KEEL: UNA HERRAMIENTA SOFTWARE PARA EL ANÁLISIS DE SISTEMAS DIFUSOS EVOLUTIVOS

Joaquín Derrac[1] Alberto Fernández[1] Julián Luengo[1] Salvador García[2]
Luciano Sánchez[3] Jesús Alcalá[1] Francisco Herrera[1]

[1] Dept. de Ciencias de la Computación e Inteligencia Artificial, CITIC-UGR, Universidad de Granada, {alberto,julianlm,jderrac,jalcala,herrera}@decsai.ugr.es
[2] Dept. de Informática, Universidad de Jaén, sglopez@ujaen.es
[3] Dept. de Informática, Universidad de Oviedo, luciano@uniovi.es

Resumen

En este trabajo se presenta *KEEL*, una herramienta software no comercial, capaz de abordar una amplia gama de problemas de minería de datos (regresión, clasificación, asociación, agrupamiento ...) mediante la generación de experimentos estándar y educativos. Dispone de una amplia librería de algoritmos de Sistemas Difusos Evolutivos basados en diferentes esquemas: Michigan, Pittsburgh, IRL y GCCL. También dispone de módulos de tratamiento de datos y análisis estadísticos. A modo de ejemplo, mostramos un caso de estudio sobre el análisis de dos propuestas diferentes de aprendizaje de Sistemas Difusos.

Palabras Clave: Herramienta software, Minería de Datos, Java, Sistemas Difusos Evolutivos.

1 INTRODUCCIÓN

La definición automática de los Sistemas Difusos (SD) puede ser considerada como un proceso de optimización o de búsqueda donde, actualmente, los Algoritmos Evolutivos (AE)[10] (en particular los Algoritmos Genéticos (AG)) son considerados la técnica de búsqueda global más conocida y utilizada. Además, la codificación genética que utilizan les permite introducir conocimiento *a priori* a partir del cual iniciar la búsqueda. Por esta razón, los Algoritmos Evolutivos han sido aplicados satisfactoriamente en los últimos años, dando lugar a la aparición de los llamados Sistemas Difusos Evolutivos (SDEs) [7].

Los SDEs han sido ampliamente utilizados en numerosos campos [3, 18]. Sin embargo, su empleo requiere cierta experiencia en programación, y una gran cantidad de esfuerzo y tiempo para escribir un programa que los implemente (suelen ser algoritmos sofisticados). Este tedioso trabajo debe ser realizado antes de poder centrar la atención en los aspectos importantes del problema. En los últimos años se han desarrollado muchas herramientas software para aliviar esta tarea. Aunque muchas se distribuyen comercialmente (p.ej. Clementine [1], SPSS, varias *toolbox* de MATLAB [2]), algunas, como Weka [26] o Java-ML [1] están disponibles como software libre (recomendamos visitar el Directorio de Software KDnuggets [3] como referencia). Estas últimas han adquirido una especial relevancia recientemente [20].

En esta contribución presentamos *KEEL* (*Knowledge Extraction based on Evolutionary Learning*)[4], una herramienta software no comercial escrita en Java. *KEEL* permite al usuario emplear AEs en diferentes tipos de problemas de minería de datos: Regresión, clasificación, agrupamiento, asociación, etc., incluyendo una gran recopilación de los SDEs existentes. Además, ofrece varias ventajas:

- Primero, reduce el trabajo de programación. Incluye una gran librería con algoritmos de SDEs basados en diferentes paradigmas (Pittsburgh, Michigan, IRL y GCCL) y simplifica su integración con diferentes técnicas de preprocesamiento. Libera a los investigadores del esfuerzo de programación, permitiéndoles centrarse en el análisis de sus nuevos modelos de aprendizaje, en comparación con los ya existentes.
- Segundo, amplía el rango de posibles usuarios de los SDEs. Es un software fácil de usar, con una gran cantidad de propuestas ya implementadas, por lo que reduce considerablemente el nivel de conocimientos y experiencia requeridos a la hora de realizar un estudio que incluya SDEs, incluso para investigadores con menor experiencia.

[1] http://www.spss.com/clementine
[2] http://www.mathworks.com
[3] http://www.kdnuggets.com/software
[4] http://www.keel.es

- Tercero, gracias al empleo de un paradigma estricto de orientación a objetos tanto en la librería como en la herramienta software, ambos pueden ser utilizados en cualquier máquina Java, con independencia del sistema operativo existente. Esto simplifica considerablemente el empleo de la herramienta por parte de cualquier investigador.

El presente trabajo se organiza de la siguiente forma: La Sección 2 describe las principales propuestas de aprendizaje con respecto a la forma de codificar la base de reglas. La Sección 3 presenta los principales SDEs existentes en KEEL. La Sección 4 describe las principales características y módulos de KEEL. En la Sección 5, se muestra un caso de estudio para ilustrar el empleo de KEEL. Finalmente, la Sección 6 muestra algunas conclusiones y trabajo futuro.

2 APRENDIZAJE EVOLUTIVO

Aunque los AEs no fueron diseñados específicamente para aprendizaje, sino como algoritmos de búsqueda global, su empleo ofrece varias ventajas. Muchas metodologías están basadas en la búsqueda de un buen modelo dentro del espacio de modelos posibles. En este sentido, los AEs son muy flexibles, ya que pueden manejar diferentes representaciones. Los procesos de aprendizaje evolutivo cubren diferentes niveles de complejidad con respecto a los cambios estructurales del SDE, desde el caso más simple de optimización de parámetros hasta el más alto nivel de complejidad de aprender una base de reglas completa, mediante la cooperación o competición entre cromosomas y la codificación empleada.

Existe una amplia gama de posibilidades para la tarea de aprender una base de reglas. Los métodos de aprendizaje evolutivo siguen dos enfoques distintos para codificar reglas en una población de individuos:

- "Cromosoma = Conjunto de reglas", o enfoque Pittsbourgh, en el que cada individuo representa un conjunto de reglas [19]. En este caso, los cromosomas evolucionan bases de reglas completas, y compiten entre sí durante la búsqueda. GABIL es una propuesta que sigue este enfoque [17].
- "Cromosoma = Regla", en el que cada individuo representa una única regla, y el conjunto de reglas completo se obtiene combinando varios individuos de una población (cooperación) o a partir de varias ejecuciones (competición).

  Dentro el enfoque "Cromosoma = Regla", existen:

  – El enfoque Michigan, en el que cada individuo codifica una única regla. Este tipo de sistemas suelen denominarse sistemas de aprendizaje de clasificadores [14]. Son sistemas de paso de mensajes, basados en reglas que emplean aprendizaje por refuerzo y un AG para aprender reglas que guíen el aprendizaje en un problema dado. El AG detecta nuevas reglas que reemplazan a las peores mediante competición entre cromosomas en el proceso evolutivo. XCS [25] es un ejemplo de este enfoque.

  – El enfoque IRL (*Iterative Rule Learning*, aprendizaje iterativo de reglas), en el que cada cromosoma representa una regla. Los cromosomas compiten entre sí, tomándose el mejor en cada ejecución del AG. La solución global se forma a partir de ellos, ejecutando múltiples veces el algoritmo. SIA [23] es una propuesta que sigue este enfoque.

  – El enfoque GCCL (*Genetic Cooperative-Competitive Learning*, aprendizaje genético cooperativo-competitivo), en el que la base de reglas se codifica a partir de la población completa o de un subconjunto de ella. En este modelo, los cromosomas compiten y cooperan simultáneamente. LOGENPRO [27] es un ejemplo de este enfoque.

Todos ellos han sido empleados para el aprendizaje de base de reglas. Es posible encontrar varios ejemplos empleando reglas difusas o intervalares en KEEL[5].

3 ALGORITMOS EVOLUTIVOS PARA SISTEMAS DIFUSOS EN KEEL

En los últimos años, el número de artículos publicados en el área los SDEs se ha incrementado, debido a su potencial. Al contrario que las redes neuronales, el agrupamiento, la inducción de reglas y otros muchos enfoques, los AGs ofrecen un mecanismo para codificar y evolucionar operadores de agregación para los antecedentes de las reglas, diferentes semánticas de reglas, operadores de agregación de base de reglas y métodos de *defuzzificación*. Por ello, los AGs siguen siendo hoy una de las pocas técnicas disponibles para diseñar y optimizar por completo los SDs basados en reglas, permitiendo a los investigadores decidir qué componentes son fijos y cuáles deben evolucionar durante el proceso de búsqueda.

KEEL divide las propuestas de SDEs en dos partes, ajuste (*tuning*) y aprendizaje:

- Ajuste evolutivo. Se aplica un proceso de ajuste *a posteriori* sobre la base de conocimiento (BC) para mejorar el rendimiento del SD.
- Aprendizaje evolutivo. Consiste en aprender componentes de la BC (incluyendo aspectos como obtener un mecanismo de inferencia adaptativo).

[5] http://www.keel.es/algorithms.php

Es posible clasificar los SDEs existentes en KEEL[5] con respecto a estas partes o a los componentes del SD involucrados en el proceso evolutivo [12]:

- Ajuste evolutivo de parámetros de la BC: P.ej. un método de ajuste para la obtención de reglas de control difusas de alto rendimiento por medio de AGs [13].
- Aprendizaje evolutivo de pesos para reglas: P.ej. un AG para aprendizaje de pesos y selección de reglas [4].
- Aprendizaje evolutivo de la base de reglas: P.ej. el método Pittsburgh pionero propuesto por Thrift [21], SLAVE [11], y otros.
- Aprendizaje evolutivo de SDs TSK: P.ej. un proceso evolutivo con identificación local de prototipos para obtener un conjunto inicial de reglas TSK locales basadas en semántica [2].
- Aprendizaje simultáneo de componentes de la BC: P.ej. un proceso multi-etapa híbrido de AG y estrategias de evolución para el diseño aproximado de SDs [6].
- La primera contribución en el área de selección evolutiva de reglas, propuesta por Ishibuchi [15].

4 DESCRIPCIÓN DE KEEL

KEEL es una herramienta software para la preparación de AEs para problemas de minería de datos. Permite ejecutarlos dentro del propio entorno (ejecución *on-line*) o generarlos para una ejecución posterior en distintas máquinas (ejecución *off-line*) . La versión actual de KEEL está compuesta por los siguientes módulos (ver Figura 1):

Figura 1: Pantalla principal de KEEL

- *Tratamiento de datos* (Data Management): Este módulo contiene una serie de herramientas de tratamiento de datos: Importación, exportación, edición y visualización de datos, aplicación de transformaciones, etc.
- *Experimentos* (Experiments): Este módulo genera procedimientos de análisis y evaluación automática de algoritmos *off-line*, proporcionando numerosas opciones: Tipo de validación, tipo de aprendizaje (clasificación, regresión, aprendizaje no-supervisado), etc.
- *Educacional* (Educational): Este módulo realiza la ejecución de algoritmos *on-line*. Tiene una estructura similar al módulo anterior, pero permite diseñar experimentos para ser ejecutados paso a paso, con propósitos educativos.

Esta estructura hace que KEEL sea interesante para distintos tipos de usuarios, dependiendo de sus necesidades. A nivel general, las principales características de KEEL son las siguientes:

- Incluye algoritmos de aprendizaje de modelos predictivos, de preprocesamiento (transformación de datos, discretización, selección de instancias y selección de características) y postprocesamiento, con especial atención a las propuestas evolutivas.
- Posee una librería estadística para analizar resultados de algoritmos. Dicha librería contiene tests estadísticos para analizar la bondad de los resultados obtenidos y también para realizar comparaciones paramétricas y no paramétricas.
- Incluye una Librería de Programación de Algoritmos Evolutivos en Java, *Java Class Library for Evolutionary Computation* (JCLEC) [22].
- Ofrece al usuario una interfaz amigable, orientada al análisis de algoritmos.
- Permite crear experimentaciones conteniendo múltiples conjuntos de datos y algoritmos conectados entre sí. Los experimentos son generados mediante scripts independientes de la interfaz de usuario, para permitir una ejecución *off-line* en la misma u otras máquinas.
- KEEL también permite ejecutar experimentos en modo *on-line*, dando un soporte educativo a la tarea de comprender el funcionamiento de los algoritmos incluidos.

A continuación describimos brevemente las principales características del módulo *Experimentos*. Este módulo es el encargado de generar los ficheros y la estructura de directorios requerida para ejecutarlo en cualquier máquina con Java (ver Figura 2).

Los experimentos son modelados gráficamente, basados en flujos de datos y representados mediante grafos y conexiones arco-nodo. Permiten escoger el tipo de validación a emplear (validación cruzada de k-particiones o validación cruzada 5x2) y el tipo de aprendizaje (regresión, clasificación o no supervisado).

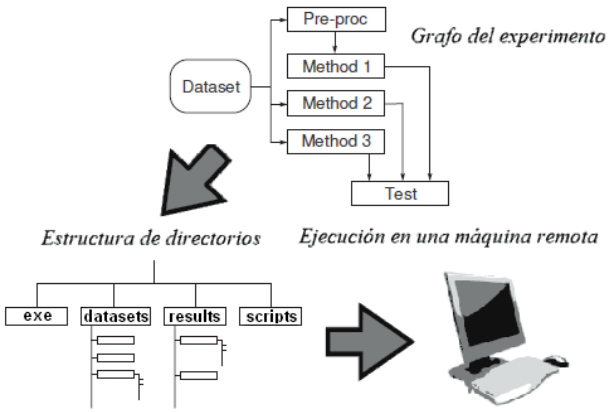

Figura 2: Diseño de experimentos

Después se seleccionan los conjuntos de datos, se arrastran los métodos seleccionados al espacio de trabajo y se establecen conexiones entre métodos y datos (incluyendo, si se desea, test estadísticos de análisis de resultados). En cualquier momento, cualquier componente puede ser configurado haciendo doble click en su nodo correspondiente. La Figura 3 muestra un ejemplo de experimento de clasificación, siguiendo la metodología MOGUL [8] y empleando un método de informe para obtener un resumen de resultados. También se muestra la ventana de configuración del método MOGUL en dicha figura.

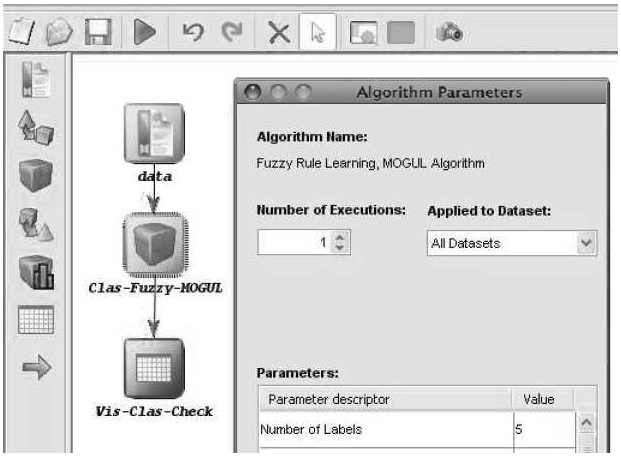

Figura 3: Ejemplo de experimento y de ventana de configuración de un método.

Cuando el experimento ha sido diseñado, el usuario puede escoger entre guardarlo en un fichero XML u obtener un fichero .zip. Éste último contendrá los ficheros requeridos para ejecutar el experimento en una máquina independiente con Java (los conjuntos de datos, ficheros .jar de los algoritmos, ficheros XML de configuración y la herramienta *RunKeel*, un entorno simple de scripting que interpreta scripts de configuración de KEEL en formato XML).

5 CASO DE ESTUDIO

En esta sección presentamos un ejemplo de creación de experimentos en KEEL. Este estudio está centrado en la comparación de dos algoritmos de aprendizaje basados en reglas difusas sobre 12 problemas de clasificación. La Tabla 1 muestra dichos problemas.

Tabla 1: Problemas empleados.

Problema	#Ejemplos	#Atr.	#Clases
Bupa	345	6	2
Cleveland	297	13	5
Ecoli	336	7	8
Glass	214	9	7
Haberman	306	3	2
Iris	150	4	3
Monk-2	432	6	2
New-thyroid	215	5	3
Pima	768	8	2
Vehicle	846	18	4
Wine	178	13	3
Wisconsin	683	9	2

Los métodos considerados son el algoritmo SLAVE [11] y el algoritmo Chi et al. [5] con pesos en la reglas [16](denominado algoritmo Chi-RW). El algoritmo SLAVE es un algoritmo de aprendizaje inductivo de la base de reglas, basado en el enfoque IRL y el algoritmo Chi-RW es una adaptación del algoritmo de Wang-Mendel [24] a problemas de clasificación.

Para desarrollar el experimento se ha considerado un esquema de validación cruzada de 10 particiones. Debido a que SLAVE es un método probabilístico, ha sido ejecutado 5 veces por cada partición (un total de 50 ejecuciones).

Las particiones lingüísticas iniciales están compuestas por *tres términos lingüísticos*, con funciones de pertenencia triangulares uniformemente distribuidas. Los valores considerados para los parámetros de entrada de cada método son:

- SLAVE: *Tamaño de la población* = 100, *Iteraciones sin cambios* = 500, *Probabilidad de mutación* = 0.01, *Probabilidad de cruce* = 1.0.
- Chi-RW: *Cálculo del grado de compatibilidad*: T-norma producto, *Pesos en reglas*: Factor de certeza penalizado, *Combinación del grado de compatibilidad y de pesos en reglas*: T-norma producto, *Método de inferencia*: mejor regla.

Para realizar este experimento en KEEL, primero seleccionamos la opción *Experiments* del menú principal de KEEL, definimos el experimento como problema de clasificación e indicamos el procedimiento de validación cruzada de 10 particiones. Después, el primer paso de configuración del grafo del experimento consiste en seleccionar los conjuntos de datos de la Tabla

1. Las particiones generadas por KEEL son estáticas, por lo que permiten ejecutar experimentos similares posteriormente, sin alterar las particiones realizadas. Opcionalmente, el usuario puede crear sus propias particiones mediante el módulo de *Tratamiento de datos*.

El grafo de la Figura 4 representa el flujo de datos y resultados entre algoritmos y técnicas estadísticas. Un nodo puede representar un flujo inicial de datos, un algoritmo de preprocesamiento o postprocesamiento, un algoritmo de aprendizaje, un test o un módulo de visualización de resultados. Todos sus parámetros pueden ser ajustados haciendo doble click en el nodo (ver Sección 4), incluyendo el número de ejecuciones para algoritmos probabilísticos (5 en el caso de SLAVE). Por otro lado, los arcos que conectan dos nodos representan una relación entre ellos (intercambio de datos o de resultados en entrenamiento y test).

Las salidas de los algoritmos de aprendizaje se emplean como entradas para los métodos de visualización y test. El módulo *Vis-Clas-Tabular* recibe dichos resultados y genera ficheros de salida con varias medidas de rendimiento calculadas a partir de ellos. La Figura 4 también muestra el nodo *Stat-Clas-Wilcoxon*, que realiza una comparación estadística entre resultados obtenidos sobre múltiples conjuntos de datos, el test de Wilcoxon [9].

Una vez que el grafo ha sido definido, podemos guardar el experimento en un fichero .zip para su ejecución *off-line*. Dicho experimento estará compuesto por un conjunto de scripts XML y un programa .jar que los ejecuta. En el directorio *results* se almacenarán los resultados de cada método durante su ejecución. P. ej., los ficheros situados en el directorio asociado al algoritmo de aprendizaje difuso contendrán BCs. En el caso de los métodos de visualización, sus directorios contendrán los ficheros de resultados. Los resultados obtenidos del análisis de los métodos se muestran en la Tabla 2, donde $\#Reg$ es el número medio de reglas y Acc_{Ent} y Acc_{Tst} son, respectivamente, el porcentaje de acierto medio obtenido sobre los datos de entrenamiento y test.

En el caso de los módulos de test, el directorio *Stat-*

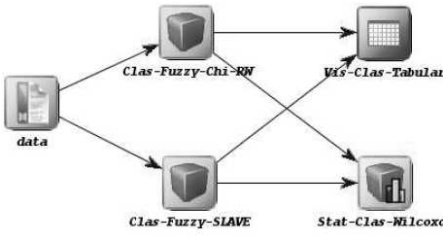

Figura 4: Grafo del experimento del caso de estudio

Tabla 2: Resultados obtenidos.

Problema	CHI-RW			SLAVE		
	Acc_{Ent}	Acc_{Tst}	#Reg	Acc_{Ent}	Acc_{Tst}	#Reg
Bupa	59.87	57.87	43.3	60.60	58.28	3.9
Cleveland	91.25	39.09	230.5	79.05	53.52	39.6
Ecoli	79.53	78.33	43.5	82.75	79.10	11.7
Glass	65.99	60.04	27.1	71.70	62.16	12.8
Haberman	74.26	73.19	16.7	74.90	74.35	2.9
Iris	93.78	94.00	14.7	96.98	94.86	3.3
Monk-2	100.0	48.84	301.8	67.36	67.23	1.3
New-thyroid	85.94	84.24	18.4	89.82	87.99	3.9
Pima	75.62	72.40	105.2	75.45	74.44	4.7
Vehicle	65.92	60.77	227.8	66.31	60.18	20.8
Wine	98.75	92.68	121.1	94.60	90.42	4.3
Wisconsin	98.08	91.21	224	97.16	95.72	5.1
Average	**82.42**	71.06	114.51	79.72	**74.85**	**9.53**

Clas-Wilcoxon contendrá los resultados del test estadístico. Dichos módulos producen su salida en formato de texto y LaTeX , generando las tablas asociadas al test. Para este estudio se ha obtenido la Tabla 3, donde los rangos positivos corresponden a SLAVE y los negativos a Chi-RW.

Tabla 3: Test estadístico.

Comparación	R^+	R^-	p-valor
SLAVE vs. Chi-RW	69.0	9.0	0.019607

Analizando los resultados de las Tablas 2 y 3 podemos destacar que, aunque el algoritmo SLAVE no obtiene la mejor precisión en entrenamiento en todos los problemas, su precisión en test es mejor en 11 de los 12 conjuntos, y los modelos obtenidos tienen un menor número de reglas. Además, el análisis estadístico (test de Wilcoxon de pares) obtiene que el algoritmo SLAVE mejora claramente al algoritmo Chi-RW, asumiendo un alto nivel de significancia $p = 0.0196$.

6 CONCLUSIONES

KEEL es una herramienta software capaz de preparar adecuadamente AEs para problemas de minería de datos, incluyendo algoritmos de SDEs. Esta herramienta libera a los investigadores de la mayoría del trabajo técnico y les permite centrarse en el análisis de sus nuevos algoritmos en comparación con los ya existentes, permitiendo que incluso investigadores con conocimientos básicos en Lógica Difusa y Computación Evolutiva puedan emplear los SDEs en su trabajo.

Hemos mostrado un caso de estudio para ilustrar el proceso de generación de experimentos en KEEL, incluyendo el empleo de tests estadísticos.

La herramienta KEEL está siendo continuamente actualizada y mejorada. Actualmente, estamos desarrollando herramientas de visualización para los módulos on-line y off-line. Además, se está trabajando en el desarrollo de un repositorio de conjuntos de datos que

incluya particiones y resultados de la aplicación de algoritmos sobre dichos conjuntos, *KEEL-dataset*[6].

Agradecimientos

Este trabajo ha sido financiado por el Ministerio Español de Educación y Ciencia bajo los proyectos TIN2008-06681-C06-01 y TIN2008-06681-C06-04.

Referencias

[1] T. Abeel, Y.V. de Peer, Y. Saeys. Java-ML: A machine learning library. *Journal of Machine Learning Research*, 10, Pág. 931-934, 2009.

[2] R. Alcalá, J. Alcala-Fdez, J. Casillas, O. Cordón, F. Herrera. Local Identification of Prototypes for Genetic Learning of Accurate TSK Fuzzy Rule-Based Systems. *International Journal of Intelligent Systems*, 22:9, Pág. 909-941, 2007.

[3] R. Alcalá, J. Casillas, O. Cordón, A. González, F. Herrera. A Genetic Rule Weighting and Selection Process for Fuzzy Control of Heating, Ventilating and Air Conditioning Systems. *Engineering Applications of Artificial Intelligence*, 18:3, Pág. 279-296, 2005.

[4] R. Alcalá, O. Cordón, F. Herrera. Combining Rule Weight Learning and Rule Selection to Obtain Simpler and More Accurate Linguistic Fuzzy Models. *in Modelling with Words*, LNCS Vol. 2873, Pág. 44-63, 2003.

[5] Z. Chi, J. Wu, H. Yan. Handwritten Numeral Recognition Using Self-Organizing Maps and Fuzzy Rules. *Pattern Recognition*, 28:1, Pág. 59-66, 1995.

[6] O. Cordón, F. Herrera. Hybridizing Genetic Algorithms with Sharing Scheme and Evolution Strategies for Designing Approximate Fuzzy Rule-Based Systems. *Fuzzy Sets and Systems*, 118:2, Pág. 235-255, 2001.

[7] O. Cordón, F. Herrera, F. Hoffmann, L. Magdalena. *Genetic fuzzy systems. Evolutionary tuning and learning of fuzzy knowledge bases*. World Scientific, Singapore, 2001.

[8] O. Cordón, M.J. del Jesus, F. Herrera, M. Lozano. MOGUL: A Methodology to Obtain Genetic fuzzy rule-based systems Under the iterative rule Learning approach. *International Journal of Intelligent Systems*, 14:11, Pág. 1123-1153, 1999.

[9] J. Demšar. Statistical Comparisons of Classifiers over Multiple Data Sets. *Journal of Machine Learning Research*, 7, Pág. 1-30, 2006.

[10] A.E. Eiben, J.E. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, Berlin, 2003.

[11] A. González, R. Perez. SLAVE: A genetic learning system based on an iterative approach. *IEEE Transactions on Fuzzy Systems*, 7:2, Pág. 176-191, 1999.

[12] F. Herrera. Genetic Fuzzy Systems: Taxonomy, Current Research Trends and Prospects. *Evolutionary Intelligence*, 1, Pág. 27-46, 2008.

[6] http://www.keel.es/datasets.php

[13] F. Herrera, M. Lozano, J.L. Verdegay. Fuzzy Logic Controllers by Genetic Algorithms. *International Journal of Approximate Reasoning*, 12, Pág. 299-315, 1995.

[14] J.H. Holland, J.S. Reitman. Cognitive systems based on adaptive algorithms. *SIGART Bull*, 63, Pág. 49-49, 1977.

[15] H. Ishibuchi, K. Nozaki, N. Yamamoto, H. Tanaka. Selecting Fuzzzy If-Then Rules for Classification Problems Using Genetic Algorithms. *IEEE Transactions on Fuzzy Systems*, 3:3, Pág. 260-270, 1995.

[16] H. Ishibuchi, T. Yamamoto. Rule weight specification in fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, 13:4 , Pág. 428-435, 2005.

[17] K.A. de Jong, W.M. Spears, D.F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13, Pág. 161-188, 1993.

[18] J.S. Shieh, M.H. Kao, C.C. Liu. Genetic fuzzy modelling and control of bispectral index (BIS) for general intravenous anaesthesia. *Medical Engineering and Physics*, 28:2, Pág. 134-148, 2006.

[19] S. Smith. *A learning system based on genetic algorithms*. Ph.D. thesis, University of Pittsburgh, 1980.

[20] S. Sonnenburg, M.L. Braun, Ch.S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K-R. Müller, F. Pereira, C.E. Rasmussen, G. Rätsch, B. Schölkopf, A. Smola, P. Vincent, J. Weston, R.C. Williamson. The Need for Open Source Software in Machine Learning. *Journal of Machine Learning Research*, 8, Pág. 2443-2466, 2007.

[21] P. Thrift. Fuzzy logic synthesis with genetic algorithms. *in 4th International Conference on Genetic Algorithms (ICGA91)*, Pág. 509-513, 1991.

[22] S. Ventura, C. Romero, A. Zafra, J.A. Delgado, C. Hervás. JCLEC: A Java framework for Evolutionary Computation. *Soft Computing*, 12:4, Pág. 381-392, 2008.

[23] G. Venturini. SIA: a supervised inductive algorithm with genetic search for learning attribute based concepts. *in European Conference on Machine Learning*, LNCS Vol. 669, Pág. 280-296, 1993.

[24] L. X. Wang, J. M. Mendel. Generating Fuzzy Rules by Learning from Examples. *IEEE Transactions on Systems, Man and Cybernetics*, 22:6, Pág. 1414-1427, 1992.

[25] S. W. Wilson. Classifier Fitness Based on Accuracy. *Evolutionary Computation*, 3:2, Pág. 149-175, 1995.

[26] I. H. Witten, E. Frank. Data Mining: Practical Machine Learning Tools and Techniques. Second edition, Morgan Kaufmann, 2005.

[27] M.L. Wong, K.S. Leung. *Data mining using grammar based genetic programming and applications*. Kluwer Academic Publishers, Norwell, 2000.