, *Proceedings of the fourth International Conference on Genetic Algorithms*, pages 509–513. Morgan Kaufmann, 1991.
- I. Tomek. Two modifications of CNN. *IEEE Transactions on Systems, Man and Cybernetics*, 6:769–772, 1976.
- M. Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In *4th ICGA*, pages 346–353. Morgan Kaufmann, 1991.
- R. M. M. Vallim, D. E. Goldberg, X. Llorà, T. S. P. C. Duque, and A. C. P. L. F. Carvalho. A new approach for multi-label classification based on default hierarchies and organizational learning. In *GECCO'08: Proceedings of the 2008 GECCO conference companion on Genetic and evolutionary computation*, pages 2017–2022, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-131-6. doi: <http://doi.acm.org/10.1145/1388969.1389015>.
- V. Vapnik. *The nature of statistical learning theory*. Springer Verlag, New York, 1995.
- J. Velasco. Genetic-based on-line learning for fuzzy process control. *International Journal of Intelligent Systems*, 13:891–903, 1998.
- G. Venturini. SIA: A supervised inductive algorithm with genetic search for learning attributes based concepts. In P. B. Brazdil, editor, *Machine Learning: ECML-93 - Proc. of the European Conference on Machine Learning*, pages 280–296. Springer-Verlag, Berlin, Heidelberg, 1993.
- G. Venturini. *Apprentissage adaptatif et apprentissage supervisé par algorithme génétique*. PhD thesis, Université de Paris-Sud, 1994.
- C. Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, 1989.
- G. M. Weiss. Mining with rarity: a unifying framework. *SIGKDD Explorations Newsletter, special issue on learning from imbalanced datasets*, 6(1):7–19, 2004.
- G. M. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 359–363, New York, NY, 1998. AAAI Press, Menlo Park, CA.
- G. M. Weiss and F. Provost. Learning when training data are costly: the effect of class distribution on tree induction. *Journal of Artificial Intelligence Research*, 19:315–354, 2003.
- B. Widrow and M. E. Hoff. Adaptive switching circuits. *Neurocomputing: foundations of research*, pages 123–134, 1988.
- B. Widrow and M. A. Lehr. 30 years of adaptive neural networks: perceptron, Madaline, and backpropagation. *Proceedings of the IEEE*, 78(9):1415–1442, 1990. doi: 10.1109/5.58323.
- D. Wierstra, F. Gómez, and J. Schmidhuber. Modeling systems with internal state using evolino. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1795–1802. ACM Press, 2005.

- J. R. Wilcox. Organizational learning within a learning classifier system. Master's thesis, University of Illinois at Urbana Champaign, 1995.
- F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics*, 1:80–83, 1945.
- S. W. Wilson. Get real! XCS with continuous-valued inputs. In *Learning Classifier Systems. From Foundations to Applications*, LNAI, pages 209–219, Berlin, 2000. Springer-Verlag.
- S. W. Wilson. Mining oblique data with XCS. In *IWLCS'00: Revised Papers from the Third International Workshop on Advances in Learning Classifier Systems*, pages 158–176, London, UK, 2001. Springer-Verlag. ISBN 3-540-42437-7.
- S. W. Wilson. Compact rulesets from XCSI. In P. L. Lanzi, W. Stolzmann, and S. W. Wilson, editors, *Advances in Learning Classifier Systems, 4th International Workshop*, volume 2321 of *Lecture Notes in Artificial Intelligence*, pages 197–210. Springer, 2002a. ISBN 3-540-43793-2.
- S. W. Wilson. Classifiers that approximate functions. *Journal of Natural Computing*, 1(2): 211–234, 2002b.
- S. W. Wilson. Classifier conditions using gene expression programming. Technical report, IlliGAL Report No. 2008001, Urbana-Champaign IL 61801, USA, 2008.
- S. W. Wilson. Aubert processing and intelligent vision. Technical report, Polaroid Corporation, Cambridge, MA, 1981.
- S. W. Wilson. Adaptive “cortical” pattern recognition. In *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, pages 188–196, 1985a.
- S. W. Wilson. Knowledge growth in an artificial animal. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 16–23, Mahwah, NJ, USA, 1985b. Lawrence Erlbaum Associates, Inc. ISBN 0-8058-0426-9.
- S. W. Wilson. Classifier systems and the animat problem. *Maching Learning*, 2(3):199–228, 1987. ISSN 0885-6125. doi: <http://dx.doi.org/10.1023/A:1022655214215>.
- S. W. Wilson. ZCS: a zeroth level classifier system. *Evolutionary Computation*, pages 1–18, 1994.
- S. W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995.
- S. W. Wilson. Generalization in the XCS classifier system. In *3rd Annual Conf. on Genetic Programming*, pages 665–674. Morgan Kaufmann, 1998.
- S. W. Wilson and D. E. Goldberg. A critical review of classifier systems. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 244–255, 1989.
- I. Witten and E. Frank. *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition, 2005. ISBN 0-12-088407-0.
- D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

- D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.
- X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, P. S. Yu, Z. H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg. Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37, 2007. doi: 10.1007/s10115-007-0114-2.
- I. Yalabik and T. Y.-V. Fatos. A pattern classification approach for boosting with genetic algorithms. *22nd International International Symposium on Computer and Information Sciences, 2007. ISCIS 2007*, pages 1–6, 2007. doi: 10.1109/ISCIS.2007.4456870.
- L. A. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.
- L. A. Zadeh. Outline of a new approach to the analysis of complex systems and decision processes. *IEEE Transactions on Systems, Man, and Cybernetics*, 3:28–44, 1973.

Index

- animat, 3, 27, 33
- association rules, 223

- best action map, 41, 44, 45, 50, 60
- Bonferroni-Dunn test, 176, 194, 235
- boundedly difficult problems, 4, 24, 50, 51, 67

- C4.5, 138, 194
- COGIN, 30
- comparison UCSs with UCSns, 55
- complete action map, 34, 39, 60, 72
- CS-1, 2, 27

- data complexity, 131, 156, 222
 - complexity metrics, 223
 - geometrical complexity, 131, 132
- data mining, 13
- data streaming, 209
- decision list, 28

- evolutionary computation, 15
 - biological principles of, 16
 - estimation of distribution algorithms, 18, 24
 - evolution strategies, 17
 - evolutionary programming, 18
 - genetic algorithms, 17
 - genetic programming, 18
 - taxonomy, 17
- EYE-EYE, 3, 27

- fitness dilemma, 64
- fitness sharing, 52
- Friedman's test, 137, 186, 233
- Fuzzy AdaBoost, 193
- Fuzzy GAP, 193
- Fuzzy GP, 193
- Fuzzy LogitBoost, 193

- Fuzzy MaxLogitBoost, 193
- fuzzy partition, 168
- fuzzy rule-based systems, 162
- Fuzzy SAP, 193
- Fuzzy-UCS, 166
 - action winner inference, 173
 - approximate representation, 179
 - classifier discovery, 171
 - classifier parameter update, 170
 - configuration sensitivity, 174
 - crossover, 171
 - data streaming, 209
 - fittest rules inference, 173
 - inference, 172
 - knowledge representation, 167
 - learning interaction, 169
 - linguistic fuzzy rule, 167
 - linguistic vs. approximate representation, 185
 - mutation, 171
 - rule granularity, 181
 - rule set reduction, 173
 - schematic, 167
 - selection, 171
 - study knowledge representation, 177
 - SWOT, 212
 - weighted average inference, 173

- GABL, 29
- GAssist, 29, 194
- genetic algorithms, 17, 18
 - crossover, 20
 - cycle, 19
 - description, 19
 - design decomposition, 22, 73
 - in engineering, 24
 - intuition, 21
 - mutation, 20

- natural selection, 16
- pseudo code, 21
- replacement, 20
- schema theorem, 21
- selection, 20
- steady-state niche-based, 38, 43
- theory, 21
- why use, 18
- genetic fuzzy systems, 163
 - fuzzy logic, 162
 - fuzzy systems, 162
- GIL, 29
- HIDER, 30
- HIRELin, 30
- IBk, 138, 194
- imbalance ratio, 71, 110, 130
- influential learners, 127, 159
- interval-based rule representation, 45, 128
- LCS in imbalanced domains, 74
 - design decomposition, 75
- learning classifier systems and genetic based-machine learning
 - historical remarks, 2
 - Michigan-style, 2, 26
 - Pittsburgh-style, 3
- learning classifier systems and genetic-based machine learning, 25
 - genetic cooperative-competitive learning, 30
 - iterative rule learning, 29
 - organizational classifier system, 30
 - Pittsburgh-style, 28
- learning fuzzy-classifier systems, 165
- legible models, 6
- LS-1, 3, 29
- machine learning, 1, 11
 - clustering, 14
 - dimensionality reduction, 14
 - reinforcement learning, 13, 14
 - supervised learning, 13, 14
 - unsupervised learning, 13, 14
 - why apply, 12
- Mamdani FRBSs, 162
- membership degree, 162
- Naïve Bayes, 194
- NAX, 30
- Nemenyi test, 137, 186, 233
- newboole, 27
- niche, 36, 37, 43, 70, 130, 171
 - in continuous attributes, 130
 - nourished niche, 71, 111
 - starved niche, 71, 109
- Niche extinction
 - UCS
 - tournament selection, 122
 - XCS
 - proportionate selection, 95, 121
 - tournament selection, 100
- niche imbalance ratio, 130
- over-general classifier, 70
- Part, 194
- patchquilt integration, 104, 122
- rare classes, 5
 - challenges in LCSs, 68
 - examples, 68
 - offline learners, 69
 - online learners, 69
- re-sampling techniques, 141
 - case study, 146
 - comparison, 150
 - cSMOTE, 144
 - random over-sampling, 142
 - SMOTE, 143
 - under-sampling based on Tomek links, 143
- real-world problems, 185
 - bridge the gap with the theory, 130
 - class-imbalance problem, 130
- REGAL, 30
- representative, 70, 130
 - in continuous attributes, 130
- schema, 70
- self-adaptation, 132
 - UCS, 134
 - XCS, 133

- separate-and-conquer, 29
- SIA, 29
- small disjunct, 131
- SMO, 138, 194
- statistical analysis, 229

- T-conorm, 168
- T-norm, 168
- takeover time, 92, 120
 - proportionate selection, 93, 95
 - tournament selection, 97
- the decoder problem, 54, 56, 61, 226
- the multiplexer problem, 54, 57, 59, 62, 68, 71, 123, 135, 227
 - alternating noise, 228
 - imbalanced, 228
- the parity problem, 54, 56, 61, 78, 87, 89, 91, 117–119, 225
 - lack of fitness guidance, 58
- the position problem, 54, 56, 61, 226
- thesis objectives, 7

- UCS, 27, 41, 109, 127, 169, 194
 - classifier discovery, 43
 - classifier’s parameter update, 43
 - evolutionary pressures, 45
 - deletion, 45
 - fitness, 45
 - mutation, 45
 - set, 45
 - subsumption, 45
 - exploration regime, 60
 - facetwise analysis, 110
 - fitness sharing, 53, 55
 - inference, 44
 - learning interaction, 42
 - ternary rule representation, 41
- UCS in imbalanced domains, 110
 - covering, 112
 - covering probability, 113
 - discovery minority class, 113
 - estimation of classifier parameters, 111
 - in real-world problems, 138
 - occurrence-based reproduction, 119
 - patchquilt integration, 122
 - population size bound, 115, 116
- starved niches
 - time to creation, 115
 - time to deletion, 115
- unordered bound rule representation, 47
 - covering, 47
 - crossover, 48
 - mutation, 48
 - subsumption, 48

- Wilcoxon signed-ranks test, 55, 137, 186, 230

- XCS, 3, 27, 33, 34, 67, 127
 - classifier discovery, 38
 - classifier’s parameter update, 36
 - evolutionary pressures, 39
 - deletion, 40
 - fitness, 40
 - mutation, 40
 - set, 40
 - subsumption, 40
 - facetwise analysis, 73
 - inference, 39
 - learning interaction, 35
 - ternary rule representation, 34
 - Widrow-hoff rule, 133
- XCS in imbalanced domains, 70
 - covering, 81
 - covering probability, 82
 - discovery minority class, 82
 - estimation of classifier parameters, 77
 - gradient descent, 80
 - imbalance bound, 78
 - in real-world problems, 138
 - occurrence-based reproduction, 89
 - patchquilt integration, 104
 - population size bound, 85, 86
 - starved niches
 - time to creation, 84
 - time to deletion, 85
 - Widrow-hoff rule, 78, 79

- ZCS, 3, 33
- ZeroR, 194

