UNIVERSIDAD DE JAÉN

Departamento de Informática

Métodos híbridos evolutivos cooperativos-competitivos para el diseño de Redes de Funciones de Base Radial

MEMORIA DE TESIS DOCTORAL PRESENTADA POR

María Dolores Pérez Godoy

como requisito para optar al grado de Doctor en Informática

DIRECTORES

D. Antonio Jesús Rivera Rivas **D^a. María José del Jesus Díaz**

DEPARTAMENTO DE INFORMÁTICA

Jaén, Abril 2010

La memoria titulada *"Métodos híbridos evolutivos cooperativos-competitivos para el diseño de Redes de Funciones de Base Radial"*, que presenta María Dolores Pérez Godoy para optar al grado de doctor, ha sido realizada dentro del programa de doctorado "Informática" del Departamento de Informática de la Universidad de Jaén, bajo la dirección de los doctores D. Antonio Jesús Rivera Rivas y D^a. María José del Jesus Díaz.

Jaén, Abril 2010

La doctoranda

Fdo. María Dolores Pérez Godoy

El director

Fdo. Antonio Jesús Rivera Rivas

La directora

Fdo. María José del Jesus Díaz

Tesis Doctoral parcialmente subvencionada por la Comisión Interministerial de Ciencia y Tecnología (CICYT) con el proyecto **TIN2008-06681-C06-02** y por el Plan de Investigación Andaluz **TIC-3928**.

Agradecimientos

Esta sección, aunque aparece al principio de la memoria ha sido la última en ser escrita y la que más trabajo me ha costado, pues resulta difícil expresar sentimientos y más aún intentar reunir en unas pocas palabras los apoyos y ánimos de todos aquellos que me rodean, no sólo en este punto de mi camino profesional sino en el día a día de mi vida.

Quiero agradecerles a mis directores Antonio Jesús y María José todo el apoyo que me han brindado para que consiguiera realizar el trabajo que se recoge en esta memoria. Gracias también por ofrecerme vuestra amistad, es una gran satisfacción trabajar con vosotros.

También quiero expresar mi agradecimiento a todos los compañeros del Departamento de Informática que me han apoyado y me han transmitido su cariño. A Ángel Luis y Pedro J. su ayuda en algunas cuestiones sobre Latex. A Salva y Alberto por sus sabios consejos sobre tests estadísticos. A Luis por el ánimo que me ha dado y sus constantes ofrecimientos de ayuda.

Gracias a mis amigos del café, Paco, Chequin, Antonio y Pedro, porque siempre consiguen que vuelva la sonrisa a mi cara aún en esos días en los que mi ánimo está por los suelos. También a Lidia que ya se está haciendo asidua a estas tertulias, a Andrés que las ha sustituido por su afán de arte y a María José y Jose Ramón que intervienen cuando sacan un hueco de su apretada agenda.

Gracias a mis abuelos Antonio, Loles, Ana, y Francisco (a este último no lo conocí pero otros me han transmitido sus recuerdos) por su cariño, sus enseñanzas y refranes que contienen el saber de la experiencia. En cierta manera mi abuelo sentó las bases de mi camino de aprendizaje académico ya que me enseñó a leer.

Gracias a mis padres Paquete y Pepa que nos han enseñado que con esfuerzo y honestidad se logra avanzar y que siempre hay que buscar el lado positivo de las cosas. La ayuda de mi madre también ha sido muy importante para la realización de este trabajo, ella me ha apoyado y ofrecido su tiempo para que yo aumentara el mío. Por desgracia, mi padre no ha podido ver la conclusión de esta etapa de trabajo pero estoy segura de lo orgulloso que hubiera estado. A mis hermanos Ani, Francis y Antonio por sus apoyos, sus ánimos y porque consiguen que los problemas y agobios se diluyan con unas risas. A Puri, Antonio, María y mis sobrinos que también forman parte de ese mundo virtual que formamos cuando estamos juntos, en el que las dificultades parecen menguar.

Finalmente, quiero agradecer a Antonio todo su apoyo en cada momento de mi vida laboral y personal, él es que ha aguantado la mayoría de mis nervios y ansiedades en esta última etapa. A mis hijos que dan color y alegría a mi vida y que tanto me han acompañado en estos últimos meses, sentándose conmigo frente al ordenador “para que les enseñara a hacer una tesis y así me podrían ayudar”.

Muchas gracias a todos.

Índice general

Índice de figuras

Índice de tablas

Prólogo

Motivación

Las nuevas tendencias en generación y uso de información y el interés en la obtención de conocimiento, generan nuevos retos en investigación dentro del campo conocido como extracción de conocimiento en bases de datos (KDD, *Knowlege Discovery in Databases*) y en particular en una de sus etapas, la minería de datos (*Data Mining*).

Es necesario que el conocimiento detectado sea potencialmente útil, que se extraigan patrones comprensibles para las personas que finalmente utilizarán este conocimiento y que normalmente no son expertos en minería de datos. En este aspecto, el desarrollo de algoritmos de minería de datos que proporcionen información compacta, sencilla y fácil de interpretar tendrá un impacto positivo en este área. En minería de datos predictiva, área en la que se centra este trabajo, el objetivo es proporcionar conocimiento para predecir, es importante la interpretabilidad del conocimiento extraído para justificar la predicción o lo que es igual, conocer la forma en que ésta se realiza para entender mejor el problema.

El proceso KDD [Fayyad y otros, 1996a,b] está formado por un conjunto de pasos interactivos e iterativos, entre los que se incluye el preprocesamiento de los datos, la búsqueda de patrones de interés con una representación particular y la interpretación de los patrones obtenidos. La segunda de las etapas comentadas es la que se conoce como minería de datos y en muchos contextos es el término que ha tenido más aceptación por lo que, con frecuencia, se utiliza para hacer referencia a todo el proceso de KDD [Han y Kamber, 2000; Witten y E.Frank, 2005].

La minería de datos es un campo interdisciplinar cuyo objetivo general es producir nuevo conocimiento que pueda resultar útil mediante la construcción de un modelo a partir de los datos existentes. Son muchas las tareas que puede abordar la minería de datos. La mayoría de ellas se pueden agrupar en tres grandes categorías:

- Predictivas: el objetivo es pronosticar o predecir el comportamiento futuro del modelo construido en base a los datos disponibles.
- Descriptivas: el objetivo es obtener patrones que resuman las relaciones subyacentes entre los datos.
- Híbridas: con características de las dos categorías anteriores.

Dada la importancia que tiene la automatización de la extracción de conocimiento, en la última década se han desarrollado múltiples propuestas de algoritmos de minería de datos predictiva, descriptiva e híbrida y con distintos tipos de modelos, y en particular en el campo de la minería de datos predictiva, múltiples desarrollos para problemas de clasificación, de regresión numérica y de predicción de series temporales.

Para desarrollar propuestas de algoritmos de minería de datos se utilizan diferentes técnicas. Entre ellas toman especial interés las conocidas como técnicas *soft-computing* [Tettamanzi y Tomassini, 2001] o técnicas de *computación flexible*. El auge de este tipo de técnicas radica en su capacidad para trabajar en entornos de imprecisión, lo que les confiere una gran capacidad de adaptación y hace posible su aplicación a la resolución de problemas en entornos cambiantes de forma robusta y con bajo coste. Estas situaciones de imprecisión y cambio suelen ocurrir en la mayoría de los problemas reales.

Según Zadeh [Zadeh, 1994] el término *soft-computing* engloba un conjunto de técnicas que son tolerantes con la incertidumbre, la imprecisión y la verdad parcial. Se considera que la lógica difusa, las redes neuronales, la computación evolutiva y el razonamiento probabilístico, son las principales técnicas *soft-computing*. Estas tecnologías no deben considerarse como excluyentes en la resolución de problemas [Bonissone, 2000; Tettamanzi y Tomassini, 2001], sino que deben cooperar entre sí para lograr sistemas híbridos de componentes especializados. En esta memoria se describen propuestas desarrolladas para minería de datos descriptiva bajo este enfoque, hibridando redes neuronales, algoritmos evolutivos y lógica difusa.

Las Redes Neuronales Artificiales (RNA), utilizadas inicialmente por [Rosenblatt, 1957; Widrow y Hoff, 1960], son un paradigma de aprendizaje y procesamiento automático inspirado en el sistema nervioso de los animales. Constituyen una importante herramienta dentro de la minería de datos debido a algunas de sus características tales como: su no linealidad, capacidad de aproximación universal, tolerancia al ruido, capacidad de procesamiento paralelo y capacidad de adaptación mediante aprendizaje.

Uno de los paradigmas más populares dentro del campo de las redes neuronales lo constituyen las Redes de Funciones de Base Radial (RBFNs) [Broomhead y Lowe, 1988].

La computación evolutiva, con sus distintos paradigmas, [Holland, 1975; Fogel, 1962; Rechenberg, 1971; Schwefel, 1975; Koza, 1992], proporciona algoritmos estocásticos de búsqueda inspirados en el proceso de evolución de Darwin [Darwin, 1859]. Estos algoritmos se aplican a la minería de datos debido a su robustez y a que sus técnicas adaptativas de búsqueda permiten realizar una búsqueda global en todo el espacio de soluciones.

La lógica difusa, introducida por Zadeh en 1965 [Zadeh, 1965] aporta un modelo de inferencia capaz de manejar conocimiento impreciso y cuantitativo más cercano al razonamiento humano. Las características más atractivas de la lógica difusa son su flexibilidad, su tolerancia a la imprecisión, su capacidad para modelar problemas no lineales y su forma de expresión similar al lenguaje natural, importantes en minería de datos.

Como resumen de lo anterior, se puede concluir que actualmente la automatización de la extracción de conocimiento es un campo en auge y una de sus principales etapas es la minería de datos. Es importante seguir avanzado en la construcción de modelos de minería de datos que permitan extraer conocimiento sencillo, fácil de interpretar y utilizar por el usuario final. Dentro de las técnicas utilizadas en minería de datos, las técnicas *soft-computing* constituyen uno de los paradigmas más interesantes.

En esta memoria se describen métodos híbridos para dos tipos de tareas de minería de datos predictiva: clasificación y predicción de series temporales.

En el ámbito de la clasificación existe y es muy frecuente en aplicaciones reales, una situación que dificulta el proceso de minería de datos, el desbalanceo de clases entre los ejemplos. Es lo que se conoce como clasificación con clases no balanceadas. En general, los algoritmos de minería de datos tienen problemas para extraer conocimiento correcto de todas las clases (especialmente la minoritaria) y en muchos casos, la clase minoritaria representa el concepto de mayor interés del problema mientras que la clase mayoritaria representa sólo contraejemplos de la anterior.

El objetivo de esta memoria es el diseño y optimización de redes de funciones de base radial (RBFNs) para tareas predictivas. No sólo se va a utilizar una técnica *soft-computing* sino que el modelo producido es una hibridación de tres de ellas. En el nivel más bajo una red neuronal se encarga de trabajar con los datos del problema. Para llevar a acabo el diseño de la red se utilizan la computación evolutiva y la lógica difusa, usadas como *meta-heurísticas* que definen un nivel más elevado.

En particular, la memoria se centra en problemas de clasificación: se analizan los problemas abiertos para el diseño de RBFNs simples y precisas en clasificación; se desarrollan propuestas y se analiza el comportamiento en problemas de clasificación con bases de datos no balanceadas. Además, se analiza y se adapta la propuesta para su aplicación a problemas de predicción de series temporales.

Objetivos

El principal objetivo de esta tesis es el desarrollo de un método híbrido, evolutivo, con enfoque cooperativo-competitivo, para el diseño de Redes de

Funciones de Base Radial, aplicado a problemas de clasificación.

En particular, un método *sotf-computing* en el que se hibridan diferentes técnicas como las redes neuronales, la computación evolutiva y la lógica difusa. El método debe obtener Redes de Funciones de Base Radial mediante un diseño evolutivo de las mismas, siguiendo un enfoque cooperativo-competitivo. Dentro del diseño evolutivo se utiliza la lógica difusa para representar conocimiento experto en el proceso de diseño.

Para alcanzar este objetivo general, se abordarán los siguientes objetivos específicos:

- Revisión de otras propuestas de diseño de RBFNs existentes. Se realizará un estudio de los modelos y paradigmas propuestos, en la bibliografía especializada, para solucionar los problemas de diseño de RBFNs.

- Análisis de los problemas a resolver en la tarea de clasificación tanto en bases de datos balanceadas como en bases de datos no balanceadas. Los problemas que se presentan en las bases de datos no balanceadas son debidos a que los ejemplos de las diferentes clases siguen diferentes distribuciones. Este tipo de bases de datos no balanceadas aparecen en muchos problemas reales.

- Estudio de las medidas utilizadas para el tratamiento de los datos tanto numéricos como nominales. Se pretende el uso de funciones de distancia adecuadas, de forma que se consiga que la pérdida de información sea mínima.

- Desarrollo de un método híbrido evolutivo para obtención de RBFNs

compactas, con un adecuado nivel de precisión y simplicidad en problemas de clasificación. Es decir, RBFNs que con un bajo número de neuronas consigan una buena generalización del conjunto de datos.

- Aplicación del modelo para la obtención de RBFNs compactas y precisas en problemas de clasificación en bases de datos no balanceadas. Análisis de la influencia de métodos de pre-procesamiento de datos para mitigar el efecto del desbalanceo existente.
- Análisis formal de los resultados obtenidos mediante técnicas estadísticas que validen nuestro modelo frente a otros ya existentes.
- Análisis y adaptación de la propuesta para problemas de predicción de series temporales.

Contenido de la Memoria

Esta memoria se divide en seis capítulos cuyo contenido se resume a continuación.

En el Capítulo 1, se describe la extracción de conocimiento en bases de datos así como cada una de sus fases. La fase de minería de datos se describe con mayor nivel de detalle, dado que el modelo que se presenta en esta memoria se encuadra dentro de ella. Se enumeran las tareas de minería de datos así como las técnicas utilizadas para abordar éstas. Dentro de las técnicas usadas en minería de datos, destacan los algoritmos evolutivos, redes neuronales y lógica difusa, todas ellas técnicas *soft-computing*, dado que son las técnicas que el modelo propuesto va a utilizar.

El modelo propuesto en la memoria tiene como objetivo el diseño de

redes de funciones de base radial, un modelo de red neuronal artificial. Por tanto es necesario realizar un estudio sobre este tipo de redes así como una revisión de los modelos existentes en la bibliografía para el diseño de las mismas. Este estudio se lleva a cabo en el Capítulo 2.

La idea inicial en la construcción del modelo que se propone, es la de diseñar un modelo aplicado a la tarea de clasificación. En el Capítulo 3, se presenta el método propuesto. Los resultados obtenidos se comparan con los obtenidos mediante distintos métodos basados en técnicas *soft-computing*. Por último, se realiza un análisis formal de los resultados obtenidos mediante un conjunto de test estadísticos.

Dentro de los problemas de clasificación, toma importancia la clasificación en bases de datos desbalanceadas, es decir, la clasificación cuando la distribución de los ejemplos de las clases es muy diferente. En el Capítulo 4 se presenta el problema de clasificación en bases de datos no balanceadas, se estudian los métodos de preprocesamiento utilizados para tratar dichas bases de datos y se aplica el algoritmo a bases de datos no balanceadas. Al igual que en el capítulo anterior, los resultados obtenidos se comparan con los de otros métodos y se realiza un análisis estadístico de los mismos.

En el Capítulo 5 se estudian los problemas de predicción de series temporales y los métodos estadísticos clásicos utilizados para el análisis de las mismas (destacando el método ARIMA) y se adapta el método CO^2RBFN para extraer conocimiento en este tipo de problemas. Posteriormente se analiza el comportamiento del método aplicado a un conjunto de series temporales estudiadas en la literatura. Para finalizar, el algoritmo se aplica a un problema real, la predicción del precio del aceite de oliva virgen extra. En todas las experimentaciones los resultados obtenidos se comparan con

las otros métodos.

Para concluir la memoria, en el Capítulo 6 se presentan las conclusiones obtenidas sobre el comportamiento del método, las líneas de trabajo futuro y se enumeran las publicaciones asociadas al trabajo desarrollado en esta memoria.

Capítulo 1

Introducción

Extraer conocimiento a partir de datos es un proceso que se viene haciendo a lo largo de la historia y que en la actualidad está teniendo una gran importancia debido, entre otras cosas, a los grandes volúmenes de información que se tienen que manejar. Surge así la necesidad de explorar nuevas formas de extracción automática de conocimiento a partir de la información disponible.

Este proceso, denominado formalmente descubrimiento de conocimiento en bases de datos, está formado por un conjunto de pasos interactivos e iterativos, entre los que se incluye el pre-procesamiento de los datos para corregir los posibles datos erróneos, incompletos o inconsistentes, la reducción del número de registros y/o características encontrando los más representativos, la búsqueda de patrones de interés con una representación particular y la interpretación de estos patrones.

La fase de búsqueda de patrones de interés, es conocida como minería de datos y en muchos contextos dicho término es el que ha tenido más aceptación por lo que, con frecuencia, se utiliza éste para hacer referencia a

todo el proceso completo de descubrimiento de conocimiento.

En este capítulo se describirán brevemente las distintas fases del proceso de descubrimiento de conocimiento y se centrará la atención en la fase de minería de datos, sus tareas y técnicas. Se explicarán con más detalle técnicas *soft-computing* como la computación evolutiva, lógica difusa y redes neuronales.

1.1. Descubrimiento de conocimiento en bases de datos y minería de datos

Desde sus orígenes, el hombre ha tenido la necesidad de manejar información. Podemos decir que el tratamiento de la información es casi tan antiguo como el hombre y dicho tratamiento va evolucionando y haciéndose más sofisticado con el transcurso del tiempo. A lo largo de la historia se han ido creando máquinas y métodos que permiten procesar la información, de hecho la disciplina de la Informática surge para poder tratar la información de forma automática.

La cantidad de información disponible crece espectacularmente y surgen estructuras de almacenamiento como las Bases de Datos y métodos de consulta asociados que permiten extraer información resumida. No obstante, estos métodos sólo generan información resumida de forma previamente establecida, poco flexible y poco escalable a grandes volúmenes de datos. Ante las nuevas demandas que surgen debido al crecimiento masivo y la diversidad de fuentes de información, se da un paso más y surge el almacén de datos (*data warehouse*). Se trata de un almacén de fuentes

heterogéneas de datos, integrados y organizados bajo un esquema unificado para facilitar su análisis y dar soporte a la toma de decisiones. Esta tecnología incluye operaciones de procesamiento analítico en línea (*On-Line Analytical Processing*, OLAP), técnicas de análisis como pueden ser el resumen, consolidación o agregación, así como la posibilidad de ver la información desde distintas perspectivas. Estas técnicas son capaces de obtener informes avanzados a base de agregar los datos de cierta forma compleja pero ya predefinida. Sin embargo, no permiten extraer de los datos reglas, patrones, pautas, en general conocimiento que se pueda aplicar a otros datos. Las nuevas necesidades parten de que la importancia no radica en los datos como tales sino en el conocimiento que se puede extraer a partir de ellos y aún más, en que dicho conocimiento se pueda utilizar. Estas necesidades y las limitaciones de las técnicas existentes propician el nacimiento de una nueva generación de herramientas que permitan la extracción de conocimiento útil a partir de la información disponible, englobadas bajo la denominación de *minería de datos* (*Data Mining*).

La minería de datos se distingue de las aproximaciones anteriores en que no obtiene información explícita extensional (datos) sino implícita intensional (conocimiento) y dicho conocimiento no es preestablecido por el usuario sino que es modelo novedoso extraído completamente con la herramienta [Hernández y otros, 2004]. La minería de datos constituye la fase central de un proceso más amplio denominado *descubrimiento de de conocimiento en bases de datos* (*Knowledge Discovery in Databases*, KDD) [Fayyad y otros, 1996b]. Este proceso incluye la aplicación de de distintos métodos de pre-procesamiento orientados a facilitar la aplicación de los algoritmos de minería de datos y métodos de post-procesamiento que refinan

y mejoran el conocimiento descubierto.

1.1.1. Extracción de conocimiento en bases de datos

Tal y como se ha comentado, en la actualidad se manejan grandes volúmenes de información que son recogidos desde diferentes dominios de aplicación. Dicha información se encuentra almacenada en diferentes bases de datos (relacionales, transaccionales, dirigidas a objetos, etc.), en diferentes *data warehouses*, y en otros almacenes (tales como datos www, datos multimedia, etc.). Partiendo de grandes volúmenes de datos que provienen de fuentes heterogéneas, con diferente variabilidad en el nivel de confianza sobre la validez de los mismos, se pretende obtener conocimiento útil que se pueda aplicar a otros conjuntos de datos.

Son muchas las definiciones que aparecen de la extracción o descubrimiento de conocimiento en bases de datos. En [Fayyad y otros, 1996a] se define el KDD como *"el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y, en última instancia, comprensibles a partir de los datos"*. En esta definición, los datos son un conjunto de hechos (como los casos de la base datos), y un patrón es una expresión en algún lenguaje que describe un subconjunto de los datos o un modelo aplicable a este subconjunto. El término proceso implica que el KDD incluye distintos pasos interactivos e iterativos, como la preparación los datos, la búsqueda de patrones, la evaluación del conocimiento, y el refinamiento. Por no trivial, se entiende que es necesaria cierta búsqueda o inferencia; es decir, no es un cálculo directo como puede ser calcular la media de un conjunto de números. Los patrones descubiertos deben ser válidos en nuevos datos con un grado de certidumbre dado. También queremos que los patrones sean

novedosos (al menos para el sistema y preferiblemente para el usuario) y potencialmente útiles, es decir, que supongan algún beneficio y cumplan las metas del usuario. Finalmente, los patrones deberían ser comprensibles, si no inmediatamente, si después llevar a cabo algún tipo de post-procesamiento.

Un proceso KDD, parte de bases datos y permite la selección, preprocesado y transformaciones de éstas; a continuación la aplicación de métodos (algoritmos) de minería de datos para extraer patrones y modelos adecuados; y por último la evaluación y posible interpretación de los resultados de la minería de datos para identificar el conocimiento obtenido a través de los patrones identificados.

En la figura 1.1 se muestra el proceso KDD.

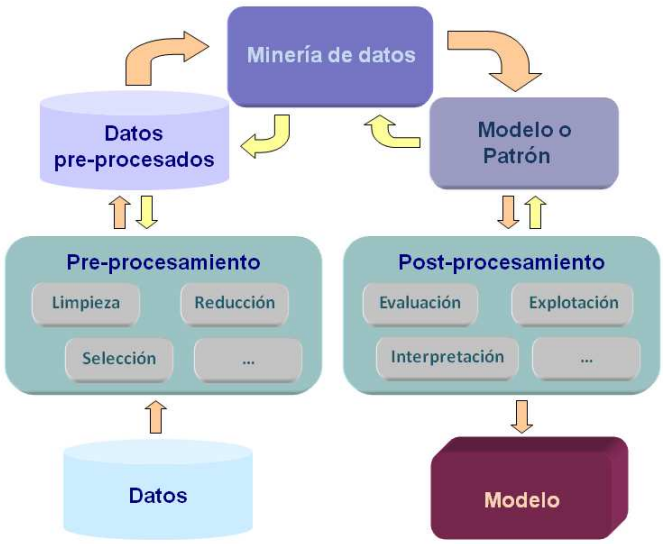


Figura 1.1: *Proceso KDD*

Las etapas en las que se descompone el proceso KDD son las siguientes [Fayyad y otros, 1996b]:

- Pre-procesamiento o preparación de los datos: Los datos almacenados pueden tener valores perdidos, ruido, inconsistencias o un formato inadecuado para su procesamiento. Por todo esto se hace necesario un pre-procesamiento de los mismos para corregir los posibles datos erróneos, incompletos o inconsistentes y reducir el número de registros y/o características encontrando los más representativos que serán aquellos que se procesen. El pre-procesamiento puede implicar una o varias tareas [Ghosh y Jain, 2005]:

  - Limpieza de datos. Se realiza para eliminar ruido e inconsistencias, completar valores perdidos o identificar valores anómalos.
  - Integración de datos. Los datos pueden provenir de fuentes muy diversas y se hace necesaria la integración de los mismos de forma que se eviten conflictos y se resuelva la heterogeneidad semántica.
  - Transformación de datos. Puede que los datos no tengan un formato adecuado para ser procesados, de forma que éstos se tengan que trasformar para adecuarlos a la tarea que se va a aplicar. Las técnicas de transformación pueden ser tales como agregación, generalización, normalización, etc.
  - Discretización. Consiste en transformar los atributos continuos en atributos nominales (o categóricos).
  - Reducción de datos. Se aplican técnicas como reducción de la dimensionalidad (eliminación de algunas características del conjuntos de datos), compresión de los datos, reducción del número de datos (usando alternativas como muestreo, histogramas), etc.
  - Selección de datos. Los datos más relevantes serán seleccionados para el posterior procesamiento de los mismos.

- Minería de datos: Durante esta etapa se seleccionan y aplican algoritmos para encontrar patrones de interés a partir de los datos. El tipo de algoritmo que se elija dependerá de la tarea que se pretenda realizar (clasificación, regresión, agrupamiento, extracción de reglas de asociación, descubrimiento de subgrupos, sumarización, etc.). Es necesario también especificar un criterio de evaluación que permita definir qué modelo es mejor.

- Post-procesamiento: Los patrones extraídos en la fase de minería de datos se han de interpretar apropiadamente, de forma que se extraiga correctamente el conocimiento que conllevan. Medir la calidad de los patrones descubiertos por un algoritmo de minería de datos no es fácil, ya que se pueden utilizar muchos criterios y algunos de ellos son subjetivos. Un patrón se considera interesante si es fácilmente entendible, potencialmente útil, novedoso y es válido sobre los datos de prueba con algún grado de certeza. Según la aplicación a la que se vaya a destinar el modelo, puede interesar mejorar algunos de los criterios vistos y sacrificar ligeramente otros. A menudo el conocimiento descubierto mediante algún algoritmo de minería de datos ha de ser post-procesado para simplificarlo de forma que se aumente la comprensibilidad del mismo por parte de los usuarios. Durante el post-procesamiento se realizan las siguientes acciones:

1.1.2. Minería de datos

Representa la fase más característica del KDD, por lo que a veces se utiliza el término de minería de datos como sinónimo de KDD [Han y Kamber, 2000; Witten y E.Frank, 2005]. Su objetivo es producir nuevo

conocimiento que pueda resultar útil mediante la construcción de un modelo a partir de los datos recopilados para ello. Dicho modelo es una descripción de los patrones o relaciones entre los datos y puede usarse para entender mejor los datos, hacer predicciones o explicar situaciones pasadas. La minería de datos consiste en el *uso de algoritmos concretos que generan una enumeración de patrones a partir de los datos preprocesados* [Fayyad y otros, 1996a].

La minería de datos y en general la extracción de conocimiento, surgen debido a la aparición de cambios en los datos que hacen imposible la aplicación de las técnicas de análisis tradicionales [Tan y otros, 2006]. Algunos de éstos cambios son:

- Escalabilidad: debido a los avances en la generación y recolección de datos se tiene que trabajar con grandes volúmenes de información. Los algoritmos de minería de datos deben ser escalables para poder manejar dichas cantidades masivas de información y para poder tratar con grandes cantidades de atributos.

- Datos complejos y heterogéneos: el análisis tradicional de datos manejaba bases de datos donde los atributos eran del mismo tipo, ahora se necesitan técnicas que permitan manejar datos con atributos heterogéneos (numéricos, continuos), así como con datos más complejos (colecciones de páginas web con texto e hiperenlaces, ...).

- Datos distribuidos: se tiene la necesidad de analizar datos que pueden estar almacenados en una única localización o distribuidos entre varias y heterogéneas fuentes. Así los algoritmos de minería de datos deben intentar reducir la cantidad de comunicación necesaria, consolidar

resultados de múltiples fuentes y garantizar seguridad en los datos.

La minería de datos es un campo multidisciplinar (figura 1.2) que se ha desarrollado en paralelo o como prolongación de otras disciplinas, tales como estadística, inteligencia artificial u optimización. Se puede señalar que las disciplinas más influyentes en la minería de datos son:

Figura 1.2: *Relación de la Minería de Datos con otras disciplinas*

- Bases de datos: en particular técnicas de acceso eficiente a los datos y técnicas de indización para conseguir algoritmos eficaces.
- Estadística: ha proporcionado muchos elementos como teoría del muestreo, diversos tests, métodos de estimación, etc.
- Aprendizaje automático: éste área de la inteligencia artificial que se ocupa de desarrollar algoritmos capaces de aprender, constituye junto con la estadística, el corazón del análisis inteligente de datos. Sus

principios y los de la minería de datos son los mismos: la máquina debe aprender un modelo a partir de los datos y los usa para resolver problemas.

- Visualización de datos: las técnicas de visualización permiten al usuario entender mejor los patrones generados.
- Computación paralela y distribuida: conocimiento para que los algoritmos de minería de datos puedan ser escalables y sus tareas se puedan ejecutar en diferentes procesadores.
- Otras disciplinas: dependiendo del tipo de datos a los que se va a aplicar la minería o del tipo de aplicación se pueden usar también técnicas de otras disciplinas como el lenguaje natural, análisis de imágenes, procesamiento de señales, etc.

Como ya se ha mencionado anteriormente, el objetivo de la minería de datos es producir nuevo conocimiento que pueda resultar útil mediante la construcción de un modelo a partir de los datos recopilados para ello. Dicho modelo es una descripción de los patrones o relaciones entre los datos y puede usarse para entender mejor dichos datos, hacer predicciones o explicar situaciones pasadas. En la práctica, los modelos pueden ser de tres tipos:

- predictivos: encuentran un patrón que permite estimar valores futuros o desconocidos de variables de interés (variables objetivo o dependientes) usando otras variables (variables independientes o predictivas).
- descriptivos: identifican patrones que explican o resumen los datos, es decir, sirven para explorar las propiedades de los datos examinados y no predicen datos nuevos.

- hibridaciones de ambos que tratan de extraer información descriptiva sobre un problema con naturaleza predictiva. En esta categoría entran por ejemplo las técnicas de inducción supervisada de reglas descriptivas [Novak y otros, 2009].

Cuando se va a aplicar minería de datos se ha de determinar el tipo de tarea a la que se dirige, es decir, el tipo de problema que se intenta resolver y después se tendrá que elegir el tipo de técnica más adecuada para abordarla.

El método que se va a proponer en esta memoria entra dentro de la minería de datos predictiva, en la que se engloban varios tipos de tareas, es decir, clases de problemas que se van a intentar resolver. Cada tarea tiene sus propios requerimientos y el modelo obtenido al resolver una de ellas es distinto al obtenido al resolver otra. Las principales tareas dentro de la minería de datos predictiva son [Hernández y otros, 2004]:

- Clasificación: Es probablemente la tarea más estudiada en minería de datos. En ella cada instancia (ejemplo, tupla, registro,...) pertenece a una clase, la cual se indica mediante el valor de un atributo (variables o características). El rango de valores para dicho atributo es un conjunto pequeño y discreto, y cada uno representa una clase. El resto de atributos se utilizan para poder predecir el atributo que representa la clase. El objetivo de esta tarea es predecir la clase a la que pertenecen las nuevas instancias que se puedan presentar. El objetivo del algoritmo es maximizar la precisión en la clasificación de las nuevas instancias, la cual se calcula genéricamente, como el cociente entre las predicciones correctas y el número total de predicciones.
- Regresión: Consiste en aprender una función real que asigna a cada

instancia un valor. En este caso, si lo comparamos con la clasificación, el valor a predecir es numérico. El objetivo en este caso es minimizar el error entre el valor predicho y el valor real.

- Predicción de series temporales: En la predicción se usan valores conocidos actuales para predecir valores futuros. Las observaciones se recogen en periodos sucesivos de tiempo.

- Categorización: en este caso, a diferencia de la clasificación un ejemplo puede pertenecer a más de una categoría. Y el problema a resolver consiste en encontrar cuáles son las categorías a las que pertenece un ejemplo. Al igual que en clasificación, la categorización se puede presentar en forma de categorización suave (cada categoría va acompañada de su certeza) o en forma de estimador de probabilidades (se estima una probabilidad para todas las categorías, en este caso su suma puede ser mayor que 1).

- Preferencias o priorización: consiste en determinar a partir de dos o más ejemplos un orden de preferencia. En este caso lo que se prioriza es el ejemplo completo frente a la priorización de la clase.

De todas estas tareas mencionadas el método propuesto va a abordar la clasificación y la predicción de series temporales.

Técnicas de minería de datos

Dado que la minería de datos es un campo interdisciplinar, los algoritmos de minería de datos están basados en diversos paradigmas, tales como construcción de árboles de decisión, inducción de reglas, algoritmos evolutivos,

redes neuronales, aprendizaje basado en instancias, aprendizaje bayesiano, programación lógica inductiva, métodos basados en núcleo y diferentes tipos de algoritmos estadísticos. Dentro de cada uno de estos paradigmas existen diferentes algoritmos y variaciones de los mismos que presentan restricciones que hacen que el algoritmo sea adecuado para determinados conjuntos de datos y no lo sea para otros. Se puede concluir que no existe algoritmo, ni paradigma, que se pueda aplicar a todos los problemas.

Cada una de las tareas descritas de la minería de datos, puede tener diferentes métodos que la puedan resolver y por otro lado tenemos que el mismo método (o al menos la misma técnica) puede resolver un conjunto de tareas. Esto es debido a que la mayoría de las tareas descritas anteriormente y los métodos, se centran en la idea de aprendizaje inductivo. El aprendizaje inductivo se puede definir como la identificación de patrones, regularidades, existentes en la evidencia. Es decir, se parte de casos particulares (ejemplos) y se obtienen casos generales (modelo) que generalizan la evidencia. Existen una gran cantidad de técnicas e hibridaciones de las mismas que se pueden aplicar a las tareas de la minería de datos.

En esta memoria se utilizarán tres técnicas que se engloban dentro del *soft-computing*, las redes neuronales, la computación evolutiva y la lógica difusa, que serán descritas posteriormente.

Evaluación y validación del modelo

Las medidas que se utilicen para evaluar el modelo dependen de la tarea de minería de datos a la que esté orientado el modelo. En el caso de la clasificación, la calidad del modelo se evalúa con respecto a su precisión

sobre el conjunto de tests. En el caso de reglas de asociación se mide la cobertura y confianza de las reglas obtenidas. Si la tarea es regresión el modelo habitualmente se evalúa considerando el error cuadrático medio del valor predicho respecto al real. Para la tarea de agrupamiento se suele medir la cohesión y separación de los grupos descubiertos. Además de las medidas comentadas existen otras algunas más subjetivas como el interés, la novedad, la simplicidad o la comprensibilidad [Hernández y otros, 2004].

La validación de un modelo de minería de datos es el proceso que pretende obtener una estimación adecuada del rendimiento del modelo con nuevos datos. Los datos disponibles, normalmente un conjunto reducido de muestras, se han de utilizar tanto para entrenar modelo como para probar la validez de éste. Para ello, los datos se suelen dividir en dos conjuntos: el conjunto de entrenamiento (*training set*) y el conjunto de test o prueba (*test set*). El primero de los conjuntos se utiliza para entrenar el modelo y hacer que éste aprenda y el segundo conjunto se utiliza para probar si el modelo es válido; con la división de los datos se consigue que la validación de la precisión del modelo sea una medida independiente de los datos que se tengan. Hay ocasiones en las que los datos se dividen en tres grupos: entrenamiento, validación (*validation set*) y test. El conjunto de validación se utiliza para ir refinando el modelo o elegir entre diferentes modelos antes de la obtención del modelo final en el algoritmo [Freitas, 2002; Hernández y otros, 2004].

Para realizar la validación del modelo aprendido se pueden utilizar alguno de los métodos siguientes [Tan y otros, 2006]:

- Resustitución: se utilizan todos los datos para entrenamiento y luego

todos para test. Se obtienen estimaciones demasiado optimistas.

- Partición (*Holdout*): el conjunto de muestras total se divide en dos conjuntos mutuamente exclusivos, conjuntos de entrenamiento y test (aproximadamente el conjunto de entrenamiento con $\frac{2}{3}$ de los datos y el de test con $\frac{1}{3}$ o $\frac{1}{2}$ y $\frac{1}{2}$). En este caso el modelo aprendido presenta un comportamiento peor que cuando aprende a partir de todas las muestras y además depende de la partición realizada. Este método se utilizará cuando el conjunto total de datos sea del orden de millares o superior.

- Submuestreo aleatorio (*random subsampling*): se realizan sucesivas particiones aleatorias del conjunto original y se aplica sucesivamente el método de partición. La estimación del error se calculará como la media de la estimación hecha para cada partición. En este caso hay una menor dependencia de la partición, aunque sigue sin utilizar todos los datos que son posibles para el entrenamiento y además no se controla el número de veces que un ejemplo es utilizado para entrenamiento y test (ciertos ejemplos pueden ser más utilizados para entrenamiento que otros).

- *Leaving-one-out*: Si hay n muestras, se realizan n particiones dejando en cada una de ellas 1 muestra para test y utilizando las $n-1$ restantes para entrenamiento. La estimación del error se calcula como la media de las n estimaciones y supone una cota superior de las probabilidades del error al igual que con *Holdout*.

- k-validación cruzada (*k-fold cross validation*): En este método los datos se dividen aleatoriamente en k particiones mutuamente exclusivas, k

es un parámetro definido por el usuario. Cada partición tiene el mismo número de instancias, $1/k$ del conjunto original. En la i-ésima ejecución la partición i es usada como conjunto de test y las $k-1$ restantes son mezcladas temporalmente y usadas como conjunto de entrenamiento. El proceso se repite k veces de forma que cada partición sea una vez conjunto de test. El error se calcula como la media de los errores ocurridos en cada una de las ejecuciones. Un valor de k habitual es 10 o 5.

- *Bootstraping*: en los método vistos se supone que los ejemplos en los conjuntos son muestras sin reemplazamiento, por lo que no hay ejemplos duplicados en los conjuntos de entrenamiento y test. En este método las muestras se seleccionan con reemplazamiento. Se escogen k conjuntos de tamaño n con reemplazamiento, con cada una de estas muestras se aprende el modelo que se prueba con los ejemplos no seleccionados.

En [Kohavi, 1995] se pueden encontrar comparaciones entre distintos métodos de validación.

1.2. Soft-computing

En el proceso de extracción de conocimiento en bases de datos, y en concreto en la fase de minería de datos, diversas tareas o problemas se pueden enfocar y resolver como problemas de optimización y búsqueda. Las técnicas tradicionales de optimización, conocidas como técnicas *hard-computing*, utilizan procedimientos determinísticos para alcanzar una solución óptima.

Dichas técnicas presentan limitaciones tales como que la convergencia a una solución depende de la solución inicial, que pueden quedar atrapados en óptimos locales, falta de eficiencia en problemas con espacios discretos o dificultad para el uso eficiente en máquinas paralelas. Frente a estos métodos clásicos aparecen nuevas tecnologías conocidas como *soft-computing* cuya característica principal es la tolerancia a la imprecisión y la incertidumbre, lo que les confiere una gran capacidad de adaptación que hace posible que se puedan aplicar a la resolución de problemas en entornos cambiantes y lo hagan de forma robusta y con bajo coste.

Las principales técnicas *soft-computing* son la lógica difusa, las redes neuronales, la computación evolutiva y el razonamiento probabilístico [Zadeh, 1994]. A continuación se explican tres de ellas que son las que se utilizan en esta memoria.

1.3. Computación evolutiva

Los Algoritmos Evolutivos (AEs) son algoritmos estocásticos de búsqueda inspirados en el proceso de evolución de Darwin [Darwin, 1859]. Estos algoritmos son aplicados a la minería de datos debido a su robustez y a que sus técnicas adaptativas de búsqueda permiten realizar una búsqueda global en todo el espacio de soluciones.

Un AE es esencialmente un algoritmo inspirado en los principios de selección y genética natural. En la naturaleza los individuos evolucionan de forma que se adaptan cada vez mejor al entorno. Los AEs trabajan con una población finita de individuos, cada uno de los cuales representa una solución candidata al problema propuesto y evolucionan para tratar de encontrar

cada vez mejores soluciones. Cada individuo se evalúa mediante una función que mide la calidad de la solución que representa. Dicha función asocia a cada individuo un valor de adaptación o *fitness*. Los individuos van evolucionando, a lo largo de generaciones (iteraciones), utilizando procedimientos basados en la selección natural (supervivencia y reproducción) y operadores basados en la genética natural (cruce y mutación), para obtener una nueva descendencia que reemplazará a los padres.

Los operadores de cruce y mutación son operadores estocásticos y a menudo se aplican con una probabilidad pre-establecida. Con el operador de cruce parte del material genético de dos individuos se intercambia para producir un nuevo individuo, mientras que con el operador de mutación se producen cambios aleatorios en parte del material genético del individuo. De esta forma se van obteniendo nuevas generaciones de individuos y el proceso continúa de forma iterativa hasta que se alcanza un criterio de parada, tal como que se logre un determinado número de generaciones o que la solución conseguida sea satisfactoria.

En los algoritmos evolutivos se distinguen los modelos *generacionales* frente a los *estacionarios*. En el primer tipo de modelo durante cada iteración se selecciona una población completa con nuevos individuos, es decir, la nueva población reemplaza directamente a la antigua. En el segundo tipo de modelo en cada iteración se escogen individuos a los que se les aplican los operadores genéticos, y el/los descendiente/s reemplazan a algunos individuos de la población inicial permaneciendo el resto sin modificaciones.

En el contexto de la computación evolutiva se suelen utilizar los términos *genotipo* y *fenotipo* al igual que en la genética natural. El genotipo de un individuo hace referencia a la representación interna de dicho individuo,

mientras que el fenotipo hace alusión a la solución que dicho individuo representa.

El uso de modelos de computación evolutiva presenta una serie de ventajas sobre la utilización de otros métodos:

- Los AEs se pueden aplicar aunque se tenga conocimiento incompleto del problema a resolver.
- Los AEs son menos susceptibles de quedar atrapados en óptimos locales. Esto es debido a que mantienen una solución de poblaciones alternativas y un balance entre la *explotación* de regiones del espacio de búsqueda, y la *exploración* de nuevas regiones. La exploración permite que el algoritmo se mueva por todo el espacio de soluciones posibles con lo que se evitan los óptimos locales, mientras que la explotación permite encontrar una posible solución óptima y moverse en el espacio circundante a ésta.

En los algoritmos evolutivos existen un conjunto de parámetros que rigen su funcionamiento, tales como pueden ser el tamaño de la población, el porcentaje de cruce, porcentaje de mutación, número de generaciones o tamaño de los individuos.

Existen principalmente cuatro modelos computacionales dentro de los AEs:

- Algoritmos Genéticos (AGs). Fueron introducidos por Hollan [Holland, 1975] y posteriormente estudiados por De Jong [Jong, 1975] y Goldberg [Goldberg, 1978]. Estos algoritmos enfatizan el cruce como el principal

operador para explorar el espacio de búsqueda y consideran la mutación como un operador menos importante, por lo que típicamente se aplica con una probabilidad baja. Al principio los AGs clásicos representaban a los individuos mediante vectores binarios aunque en la actualidad existen representaciones más elaboradas tales como vectores de valores reales.

- Programación Evolutiva (PE). Fue desarrollada por Fogel [Fogel, 1962], originariamente para evolucionar máquinas de estados finitos. En la actualidad es usada para evolucionar individuos que consisten en vectores de valores reales. Por regla general no se usa el operador de cruce y las mutaciones constituyen la única fuente cambios para las posibles soluciones. La PE se centra en la evolución a nivel de fenotipo.

- Estrategias de Evolución (EE). Desarrolladas por Rechenberg y Schwefel [Rechenberg, 1971; Schwefel, 1975]. Utilizan como representación típica de un individuo un vector de valores reales. Los primeros algoritmos de EE enfatizaban la mutación como el principal operador para la explotación, pero en la actualidad se utilizan ambos operadores, mutación y cruce. A menudo un individuo representa además de las variables reales del problema a resolver, los parámetros que controlan la distribución de la mutación, de forma que se tiene una autoadaptación de los parámetros de mutación. Lo más usual es que las mutaciones sobre los individuos sean pequeñas y sigan una distribución normal.

- Programación Genética (PG). Propuesto por Koza [Koza, 1992] [Koza, 1994]. A menudo se considera una variación de los AGs más que como

otro paradigma. En la PG los individuos no sólo consisten en los datos o parámetros de las funciones sino que las propias funciones (u operadores) están representadas en éstos.

Los pseudocódigos básicos de los AEs son bastante similares. En la figuras 1.3 se muestra el pseudocódigo representativo de los AG y PG, mientras en la figura y 1.4 se muestra el pseudocódigo que representa la EE y la PE. La diferencia principal es que en AG y PG la selección de los mejores individuos se hacer normalmente antes de la aplicación de los operadores genéticos que crean la descendencia, así los operadores son aplicados a los mejores individuos. En cambio EE y PE normalmente se aplican los operadores primero, para crear la descendencia y entonces se seleccionan los mejores individuos [Deb, 2000; Freitas, 2002].

```
Creación de la población inicial
Calculo del fitness de los individuos
Repetir
     Selección de los individuos
     Aplicación de los operadores genéticos
       a los individuos seleccionados y
       creación de la descendencia
     Cálculo del fitness de la descendencia
     Cambio de la población actual
Hasta (condición de parada)
```

Figura 1.3: *Pseudocódigo básico para AG y PG*

La caracterización anterior de los AEs en diferentes paradigmas debe considerarse sólo como comentarios generales y no como divisiones muy específicas. Dentro de cada uno de esos paradigmas hay muchos algoritmos,

```
Creación de la población inicial
Calculo del fitness de los individuos
Repetir
     Aplicación de los operadores genéticos
       a los individuos y creación de la descendencia
     Cálculo del fitness de la descendencia
     Selección de los individuos
     Cambio de la población actual
Hasta (condición de parada)
```

Figura 1.4: *Pseudocódigo básico para PE y EE*

con diferentes características. La tendencia actual es la unificación en el campo de los AEs, de tal forma que la existencia de clases diferenciadas tiende a desaparecer.

1.3.1. Componentes de un algoritmo evolutivo

Para definir un AE particular se han de especificar una serie de componentes, procedimientos u operadores. Los componentes más importantes son [Eiben y Smith, 2003]:

- Representación de los individuos
- Función de adaptación (*fitness*)
- Población
- Mecanismo de selección de padres
- Operadores (mutación y cruce)

- Estrategia de reemplazamiento para la nueva población

La elección del tipo de cada uno de los componentes anteriores es lo que diferencia unos paradigmas de otros dentro de la computación evolutiva.

Representación de los individuos. El primer paso que se ha de realizar es establecer un enlace entre el contexto original del problema y el espacio de solución del problema donde se va a realizar la evolución. Es decir, se ha de encontrar una representación que permita hacer traslaciones entre el fenotipo (solución) y el genotipo (representación de dicha solución).

En la computación evolutiva se suelen utilizar varios sinónimos para referenciar los elementos de estos dos espacios. En el contexto original del problema se utilizan los nombres individuo, solución candidata, fenotipo, para designar un punto del espacio de las posibles soluciones. En el espacio del AE, genotipo, cromosoma y de nuevo individuo son los nombres usados para los puntos del espacio donde se desarrolla la búsqueda evolutiva.

La palabra representación puede usarse de dos formas diferentes. Bien para indicar la relación entre el fenotipo y el genotipo (en este sentido es sinónimo de codificación), o bien, para indicar la estructura de datos utilizada para el genotipo.

El esquema de representación elegido para los individuos de una población dependerá del problema que se trate e influirá en los operadores que más tarde se tengan que diseñar.

Función de adaptación. El papel de esta función es evaluar cómo están

de adaptados los individuos al entorno. Dicha función asigna una medida de calidad a los individuos.

La función de adaptación es comúnmente llamada función *fitness* dentro del campo de los AEs.

Población. El papel de la población es mantener un conjunto de posibles soluciones. La población forma la unidad de evolución y es ella la que se va adaptando. Dada una representación, definir una población puede ser tan simple como especificar el número de individuos que la forman (tamaño de la población). En algunos AE más sofisticados la población tiene una estructura más compleja en la que también se detallan relaciones de vecindad o medidas de distancia.

En los AEs es interesante mantener lo que se conoce como *diversidad* de la población, que es una medida del número de soluciones diferentes presentes en ésta. La falta de diversidad hace que los individuos de la población sean parecidos, de forma que la búsqueda se puede estancar. Cuando este comportamiento ocurre antes de encontrar una solución óptima se dice que el AE sufre de *convergencia prematura*. Existen diferentes técnicas que tratan de mantener diversidad, como las técnicas de nichos (*niching*) [Beasly y otros, 1993], *sharing* [Goldberg y Richardson, 1987], *crowding* [Mahfoud, 1992] o restricciones al cruzamiento [Eshelman y Schaffer, 1991] entre otras.

Mecanismo de selección de padres. El objetivo de la selección es permitir que los mejores individuos de la población sean los padres de la siguiente generación. Junto con el mecanismo de selección de supervivientes, la selección de padres es la responsable de conseguir

mejoras en la calidad de los individuos. En la computación evolutiva la selección es típicamente probabilística, de forma que los individuos con una calidad alta tienen más probabilidad de ser elegidos como padres que los individuos con baja calidad. No obstante, se debe dar una oportunidad a los individuos menos buenos para impedir que se caiga en óptimos locales. Esta idea nos define lo que se conoce como *presión selectiva* que determina en qué grado la reproducción está dirigida por los mejores individuos. Si la presión ejercida es excesiva puede derivar en convergencia prematura.

Existen diferentes mecanismos de selección entre los cuales se encuentran la selección proporcional, por ranking y por torneo, como las más utilizadas:

- Selección proporcional: a cada individuo se le asocia una probabilidad de selección directamente proporcional a su *fitness*. De esta forma los individuos con mejor *fitness* tienen mayor probabilidad de ser seleccionados.
- Ranking: se ordenan los individuos de acuerdo con su *fitness*, de forma que al mejor individuo se le asigna el valor 1. Para realizar la selección se considera que el valor del *fitness* de los individuos es el valor que se les ha asociado en el ranking y se aplica el mecanismo de selección proporcional.
- Torneo: se eligen subgrupos de k individuos con reemplazamiento y se elige al mejor de cada subrupo. El valor k se denomina tamaño del torneo y cuanto mayor es su valor mayor es la presión selectiva.

Operadores de mutación y cruce. Estos operadores son conocidos como operadores de variación y su objetivo es crear nuevos individuos a partir de otros ya existentes en la población. En el espacio de las soluciones (fenotipos) consiste en obtener nuevas soluciones candidatas.

Mutación: Este operador se aplica a un solo individuo y lo modifica con pequeñas mutaciones de forma que se obtiene un hijo. Es un operador estocástico de forma que el hijo que se obtiene depende de una serie de elementos aleatorios. El papel de este operador es diferente en los distintos paradigmas de la CE; por ejemplo en la programación genética no se suele utilizar, en algoritmos genéticos es un operador de segundo plano que renueva el material genético de los individuos, y en programación evolutiva es el único operador de variación que se utiliza.

Cruce: Este operador también es conocido como recombinación, se aplica a dos individuos (padres) de la población y mezclando su información genética se obtienen uno o dos hijos. Al igual que el operador de mutación, el operador de cruce también es estocástico, ya que la elección de las zonas de los padres que van a ser cruzadas y cómo dichas zonas se van a cruzar dependen de decisiones aleatorias. De nuevo el papel de este operador es diferente en los distintos paradigmas de la CE. En la programación genética suele ser el único operador de variación, en algoritmos genéticos suele ser el principal operador y en programación evolutiva no se usa.

El principio de la recombinación es simple, si se cruzan dos individuos con características deseables aunque diferentes, se puede obtener un hijo que combine ambas características deseables.

Los algoritmos evolutivos crean hijos mediante cruces aleatorios y puede ocurrir que algunas combinaciones obtenidas (hijos) no sean del todo deseables, pero se espera que otros hayan mejorado las características de los padres.

Estrategia de reemplazamiento. La estrategia de reemplazamiento, también conocida como selección de supervivientes o selección medioambiental, consiste en seleccionar a los individuos que pasarán a la siguiente generación. La decisión se basa en la función de adaptación, de forma que se seleccionan los individuos mejor adaptados.

Al igual que ocurre en la selección de padres para aplicar los operadores evolutivos, hay que dar también oportunidad a que individuos menos buenos puedan pasar a la siguiente generación. Si siempre se reemplaza a los peores individuos, se puede llegar a una convergencia prematura.

En algunas ocasiones se dice que el algoritmo incorpora *elitismo*, que consiste en que algunos individuos con un buen *fitness* (la élite) se mantienen intactos para la siguiente generación.

Inicialización. La primera población, en la mayoría de las aplicaciones evolutivas, se constituye a partir de individuos generados aleatoriamente. También pueden utilizarse heurísticas basadas en el problema específico que se esté tratando, de forma que se consiga desde este primer paso una población con buen *fitness*. Esta última opción, dependiendo del porcentaje de población inicial obtenido mediante heurística, puede provocar convergencia prematura.

Condición de parada. No se puede establecer como condición de parada el que se alcance una solución óptima, dado que los AEs son

estocásticos y, por tanto no pueden garantizar que se alcance dicha solución. Se necesita, por tanto, extender dicha condición con otras que nos garanticen que el algoritmo parará. Las opciones más utilizadas para este propósito son las siguientes:

- Se impone un límite en el tiempo de ejecución del algoritmo.
- Se limita el número total de generaciones.
- Que para un periodo de tiempo dado (por ejemplo, para un número de generaciones) la mejora del *fitness* permanece por debajo de un umbral.
- Que la diversidad de la población caiga por debajo de cierto umbral.
- Que se consiga una solución satisfactoria a un nivel especificado.

1.3.2. Algoritmos co-evolutivos y cooperativos-competitivos

En estos tipos de algoritmos el problema se va a dividir en un conjunto de subtareas y para solucionarlo evolucionan componentes que realizan dichas subtareas y que interaccionan entre sí.

Estos métodos intentan solucionar problemas que aparecen en la computación evolutiva tradicional. Dichos problemas son de tres tipos: aquellos en los que se requieren múltiples y distintas soluciones (multimodales), aquellos en los que la solución está formada por subcomponentes especializados, y por último aquellos que se pueden descomponer en subtareas más simples.

Los algoritmos evolutivos clásicos tiene dificultades a la hora de abordar problemas de los tipos anteriores, debido principalmente a dos razones:

- La población de individuos tiene una fuerte tendencia a converger. Esta convergencia hace que sólo se estudien determinadas soluciones, lo cual no interesa en los problemas multimodales. Por otro lado imposibilita la preservación de los diferentes componentes, dado que cualquier individuo menos fuerte es eliminado, lo cual hace que no se puedan aplicar en problemas con soluciones formadas por componentes.

- Los individuos de los algoritmos evolutivos tradicionales representan soluciones completas que se evalúan de forma aislada. Por tanto, necesita aumentar su aplicación para permitir mantener conductas diferentes que puedan coadaptarse.

Con los algoritmos co-evolutivos y los cooperativos-competitivos se introduce en el ámbito de la programación evolutiva soluciones que están formadas por subcomponentes que evolucionan e interactúan entre sí y por tanto son buenos para tratar problemas de los tipos expuestos anteriormente.

Los aspectos que hay que abordar en estos tipos de algoritmos son:

- Determinar un número de subcomponentes adecuado así como el papel que cada uno debe desempeñar. A esta tarea se le conoce como descomposición del problema.

- Interdependencia entre subcomponentes. Existen problemas en los que los subcomponentes son independientes de forma que cada uno evoluciona en su espacio, mientras que en otro tipo de problemas existen dependencias entre subcomponentes y por tanto el hecho de cambiar uno de estos subcomponentes puede afectar al espacio de otros.

- Asignación de crédito. En este entorno en el que la solución depende del comportamiento de varios componentes al *fitness* se le conoce como *asignación de crédito*. Se ha de evaluar la contribución de cada individuo (que representa una solución parcial) a la solución total. Esta medida dependerá del número de subcomponentes, de las dependencias entre éstos y del problema que se trate.

- Diversidad de la población. En la evolución clásica los individuos representan una solución para un determinado problema y la diversidad se va manteniendo durante un tiempo de la ejecución para garantizar la exploración del espacio de búsqueda. Pero cuando la solución se alcanza mediante una colección de componentes, la diversidad hay que mantenerla hasta el final, dado que todos los componentes son necesarios para dicha solución. Para mantener dicha diversidad se pueden usar técnicas como *la compartición del valor de adaptación (fitness sharing)* [Goldberg y Richardson, 1987] u otro tipo de *técnicas de nichos* [Beasly y otros, 1993].

Algoritmos co-evolutivos

En los algoritmos co-evolutivos se tienen dos o más poblaciones, cada una de las cuales representa una parte del problema, que evolucionan e interactúan.

En la vida real existe la co-evolución dado que el entorno es compartido por multitud de especies que evolucionan. La interacción entre especies también aparece por ejemplo en relaciones tales como la simbiosis, el parasitismo, o las interacciones entre presa y depredador. Este tipo de

interacciones observadas en la naturaleza se tratan de explotar en los algoritmos co-evolutivos artificiales.

Los métodos basado en co-evolución se pueden clasificar en competitivos o cooperativos.

En un algoritmo co-evolutivo competitivo el *fitness* de un individuo se basa en la competición directa con los individuos de las otras especies y cada especie evoluciona en su propia población. Un incremento del *fitness* en una especie implica el decremento en las otras especies. Esta presión evolutiva tiende a que se produzcan estrategias en la evolución de la población que mantengan sus posibilidades de supervivencia. Esta lucha entre especies intenta incrementar las capacidades de éstas hasta que se consigue alcanzar un óptimo. En [Hillis, 1991; Rosin y Belew, 1997] se pueden encontrar ejemplos de este tipo de algoritmos.

En la co-evolución cooperativa las diferentes especies van evolucionando independientemente y entre todas intentan resolver un problema. En este caso el *fitness* de un individuo depende de su capacidad para colaborar con los individuos de otras especies y la presión evolutiva favorece las estrategias de colaboración entre las especies. Este tipo de co-evolución aparece en trabajos como [Husbands y Mill, 1991; Giordana y otros, 1994; Paredis, 1995; Moriarty y Miikkulainen, 1997; Potter y De Jong, 2000].

Algoritmos cooperativos-competitivos

En el caso de los algoritmos cooperativos-competitivos, existe una única población pero los individuos no componen una solución completa sino que representan una solución parcial. Para formar la solución completa se

necesita un grupo de individuos o todos los de la población. Éstos individuos cooperan en aras de alcanzar la solución óptima, pero también compiten por su supervivencia dado que los mejores adaptados son los que permanecerán.

En este caso el *fitness* promueve la competición entre componentes que realizan subtareas similares y la cooperación entre componentes que realizan subtareas diferentes ya que juntos consiguen resolver el problema total.

Ejemplos de algoritmos cooperativos-competitivos se pueden encontrar en [Whitehead y Choate, 1996; Topchy y otros, 1997; Rivera y otros, 2007].

1.4. Lógica difusa

Zadeh, en 1965 [Zadeh, 1965] introduce los *conjuntos difusos* y la *lógica difusa*, un nuevo modelo de inferencia capaz de manejar conocimiento impreciso y cuantitativo más cercano al razonamiento humano. Desde entonces son muchas las aplicaciones de la lógica difusa a muchas áreas de investigación, dado que proporciona un medio efectivo de manejar la inexactitud presente en los problemas del mundo real. Las características más atractivas de la lógica difusa son su flexibilidad, su tolerancia a la imprecisión, su capacidad para modelar problemas no lineales y su forma de expresión similar al lenguaje natural. Como sinónimo de difuso se suelen emplear otros adjetivos tales como borroso, vago o impreciso.

Como se ha mencionado antes, la lógica difusa permite tratar información imprecisa contenida en frases tales como “hace mucho calor”, “esa persona es bastante alta”, “el tren va lento”, etc. Las frases anteriores se caracterizan porque los adjetivos utilizados en ellas no tienen una medida exacta que

los cuantifique y a veces sus valores presentan cierta subjetividad. Esta información imprecisa se puede tratar en términos de conjuntos difusos que se combinan en reglas que permiten definir acciones tales como "si hace mucho calor enciende el ventilador", "si el automóvil va lento pisa el acelerador", etc. De esta manera un sistema de control basado en lógica difusa toma variables de entrada, definidas en términos de conjuntos difusos y produce valores de salida por medio de reglas donde se combinan las entradas.

La lógica clásica o razonamiento preciso trabaja con variables que toman un valor numérico exacto ("la altura es de 1 metro"), mientras que ahora en la lógica difusa las variables toman como valor un rango o *etiqueta lingüística* ("la altura es baja"). Una variable difusa puede tomar cualquier valor dentro de un conjunto finito de etiquetas lingüísticas, que se describen mediante *funciones de pertenencia difusas*. Entre las variables difusas también se pueden establecer conexiones lógicas, formar reglas y combinar dichas reglas, al igual que se hace con las variables utilizadas en la lógica clásica.

A continuación se describen formalmente los conceptos mencionados anteriormente y que son necesarios para implementar un sistema que permita el razonamiento utilizando lógica difusa. A este tipo de sistema se le conoce como *Sistema Basado en Reglas Difusas* (SBRD). En particular se definen conceptos implicados en un SBRD como la definición de conjuntos difusos y operaciones; se definirá el tratamiento de variables como variables lingüísticas (muy habitual en SBRDs) y alguna de las formas en las que habitualmente se realiza inferencia difusa. La sección finaliza con la descripción de los componentes del tipo de sistema difuso que se utilizará en esta memoria, el SBRD descriptivo.

1.4.1. Conjuntos nítidos y conjuntos difusos

En la lógica clásica, un conjunto *nítido* (por contraposición a difuso), en un universo de discurso U, se puede definir de forma explícita mediante la enumeración de todos sus elementos, o de forma implícita estableciendo la condición que tienen que cumplir sus elementos, por ejemplo $N = \{x \mid x > 5\}$. Para dicho conjunto se puede definir una *función de pertenencia* (también conocida como función característica), $\mu_N(X)$, definida según la ecuación 1.1.

$$\mu_N(x) = \begin{cases} 1 & si \quad x \in N \\ 0 & si \quad x \notin N \end{cases} \tag{1.1}$$

Dado un elemento cualquiera, a través de la función de pertenencia se sabe si el elemento pertenece o no al conjunto anterior. Es decir, en la teoría clásica de conjuntos sólo se contempla la pertenencia o no (0 ó 1) de un elemento al conjunto, ahora con la teoría de conjuntos difusos dado un elemento se asocia a él un grado de pertenencia, es decir un valor en el intervalo [0,1], de tal manera que hay un transición gradual desde no pertenecer al conjunto hasta pertenecer con un grado 1. Desde un punto de vista matemático el conjunto es equivalente a su función de pertenencia, dado que conocer dicha función es igual a conocer el conjunto.

Un conjunto difuso D, en un universo de discurso U, se puede definir a través de su función de pertenencia $\mu_D(X)$, que toma valores en el intervalo [0,1] y se puede representar como un conjunto de pares ordenados de un elemento x y su valor de pertenencia al conjunto (ecuación 1.2).

$$D = \{(x, \mu_D(x) \mid x \in U)\} \tag{1.2}$$

Muchos de los conceptos en la teoría clásica de conjuntos se puede extender a los conjuntos difusos, aunque otros son exclusivos de éstos últimos. Algunos de estos conceptos más utilizados son:

- El soporte de un conjunto difuso D en el universo U se define como $sop(x) = \{x \in U \mid \mu_D(x) > 0\}$.
- Dos conjuntos difusos D_1 y D_2 son iguales si y sólo si $\mu_{D_1}(x) = \mu_{D_2}(x) \quad \forall x \in U$

Tal y como se ha mencionado anteriormente, la función de pertenencia devuelve un valor que refleja el grado de pertenencia de un elemento a un determinado conjunto difuso. Existen distintos tipos de función de pertenencia dependiendo de la forma que ésta adopte, las restricciones que tiene que cumplir son la continuidad y que sus valores tienen que estar comprendidos en el intervalo [0,1]. En [Klir y Yuan, 1995] se hace una descripción detallada sobre la teoría de los conjuntos difusos. Algunas de las funciones de pertenencia más utilizadas (figuras 1.5, 1.6 y 1.7) son las siguientes:

Triangular: se define mediante tres parámetros que indican sus límites inferior a y superior b y su valor modal c, tal que $a < c < b$.

Trapezoidal: se define mediante su límite inferior a, superior d y los límites de su soporte inferior b y superior c.

Gaussiana: definida por su valor medio o cento c y su radio σ.

Una variable puede tener asociadas varias funciones de pertenencia, a mayor número de funciones asociadas se tiene mayor resolución pero

$$\mu_N(x) = \begin{cases} 0 & si \quad x \leq a \\ \dfrac{x-a}{c-a} & si \quad x \in (a,c] \\ \dfrac{b-x}{b-c} & si \quad x \in (c,b) \\ 0 & si \quad x \geq b \end{cases}$$

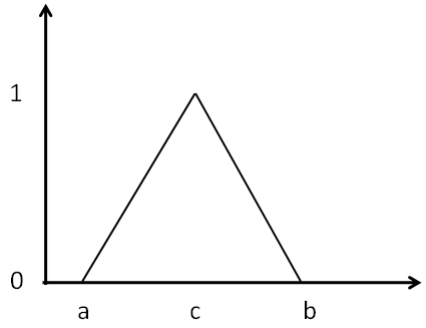


Figura 1.5: *Función de pertenencia triangular*

$$\mu_N(x) = \begin{cases} 0 & si \quad x \leq a, x \geq d \\ \dfrac{x-a}{b-a} & si \quad x \in (a,b] \\ 1 & si \quad x \in (b,c) \\ \dfrac{d-x}{d-c} & si \quad x \in [c,d) \end{cases}$$

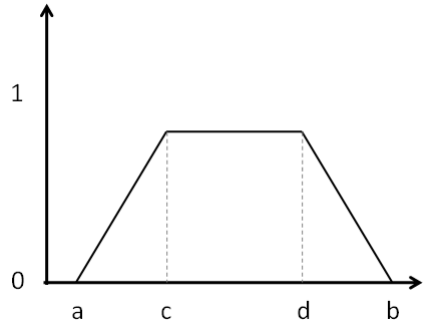


Figura 1.6: *Función de pertenencia trapezoidal*

$$\mu_N(x) = e^{-(\frac{x-c}{\sigma})^2}$$

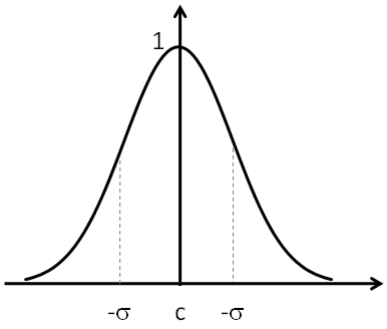


Figura 1.7: *Función de pertenencia gaussiana*

también mayor complejidad. Dichas funciones pueden estar solapadas, este solapamiento pone de manifiesto el hecho de que una variable puede

pertenecer con diferentes grados a varios conjuntos difusos a la vez.

1.4.2. Operaciones entre conjuntos difusos

Dados dos conjuntos difusos D_1 y D_2 en U con funciones de pertenencia $\mu_{D_1}(x)$ y $\mu_{D_2}(x)$ respectivamente, se definen las siguientes operaciones básicas:

- Unión de D_1 y D_2: se define como un nuevo conjunto difuso $D_1 \cup D_2$ en U para el que la función de pertenencia, $\forall x \in U$, se define como

$$\mu_{D_1 \cup D_2}(x) = max\{\mu_{D_1}(x), \mu_{D_2}(x)\} \tag{1.3}$$

  Es el menor conjunto difuso que contenga a los dos conjuntos de partida. En general, la unión se puede denotar como:

$$\mu_{D_1 \cup D_2}(x) = \mu_{D_1}(x) \oplus \mu_{D_2}(x) \tag{1.4}$$

  donde a $\oplus$ se le denomina operador *t-conorma* (conorma triangular). Existen diferentes ejemplos de t-conormas además del máximo, algunas de ellas se muestran en la tabla 1.1.

- Intersección de D_1 y D_2: se define como un nuevo conjunto difuso $D_1 \cap D_2$ en U para el que la función de pertenencia, $\forall x \in U$, se define como

$$\mu_{D_1 \cap D_2}(x) = min\{\mu_{D_1}(x), \mu_{D_2}(x)\} \tag{1.5}$$

Esta operación obtiene el mayor conjunto difuso que es contenido por los conjuntos de partida. Se puede generalizar y expresarla como:

$$\mu_{D_1 \cap D_2}(x) = \mu_{D_1}(x) \otimes \mu_{D_2}(x) \tag{1.6}$$

donde $\otimes$ expresa una *t-norma* (norma triangular). Otras t-normas, además de la del mínimo se pueden observar en la tabla 1.1.

- Complemento de un conjunto D: es un nuevo conjunto difuso $\bar{D}$ en U para el que la función de pertenencia, $\forall x \in U$, se define como

$$\mu_{\bar{D}}(x) = 1 - \mu_D(x) \tag{1.7}$$

Las tres operaciones vistas cumplen, al igual que ocurre con los conjuntos nítidos, las propiedades distributiva, asociativa y conmutativa, así como las leyes de De Morgan. Sin embargo no se cumplen el principio de contradicción ($D \cup \bar{D} = U$) y el principio de exclusión ($D \cap \bar{D} = \emptyset$).

1.4.3. Producto cartesiano, relaciones y composición difusas

Dados los conjuntos difusos D_1 y D_2, en los universos U_1 y U_2 respectivamente, se define el producto cartesiano de ambos como un nuevo conjunto difuso $D_1 \times D_2$ en el espacio $U_1 \times U_2$ con la función de pertenencia siguiente:

$$\mu_{D_1 \times D_2}(x_1, x_2) = min\{\mu_{D_1}(x_1), \mu_{D_2}(x_2)\} \tag{1.8}$$

Una relación difusa representa el grado de presencia o ausencia de asociación, interacción o interconexión entre dos o más conjuntos difusos, por

T-conormas	
Suma algebraica	$x \oplus y = x + y$
Suma limitada	$x \oplus y = min(1, x + y)$
Suma drástica	$x \oplus y = \begin{cases} x & si \quad y = 0 \\ y & si \quad x = 0 \\ 1 & si \quad x, y = 0 \end{cases}$
T-normas	
Producto algebraico	$x \otimes y = x \cdot y$
Producto limitado	$x \otimes y = max(0, x + y - 1)$
Producto drástico	$x \otimes y = \begin{cases} x & si \quad y = 1 \\ y & si \quad x = 1 \\ 0 & si \quad x, y < 1 \end{cases}$

Tabla 1.1: *Distintas T-conormas y T-normas*

ejemplo “x es mayor que y”, “z es cercano a h”, etc. Este tipo de relaciones son muy importantes en un SBRD.

Supongamos U y V dos universos de discurso, la relación difusa $R(U, V)$ es un conjunto difuso en el espacio producto $U \times V$, que se caracteriza por la función de pertenencia $\mu_R(x, y)$ donde $x \in U$ e $y \in V$, es decir:

$$R(U, V) = \left\{ \big((x, y), \mu_R(x, y)\big) \mid (x, y) \in U \times V \right\} \tag{1.9}$$

Las relaciones difusas constituyen conjuntos difusos en el espacio producto, por lo que se pueden aplicar a ellas las operaciones vistas para los conjuntos difusos, unión, intersección y complemento. De esta forma, dadas dos relaciones difusas $R(x, y)$ y $S(x, y)$ en el mismo espacio producto $U \times V$, se definen la unión y la intersección entre ellas (composiciones de dos relaciones) como:

$$\mu_{R\cup S}(x,y) = \mu_R(x,y) \oplus \mu_S(x,y) \tag{1.10}$$

$$\mu_{R\cap S}(x,y) = \mu_R(x,y) \otimes \mu_S(x,y) \tag{1.11}$$

También se pueden definir composiciones de relaciones de diferentes espacios productos que comparten un conjunto común, por ejemplo $R(U,V)$ y $S(V,W)$ ("x es menor que y e y es cercano de z"). Asociadas a la relaciones difusas R y S están sus funciones de pertenencia $\mu_R(x,y)$ y $\mu_S(y,z)$. Cuando R y S se definen en universos de discurso discretos, entonces su composición difusa, $R \circ S$, se define como una relación difusa en $U \times W$ cuya función de pertenencia es:

$$\mu_{R\circ S}(x,z) = \sup_{y\in V}\big[\mu_R(x,y) \otimes \mu_S(y,z)\big] \tag{1.12}$$

donde sup es el operador máximo y $\otimes$ es una t-norma. Como t-normas se suelen utilizar el mínimo y el producto. La forma anterior de expresar la función de pertenencia se denomina composición *sup-star* de R y S.

En la figura 1.8 se muestra un diagrama interpretativo para la composición sup-star, que de una forma simple permite relaciones difusas más complicadas. Si sólo se considera una relación difusa R como un conjunto difuso, entonces $\mu_R(x,y)$ se convierte en $\mu_R(x)$. Por ejemplo "si x es medianamente grande y z es menor que x". Entonces $V = U$ y la figura 1.8 se transforma en la figura 1.9, que representa un conjunto difuso que puede activar una relación difusa. Esta es una de las bases del funcionamiento del SBRD.

Si ocurre que $U = V$, entonces se satisface la ecuación 1.13, que es sólo

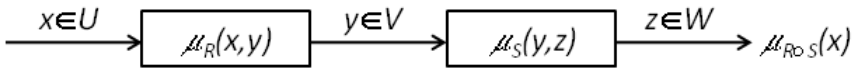


Figura 1.8: *Composición sup-star*

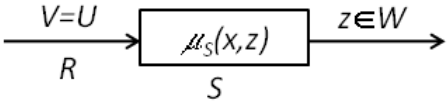


Figura 1.9: *Composición sup-star interpretando la primera relación como un conjunto difuso*

función de la variable de salida z, por lo que se puede simplificar la notación $\mu_{R\circ S}(x,z)$ a $\mu_{R\circ S}(z)$, así que cuando R es sólo un conjunto difuso, se tiene la ecuación 1.14.

$$\sup_{y\in V}\big[\mu_R(x,y)\otimes\mu_S(y,z)\big]=\sup_{x\in U}\big[\mu_R(x)\otimes\mu_S(x,z)\big] \tag{1.13}$$

$$\mu_{R\circ S}(z)=\sup_{x\in U}\big[\mu_R(x)\otimes\mu_S(x,z)\big] \tag{1.14}$$

1.4.4. Variables lingüísticas

Una variable lingüística se caracteriza por la tupla $\big(x,T(x),U,G,M\big)$, donde x es el nombre de la variable, $T(x)$ es el conjunto de términos o etiquetas de x, U es el universo de discurso, G es la regla sintáctica para generar los valores de x y M es una regla semántica para asociar cada valor con su significado. Por ejemplo si se considera la variable lingüística

temperatura, su conjunto de etiquetas podría ser

$$T(temperatura) = \{baja, media, alta\}$$

donde cada etiqueta es un conjunto difuso en un universo de discurso U. Por ejemplo se puede considerar como *baja* "una temperatura por debajo de 10°C", *media* "una temperatura en torno a los 20°C" y *alta* "temperatura superior a 30°C". Estas etiquetas se pueden caracterizar como conjuntos difusos cuyas funciones de pertenencia están representadas en la figura 1.10.

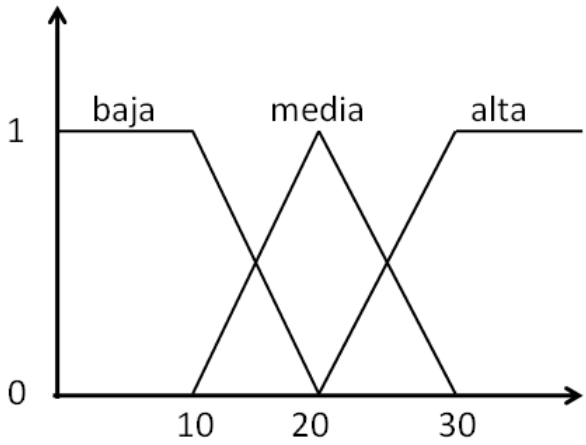


Figura 1.10: *Representación de las etiquetas lingüísticas de la variable temperatura*

1.4.5. Inferencia difusa

Se denominan *reglas difusas* (al igual que ocurre en la lógica clásica) al conjunto de proposiciones que modelan el problema que se quiere resolver. Una regla difusa simple tiene la forma:

SI x es A ENTONCES y es B

donde A y B son etiquetas lingüísticas (conjuntos difusos) definidas en universos de discursos U y V respectivamente. Se llama *antecedente* o *premisa* de la regla a la parte "x es A" y *consecuente* o *conclusión* a la parte "y es B". La regla se puede expresar como $A \rightarrow B$

Una regla expresa un tipo de relación entre los conjuntos A y B cuya función característica es $\mu_{A\rightarrow B}(x, y)$ y representa lo que se conoce como implicación lógica. La función $\mu_{A\rightarrow B}(x, y) \in [0, 1]$ y mide el grado de verdad de la relación de implicación entre x e y. Ejemplos de interpretaciones de esta función de pertenencia son:

$$\mu_{A\rightarrow B} = 1 - min\big[\mu_A(x), 1 - \mu_B(y)\big] \tag{1.15}$$

$$\mu_{A\rightarrow B} = max\big[1 - \mu_A(x), \mu_B(y)\big] \tag{1.16}$$

$$\mu_{A\rightarrow B} = 1 - \mu_A(x)\big(1 - \mu_B(y)\big) \tag{1.17}$$

Como se sabe, se puede establecer un isomorfismo entre la teoría de conjuntos, la lógica proposicional y el álgebra booleana que garantiza que cada teorema enunciado en una de ellas tiene un homólogo en las otras dos. La existencias de dichos isomorfismos permiten traducir las reglas difusas a relaciones entre conjuntos difusos y éstas a términos de operadores algebraicos con los que se puede trabajar. El *razonamiento aproximado* (o *razonamiento difuso*) es un procedimiento de inferencia usado para obtener conclusiones a partir de un conjunto de reglas difusas y una o más condiciones. Para dicha inferencia se usa una generalización del *Modus Ponens* clásico dando lugar a lo que se conoce como *Modus Ponens generalizado* y cuyo funcionamiento se muestra en la figura 1.11

premisa 1:	x es A^*
premisa 2:	SI x es A ENTONCES y es B
consecuencia:	y es B^*

Figura 1.11: *Modus Ponens generalizado*

La regla de inferencia del Modus Ponens generalizado está basada en la regla de composición de inferencia para el razonamiento aproximado, aportada por Zadeh en [Zadeh, 1973]. La diferencia con la regla de inferencia clásica estriba en los conjuntos difusos A^* y B^*, ya que el conjunto difuso A^* no es necesariamente igual a conjunto difuso antecedente A y lo mismo ocurre con B^* y B. Esto implica que en lógica difusa una regla se activa en función de la similaridad entre la primera premisa y el antecedente de la regla y el resultado es un consecuente que tendrá cierto grado de similaridad con el consecuente de la regla.

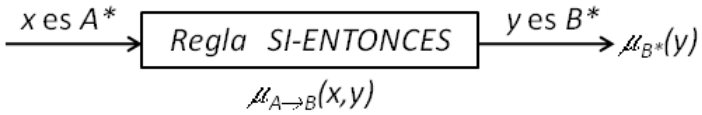


Figura 1.12: *Interpretación del Modus Ponens generalizado*

En la figura 1.12 se representa una interpretación para el Modus Ponens generalizado, como se puede observar coincide con la figura 1.9. De forma que se puede concluir que el Modus Ponens generalizado es una composición difusa en el que la primera relación difusa es el conjunto difuso A^*.

$$B^* = A^* \circ R = A^* \circ (A \rightarrow B) \tag{1.18}$$

Para obtener $\mu B^*(y)$ se puede usar la composición sup-star y aplicando transformaciones simbólicas en 1.14 se obtiene:

$$\mu_{B^*}(y) = \sup_{x \in A^*} \left[\mu_{A^*}(x) \otimes \mu_{A \to B}(x, y)\right] \tag{1.19}$$

Hay diversas formas de implementar esa implicación. Entre las más comunes destaca la de Mamdani [Mamdani, 1974]:

$$\mu_{A \to B}(x, y) = min\left[\mu_A(x), \mu_B(y)\right] \tag{1.20}$$

o la de Larsen [Larsen, 1980]

$$\mu_{A \to B}(x, y) = \mu_A(x)\mu_B(y) \tag{1.21}$$

Esto se puede expresar como una t-norma:

$$\mu_{A \to B}(x, y) = \mu_A(x) \otimes \mu_B(y) \tag{1.22}$$

Para trabajar de forma más general, la ecuación 1.19 se expresa de la siguiente forma:

$$\mu_{B^*}(y) = \underset{x \in A}{\oplus} \left[\mu_{A^*}(x) \otimes \mu_{A \to B}(x, y)\right] \tag{1.23}$$

Algunos ejemplos prácticos que se pueden presentar son los que se muestran a continuación.

Un solo antecedente y un solo consecuente. En este caso, la ecuación

1.23 se puede transformar en:

$$\mu_{B^*}(y) = \underset{x \in A}{\oplus} \left[\mu_{A^*}(x) \otimes \mu_A(x)\right] \otimes \mu_B(y) = w \otimes \mu_B(y) \qquad (1.24)$$

Esto implica que se busca primero el grado de solapamiento máximo, w, entre $\mu_{A^*}(x) \otimes \mu_A(x)$ (área sombreada en la parte antecedente de la figura 1.13), luego la función de pertenencia de B^* es igual a la función de pertenencia de B recortada por w (se muestra en la pare consecuente de la figura 1.13).

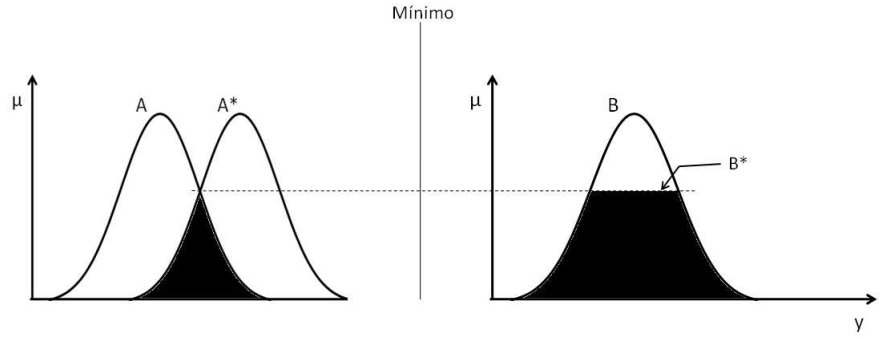


Figura 1.13: *Razonamiento con un antecedente y un consecuente*

Dos antecedentes y un consecuente. En este caso la regla es algo como "SI x es A Y y es B ENTONCES z es C". El problema a resolver en este caso, mediante el razonamiento aproximado, se expresa en la figura 1.14.

La regla difusa de la premisa 2 se puede expresar como $A \times B \rightarrow C$. Intuitivamente esta relación difusa se puede expresar con la siguiente función de pertenencia:

premisa 1:	x es A^* Y y es B^*
premisa 2:	SI x es A Y y es B ENTONCES z es C
consecuencia:	z es C^*

Figura 1.14: *Ejemplo de razonamiento con dos antecedentes y un consecuente*

$$\mu_{A\times B\to C}(x,y,z) = \mu_A(x) \otimes \mu_B(y) \otimes \mu_C(z) \tag{1.25}$$

y C^* se puede expresar como se indica en la ecuación 1.26

$$C^* = (A^* \times B^*) \circ (A \times B \to C) \tag{1.26}$$

y entonces la función de pertenencia de μ_{C^*} se puede expresar como se muestra en la ecuación 1.27:

$$\begin{aligned}
\mu_{C^*}(y) =& \underset{x,y}{\oplus} \left[\mu_{A^*}(x) \otimes \mu_{B^*}(y)\right] \otimes \left[\mu_A(x) \otimes \mu_B(y) \otimes \mu_C(z)\right] \\
=& \underset{x,y}{\oplus} \left[\mu_{A^*}(x) \otimes \mu_{B^*}(y) \otimes \mu_A(x) \otimes \mu_B(y)\right] \otimes \mu_C(z) \\
=& \left\{ \underset{x\in A}{\oplus} \left[\mu_{A^*}(x) \otimes \mu_A(x)\right] \right\} \otimes \left\{ \underset{y\in B}{\oplus} \left[\mu_{B^*}(y) \otimes \mu_B(y)\right] \right\} \otimes \mu_C(z) \\
=& \, w_1 \otimes w_2 \otimes \mu_C(z)
\end{aligned} \tag{1.27}$$

donde w_1 es el grado máximo solapamiento entre A^* y A, w_2 es el grado máximo solapamiento entre B^* y B y $w_1 \otimes w_2$ es grado de disparo de la regla para el que se escoge una t-norma mínimo. En la figura 1.15 se muestra una interpretación gráfica de este procedimiento, donde la función de pertenencia de C^* e igual a la función de pertenencia de C

recortada por el grado de disparo de la regla. Se puede generalizar a más de dos antecedentes de forma directa.

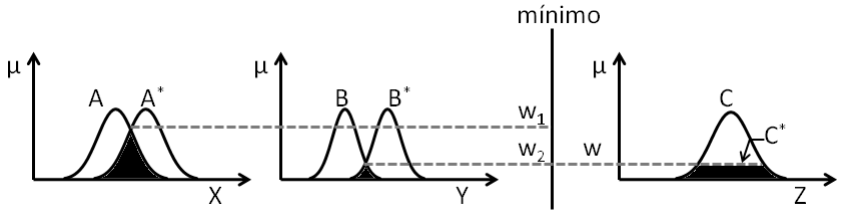


Figura 1.15: *Razonamiento con dos antecedentes y un consecuente*

Múltiples reglas con múltiples antecedentes. La interpretación de múltiples reglas se toma normalmente como la unión de relaciones difusas correspondientes a la reglas difusas. Por ejemplo dadas las premisas y reglas de la figura 1.16, para obtener C^* se realizan transformaciones aplicando la ley distributiva sobre el operador $\cup$, ecuación 1.28, donde C_1^* y C_2^* son los conjuntos difusos para la regla 1 y 2, respectivamente. En la figura 1.17 se muestra el razonamiento difuso para múltiples reglas con múltiples antecendentes, utilizando como t-conorma el máximo y como t-norma el mínimo.

`premisa 1:`	x `es` A^* `Y` y `es` B^*
`premisa 2:`	`SI` x `es` A_1 `Y` y `es` B_1 `ENTONCES` z `es` C_1
`premisa 3:`	`SI` x `es` A_2 `Y` y `es` B_2 `ENTONCES` z `es` C_2
`consecuencia:`	z `es` C^*

Figura 1.16: *Ejemplo de razonamiento con múltiples reglas y antecedentes*

$$
\begin{aligned}
C^* &= (A^* \times B^*) \circ \left[(A_1 \times B_1 \rightarrow C_1) \cup (A_2 \times B_2 \rightarrow C_2)\right] \\
&= \left[(A^* \times B^*) \circ (A_1 \times B_1 \rightarrow C_1)\right] \cup \left[(A^* \times B^*) \circ (A_2 \times B_2 \rightarrow C_2)\right] \\
&= C_1^* \cup C_2^*
\end{aligned}
\tag{1.28}
$$

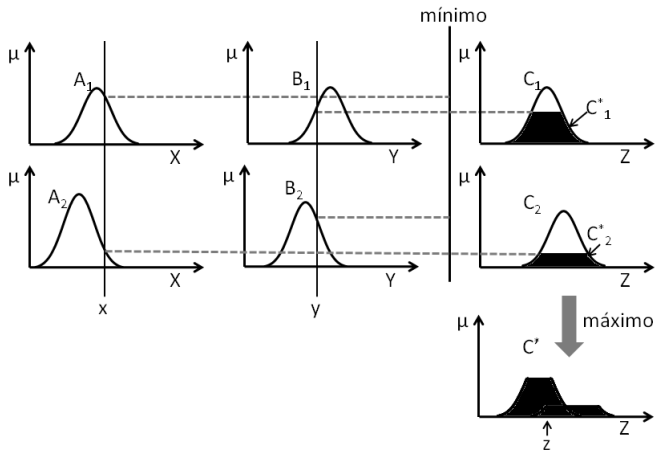


Figura 1.17: *Razonamiento difuso (máximo-mínimo) para múltiples reglas con múltiples antecedentes*

El tipo de razonamiento explicado es el razonamiento de Mamdani [Mamdani y Assilian, 1975]. En este modelo de razonamiento se puede utilizar la operación producto en la implicación en vez del mínimo (se muestra en la figura 1.18).

Existen otros modelos de razonamiento como el de Sugeno [Sugeno y Kang, 1988], o el de Tsukamoto [Tsukamoto, 1979] donde la salida se expresa

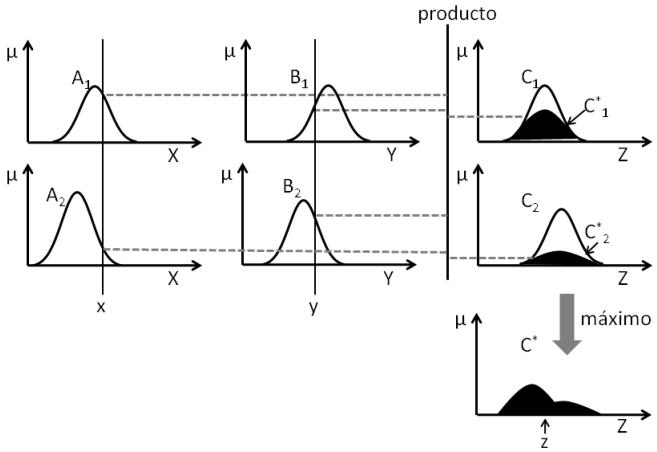


Figura 1.18: *Razonamiento difuso (máximo-producto) para múltiples reglas con múltiples antecedentes*

en función de la entrada.

Defuzzificación La defuzzificación es una transformación de un espacio de decisión difuso de un universo de discurso, en un espacio de decisión no difuso. Esta transformación sólo se utiliza cuando se necesita una salida nítida del sistema.

Algunos de los métodos de cálculo más utilizados para realizar la defuzzificación son los siguientes:

- Máximo: se elige como valor para la variable de salida aquel para el que la función característica del conjunto difuso de salida es máxima (por ejemplo el primero o último máximo encontrado).

- Media del máximo: se genera como valor de salida la media de todos los puntos cuya función de pertenencia alcanza el máximo. En el caso de un universo de discurso discreto, dicha estrategia se representa como

indica la ecuación 1.29

$$z_0 = \sum_{i=1}^{l} w_i / l \tag{1.29}$$

donde z_0 es la salida, l es el número de puntos que alcanzan el valor máximo y w_i es cada uno de esos puntos.

- Centroide: utiliza como salida del sistema el centro de gravedad de la función característica de salida. Constituye uno de los métodos más utilizados en las aplicaciones de la lógica difusa. En el caso de un universo de discurso discreto, dicha estrategia se representa como indica la ecuación 1.30

$$z_0 = \frac{\sum_{i=1}^{n} \mu(w_i) w_i}{\sum_{i=1}^{n} \mu(w_i)} \tag{1.30}$$

donde n es el total de puntos del universo de discurso.

En la figura 1.19 podemos ver representadas las estrategias de defuzzificación comentadas.

1.4.6. Sistemas basados en reglas difusas

Un sistema basado en reglas difusas (SBRD) obtiene una salida y en función de unas entradas x, $y = f(x)$. Su esquema se puede observar en la figura 1.20, donde se muestra que consta de cuatro elementos: *fuzzificador*, *base de conocimiento*, *motor de inferencia* y *defuzzificador*, que se explican a continuación.

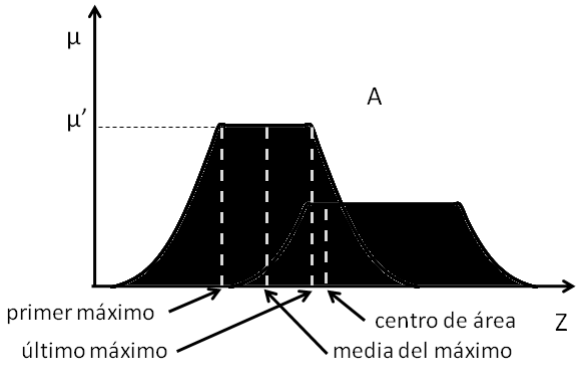


Figura 1.19: *Estrategias de defuzzificación*

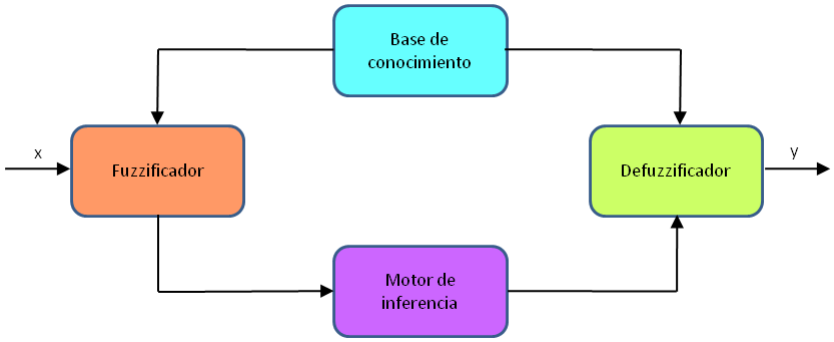


Figura 1.20: *Esquema general de un sistema basado en reglas difusas*

- Fuzzificador: toma como entrada valores nítidos y los transforma en valores del universo de discurso correspondiente, es decir, los transforma en etiquetas de conjuntos difusos, que más tarde se utilizarán para activar reglas. Para diseñar el SBRD hay que elegir la estrategia de fuzzificación y la interpretación del operador fuzzificador, en función del tipo de datos a fuzzificar, del rango en el que están definidos, de la presencia de ruido, etc [Chien, 1990].

- Base de conocimiento: representa el conocimiento del dominio de la aplicación. Por un lado está la *base de datos* que contiene las definiciones necesarias para la manipulación de los datos difusos y la *base de reglas difusas* que caracteriza el conjunto de objetivos comprendidos por las decisiones que se han de tomar para resolver un problema dado. Se ha de decidir la fuente y la derivación de las reglas difusas. Existen diversas fuentes para derivar reglas difusas, entre las que se encuentran la derivación de reglas a partir de conocimiento experto [Mamdani y Assilian, 1975], o el auto-aprendizaje mediante diversas técnicas de las reglas [Procyk y Mamdani, 1979].

  En el diseño del sistema, en cuanto a la base de datos, se ha de determinar subjetivamente una serie de conceptos tales como:

  - Discretización / normalización del universo de discurso. En general la representación va a depender del tipo del universo de discurso: discreto o continuo. Si el universo es continuo se puede discretizar o normalizar teniendo en cuenta las necesidades de representación de la información.
  - Partición difusa de los espacios de entrada y salida. Una variable lingüística en un antecedente de una regla constituye un espacio de entrada, mientras que una variable lingüística en un consecuente de una regla constituye un espacio de salida. En general una variable lingüística se asocia con un conjunto de etiquetas lingüísticas que se definen en el mismo universo de discurso. La partición difusa determina cuántas etiquetas lingüísticas existen. Dicho número determina la granularidad de las reglas y de las decisiones que se toman en el SBRD.

- Completitud. Un SBRD cumple con esta propiedad si para cualquier estado del sistema se puede inferir una acción a tomar. Para que se cumpla dicha propiedad se ha de diseñar la base de datos para que se cubran bien los espacios de entrada y salida y en cuanto al conjunto de reglas se ha de conseguir que todas las condiciones se contemplen.
- Funciones de pertenencia de los conjuntos difusos. Dependiendo de si el universo de discurso es discreto o continuo, la pertenencia a éste se puede definir de una forma numérica o funcional respectivamente. Si la función de pertenencia es numérica, para cada elemento del conjunto se indica su grado de pertenencia. De otro lado, si la función es funcional se especifica mediante una función que proporciona el grado de pertenencia para cada elemento del conjunto.

- Motor de inferencia: encargado de relacionar conjuntos difusos de entrada y de salida, manejando la forma de combinar las reglas.
- Defuzzificador: convierte los conjuntos difusos de salida en valores nítidos. Para el SBRD hay que definir las estrategias de defuzzificación y la interpretación del operador defuzzificador.

1.5. Redes neuronales artificiales

Las Redes Neuronales Artificiales (RNA) son un paradigma de aprendizaje y procesamiento automático inspirado en el sistema nervioso de los animales. Constituyen una importante herramienta dentro de la minería de

datos debido a algunas de sus características tales como: su no linealidad, capacidad de aproximación universal, tolerancia al ruido, capacidad de procesamiento paralelo, capacidad de adaptación mediante aprendizaje, etc. Todas las características anteriores hacen que las RNAs se apliquen a una gran variedad de problemas. Las principales tareas de minería de datos en las que se aplican las RNAs son la clasificación, aproximación funcional, predicción de series temporales y *clustering* [Zhang, 2005].

Las RNAs están inspiradas en el sistema neurológico animal, formado por células nerviosas, *neuronas*, interconectadas entre sí a través de conexiones sinápticas, y que son capaces de responder a estímulos. De forma análoga una RNA está compuesta por *nodos* o neuronas altamente interconectados, según diferentes topologías, que reciben entradas y, trabajando en paralelo, producen salidas en función de su arquitectura (topología y tipo de celda) y de su algoritmo de aprendizaje. Las interconexiones están caracterizadas por pesos que regulan los envíos de señales entre neuronas. Los pesos son también conocidos como *valores sinápticos*, ya que simulan la sinapsis en las conexiones neuronales animales.

Como se ha mencionado anteriormente, la salida de una RNA viene determinada por su arquitectura y su algoritmo de aprendizaje, lo que da lugar a la existencia de diversos modelos de redes neuronales.

Todos los modelos tienen en común que están formados por elementos de procesamiento interconectados que trabajan en paralelo, y a la hora de resolver problemas no necesitan una secuencia explícita de pasos a seguir, sino que la red se va adaptando al problema mediante un periodo de aprendizaje. Esto diferencia este tipo de computación de los programas de ordenador convencionales [Isasi y Galván, 2004].

A continuación se describe la arquitectura de las RNAs, su aprendizaje y una breve reseña histórica de la evolución de las mismas. Para terminar la sección se comenta el diseño evolutivo de las RNAs, dado que es el diseño que se tiene en cuenta para el método propuesto en esta memoria.

1.5.1. Arquitectura

La arquitectura de una red neuronal está definida por su topología y por la función que caracteriza a sus elementos de proceso o neuronas.

La neurona artificial

La neurona artificial (nodo o unidad de procesamiento) es un elemento de proceso que realiza una función simple. Una neurona recibe una serie de entradas a través de sus interconexiones y origina una salida (figura 1.21). Cada una de las conexiones de entrada tiene asociado un peso, si dicho peso es positivo se denomina a la conexión *excitadora* mientras que si el peso es negativo se le denomina *inhibidora*.

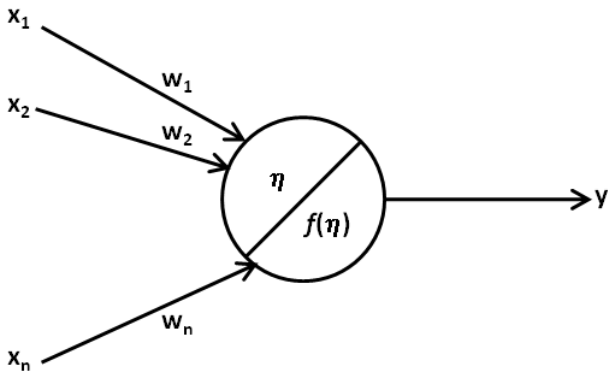


Figura 1.21: *Neurona artificial*

Para obtener la salida, la neurona aplica una serie de funciones que se componen entre ellas. De esta forma podemos decir que intervienen tres tipos de funciones: *base*, *activación* y *salida*.

La función de base, también denominada función de red, de ponderación o propagación toma los valores de las conexiones de entrada y devuelve un valor. Dicha función, η, tiene dos formas típicas:

- Función Lineal de Base (*LBF*) es una función de tipo hiperplano, es decir, una función de primer orden. El valor que devuelve es una combinación lineal de las entradas, x_i, ponderadas por los pesos, w_i, tal y como se muestra en la ecuación 1.31, donde X representa el vector de las entradas y W el de los pesos.

$$\eta(X, W) = \sum_{i=1}^{n} w_i x_i \tag{1.31}$$

- Función de Base Radial (*RBF*) es una función de tipo hiperesférico. Esto implica una función de base de segundo orden no lineal. El valor que devuelve representa la distancia a un determinado patrón. Se muestra en la ecuación 1.32, donde X representa el vector de las entradas y C es un parámetro que representa la posición de la neurona.

$$\eta(X, C) = \sqrt{\sum_{i=1}^{n} (x_i - c_i)^2} \tag{1.32}$$

La salida de la función de base es tomada y procesada por la función de activación (esta puede no existir, siendo en este caso la salida la misma función de base). Existen diferentes formas para la función de activación que

originan distintos modelos, por ejemplo:

- Lineal, la salida es una función lineal de la entrada: $f(\eta) = k\eta$, siendo k una constante. Es usada en algunas redes neuronales como el Adaline [Widrow y Hoff, 1960].
- Con umbral, la función devuelve un valor u otro dependiendo de si la entrada supera o no un determinado umbral Θ. Una función de paso por umbral clásica fue propuesta por McCulloch y Pitts, [McCulloch y Pitts, 1943], que realizaron el primer modelo matemático de RNAs. Dicho modelo se basa en la idea de que las neuronas operan mediante impulsos binarios y la función de activación se calcula como se indica en la ecuación 1.33.

$$f(\eta) = \begin{cases} 1 & si \quad \eta > \Theta \\ -1 & si \quad \eta < \Theta \end{cases} \tag{1.33}$$

- Cualquier otra función, siendo algunas de las más utilizadas la sigmoidal (ecuación 1.34) o la gaussiana (ecuación 1.35), etc.

$$f(\eta) = \frac{1}{1 + e^{-(\eta - \Theta)}} \tag{1.34}$$

$$f(\eta) = e^{-(\eta)} \tag{1.35}$$

Normalmente, la salida de la neurona es directamente el valor obtenido por la función de activación, por tanto la función de salida o transferencia no se considera. En algunas topologías de redes neuronales, sin embargo, se modifica el valor obtenido por la función de activación para acotar la salida

de la neurona o para incorporar un factor de competitividad de forma que la salida no se distribuya en el mismo valor que las neuronas vecinas.

Topología

Las redes neuronales están formadas por conjuntos de neuronas interconectadas. Se conoce como topología de la red a la distribución que toman las neuronas en ella.

Las neuronas se agrupan formando *capas* o *niveles* con un determinado número de neuronas cada una. Una capa es un conjunto de neuronas cuyas entradas provienen de la misma fuente y cuyas salidas se dirigen al mismo destino. Según cual sea la situación de las capas respecto a las entradas o salida de la red, podemos distinguir:

- Capa de entrada, formada por las neuronas que tienen una conexión directa con la entrada de la red. Las neuronas de esta capa reciben las entradas directamente del exterior.
- Capa de salida, formada por las neuronas que tienen una conexión directa con la salida de la red. Las neuronas que la forman envían la salida directamente al exterior.
- Capas ocultas, el número de estas capas oscila entre 0 y un número elevado. Las neuronas de estas capas procesan las entradas que reciben de otra capa y las transforman en salidas que serán entradas para la capa siguiente.

Para algunos autores, las capas de entrada y salida no producen procesamiento y sólo se utilizan como sensores.

La señal de salida de una neurona puede ser una entrada a otra neurona de otra capa, de la misma capa (en este caso se conoce como conexión recurrente) o una entrada de la propia neurona (conexión autorrecurrente).

En cuanto a la topología de una red los parámetros a tener en cuenta son el número de capas, el número de neuronas por capa y el tipo y grado de interconexiones entre las neuronas.

Atendiendo al número de capas en la red podemos diferenciar dos tipos de RNAs:

- Monocapa: existe una única capa de neuronas en la red y entre éstas se establecen conexiones laterales, cruzadas o autorrecurrentes.

- Multicapa: en este tipo de redes existen varias capas de neuronas.

Si tomamos como criterio el modo de interconexión, también podemos establecer una clasificación en la que se distinguen dos tipos de redes:

- Redes hacia delante (*feed-forward*): en este caso la propagación de las señales va desde la capa de entrada hacia la salida y no existen ciclos ni conexiones autorrecurrentes.

- Redes hacia atrás (*feed-back*): en este otro tipo, las salidas pueden conectarse como entradas de neuronas de niveles previos o del mismo nivel, incluyéndose ellas mismas, es decir, las señales viajan en ambas direcciones y existen ciclos en la topología.

1.5.2. Aprendizaje

Como se ha comentado las RNAs tratan de emular características semejantes a las redes de neuronas biológicas. Una de estas características es la de aprender de la experiencia, es decir, las RNAs son capaces de generalizar a partir de casos presentados y aplicar lo aprendido a nuevos casos.

El aprendizaje en la red se va realizando mediante la modificación de los pesos en las conexiones entre las neuronas en respuesta a los ejemplos de entrada que se le proporcionan a la red. Los cambios que se producen durante el proceso de aprendizaje se pueden centrar en la destrucción, modificación y creación de conexiones entre las neuronas.

La finalización del periodo de aprendizaje puede ocurrir cuando se cumpla un determinado número de ciclos, cuando el error que comete la red cae por debajo de una cantidad establecida o cuando la modificación que se produce en los pesos deja de ser significativa.

Básicamente se pueden considerar dos tipos de aprendizaje para RNAs:

- Supervisado: es un tipo de aprendizaje controlado de tal manera que se conoce cuál debe ser la salida para una entrada determinada. Se comprueba la salida de la red con la salida deseada y en el caso de que no coincidan se modifican los pesos de las conexiones, a fin de conseguir aproximar la salida de la red a la deseada. Existen tres formas de realizar este tipo de aprendizaje:
    - Aprendizaje por corrección de error: se ajustan los pesos para minimizar el error obtenido entre la salida de la red y la deseada.

Este es el tipo de aprendizaje más usado.

- Aprendizaje por refuerzo: durante el entrenamiento no se indica la salida deseada pero se van dando refuerzos positivos o negativos en función de que la salida de la red se aproxime a la deseada o no, y en función de dichos refuerzos se van adaptando los pesos.

- Aprendizaje estocástico: se modifican los pesos de forma aleatoria y se va evaluando su efecto.

- No supervisado: en este caso no se conoce cuál debe ser la salida de la red, de forma que la red no recibe ningún tipo de información del entorno que le guíe en la obtención de la salida. En general se suelen considerar dos formas de realizar este tipo de aprendizaje:

  - Aprendizaje Hebbiano ([Hebb, 1949]): consiste en el ajuste de los pesos de acuerdo con la correlación entre las neuronas conectadas. De esta forma si dos neuronas conectadas son activas se refuerza su conexión, por el contrario si una es activa y otra pasiva se debilita su conexión.

  - Aprendizaje competitivo: se presenta a la red una información de entrada y las neuronas compiten por ser la que se activa, así queda una o una por grupo como neurona vencedora.

En el trabajo descrito en esta memoria de tesis se utilizará aprendizaje supervisado por corrección de error.

1.5.3. Breve reseña histórica de las redes neuronales artificiales

Los primeros desarrollos sobre RNAs se producen a principios de los años 40 cuando McCulloch y Pitts [McCulloch y Pitts, 1943] realizaron el primer modelo matemático de RNA, basado en la idea de neuronas que operan mediante impulsos binarios. Posteriormente Hebb [Hebb, 1949] desarrolla un procedimiento matemático de aprendizaje, que se sigue usando en los modelos actuales. En 1951, Minsky [Minsky, 1954] obtuvo los primeros resultados prácticos en RNAs: diseñó una máquina con 40 neuronas cuyas conexiones se ajustaban por medio de aprendizaje.

Una generalización del modelo de células de McCulloch-Pitts añadiéndole aprendizaje da lugar a la aparición en 1957 del Perceptrón, debido a Rosenblatt [Rosenblatt, 1957]. Primero desarrolló un modelo de dos niveles que era capaz de ajustar los pesos de las conexiones entre los niveles de entrada y salida. Algunos de los trabajos que Rosenblatt realizó en torno al Perceptrón se presentan en su libro *Principles of Neurodynamics* [Rosenblatt, 1962].

Dos años después Widrow diseñó una RNA similar al Perceptrón, llamada ADALINE [Widrow y Hoff, 1960] (*Adaptative Linear Element*). Las diferencias entre el Perceptrón y ADALINE son pequeñas pero sus aplicaciones son distintas.

A finales de la década de los 60's Minsky publica un libro [Minsky y Papert, 1969] donde generaliza las limitaciones que podía tener la expansión del Perceptrón de simple capa al Perceptrón Multicapa. Esto dio lugar a que se perdiera el interés que habían despertado las RNAs. A pesar

de todo, los trabajos en el área continuaron y Grossberg desarrolló su Teoría de Resonancia Adaptativa (*Adaptative Resonante Theory*, ART), que contemplaba nuevas hipótesis sobre los principios que gobiernan los sistemas neuronales biológicos [Grossberg, 1976a,b]. Estas ideas sirvieron para que Carpenter y Grossberg dieran lugar a tres clases de arquitecturas de RNAs: ART 1, ART 2, y ART 3 [Carpenter y Grossberg, 1983, 1987a,b].

Paul Werbos, a mediados de los 70, es uno de los pioneros en el desarrollo del algoritmo de aprendizaje *Back-propagation* [Werbos, 1974], sin embargo pasaron algunos años hasta que este famoso método se popularizara para el aprendizaje de redes perceptrones multicapa.

Otra teoría importante fue la aportada por Kohonen que presentó un modelo de red denominado mapas auto-organizados (*self-organizing maps*, SOM). Este tipo de red posee un aprendizaje no supervisado competitivo [Kohonen, 1982]. Más tarde realizó investigaciones en métodos de aprendizaje y desarrolló el LVQ (*Learning Vector Quantization*, un sistema de aprendizaje competitivo [Kohonen, 1988].

Hopfield, basándose en trabajos anteriores de Hebb, utiliza ciertas reglas de aprendizaje para un tipo de redes recurrentes conocidos en la actualidad como modelos de Hopfield [Hopfield, 1982]. Más tarde estos trabajos sirven de base para creación del algoritmo de la máquina de Boltzmann [Hinton y otros, 1984], que entre otras utiliza técnicas como el *simulated annealing*. Extendiendo a un mayor orden el algoritmo de la máquina de Boltzmann, Sejnowski junto a Hinton presentan la primera red de neuronas artificial que reconocía un algoritmo de aprendizaje para una red de tres niveles [Sejnowski y Hinton, 1986].

Anderson, trabajó con un modelo de memoria basado en la asociación de activaciones de las sinapsis de una neurona. Siguiendo también el planteamiento de Hebb y empleando un nuevo método de corrección del error creó un nuevo modelo llamado *Brain-state-in-a-box* [Anderson y Murphy, 1986].

Broomhead realiza unos de los estudios más importantes sobre redes de funciones de base radial [Broomhead y Lowe, 1988]. La idea básica de este tipo de redes fue desarrollada en los años 30 con el nombre de método de funciones potenciales. También por las mismas fechas, Kosko creó una familia de paradigmas de RNAs que extienden las memorias autoasociativas de Hebb de un nivel, a dos niveles utilizando aprendizaje sin supervisión [Kosko, 1988].

1.5.4. Diseño evolutivo de redes neuronales artificiales

Como se ha visto, en el diseño de las redes neuronales se ha de fijar su arquitectura (número de capas, cantidad de neuronas por capa, tipo de neurona, grado de conectividad y el tipo de conexión entre neuronas) y su algoritmo de aprendizaje. Por tanto, son muchos los parámetros que hay que fijar de antemano y para ello es necesario tener conocimiento acerca del dominio del problema que se intenta resolver. En la mayoría de los casos no se tiene dicho conocimiento o es escaso de forma que algunos métodos de diseño se basan en prueba y error.

Los métodos clásicos de diseño de redes neuronales, vistos anteriormente, presentan el inconveniente de quedar atrapados en óptimos locales en el espacio de búsqueda de los mejores parámetros.

Una posible solución es hibridar las redes neuronales con los algoritmos evolutivos, dado el potencial que presentan los últimos para realizar búsquedas. De esta forma se aplican algoritmos evolutivos para obtener los parámetros óptimos de la red neuronal. La aplicación de los algoritmos evolutivos a las redes neuronales se conoce también como neuro-evolución.

En el diseño evolutivo de redes neuronales se puede detectar un problema conocido como *problema de la permutación*, que se produce cuando se tienen redes que son funcionalmente equivalentes aunque sus representaciones (genotipos) son diferentes. Esto implica que en el algoritmo evolutivo se tienen individuos diferentes, pero que representan redes con funcionamiento idéntico. Si se aplica entre ellas el operador de cruce producen descendientes con valor de adaptación bajo. Esto se ha intentado solucionar no utilizando el operador de cruce o ideando operadores de cruce que eviten el problema.

Existen diversos enfoques en el diseño evolutivo de las redes neuronales, los más importantes se centran en la evolución de los pesos dentro de una topología fija, la evolución de la topología y la evolución de las reglas de aprendizaje [Balakrishnan y Honavar, 1995; Grönroos, 1998; Yao, 1999; Tettamanzi y Tomassini, 2001; Castillo y otros, 2001], los cuales se describen a continuación.

Diseño evolutivo de los pesos dada una arquitectura fija

Normalmente, el aprendizaje de los pesos de una red se formula en términos de minimización de una función de error. Dicho error suele calcularse como el error cuadrático medio total entre las salidas de los ejemplos y las salidas calculadas por la red. La mayoría de los algoritmos de

entrenamiento no evolutivos, tales como *Back-propagation* y otros algoritmos de gradiente conjugado [Rumelhart y otros, 1986; Widrow y Lehr, 1990; Moller, 1993] se basan en aplicar técnicas de gradiente para encontrar mínimos en el espacio del error. Sin embargo presentan una serie de problemas tales como que pueden quedar atrapados en mínimos locales y además requieren que dicho espacio de búsqueda sea diferenciable [Sutton, 1986]. Los algoritmos evolutivos no requieren información del gradiente por lo que pueden realizar búsquedas en cualquier tipo de espacio de error. Los algoritmos evolutivos pueden evitar quedar atrapados en mínimos locales y se presentan como una alternativa a los métodos de entrenamiento tradicionales.

En el diseño de un algoritmo evolutivo para los pesos de una red, se tiene decidir la representación genotípica de los pesos y el modelo de evolución que se van a utilizar. Existen dos alternativas para representar los pesos:

- Representación mediante cadenas de bits. Esta es la representación clásica que utilizan los algoritmos genéticos. En este casos los pesos se representarían como una secuencia de bits y el individuo entero (la red) como la concatenación de los pesos. Las ventajas de este tipo de representación es su simplicidad y la facilidad de aplicar los operadores típicos de mutación y cruce. Entre sus inconvenientes está la conversión genotipo-fenotipo y su pobre escalabilidad, ya que a medida que aumenta la precisión aumenta la longitud del individuo, lo cual hace que la evolución sea ineficiente. Un estudio sobre dicho problema se puede encontrar en [Bernier y otros, 2000].

- Representación mediante números reales. En este caso los pesos se

representan como números reales y el individuo es un vector de reales. En este caso no se podrán utilizar los operadores clásicos de mutación y cruce. En [Montana y Davis, 1989] se presenta este tipo de codificación para la utilización de un algoritmo genético y se definen sus propios operadores. Posteriormente, las propuestas que utilizan este tipo de representación pueden utilizar cualquiera de los operadores genéticos para representación real propuestos en la bibliografía especializada. Otra forma de evolucionar este tipo de representación es utilizar programación evolutiva o estrategias evolutivas, que están pensadas para la optimización continua [Fogel y otros, 1990]. En este tipo de representación lo normal es transformar los genotipos en fenotipos, utilizar los pesos en la red y medir el error que se produce; el valor de adaptación de cada individuo se fija en función de dicho error.

En algunas ocasiones se suele considerar un entrenamiento híbrido de los pesos de la red. Dado que uno de los inconvenientes de los algoritmos evolutivos es que presentan cierta ineficiencia para afinar una búsqueda local, se podría combinar con un algoritmo no genético de búsqueda local. De esta forma el algoritmo genético se encargaría de realizar una búsqueda global para asignar unos pesos iniciales y después se puede aplicar un algoritmo como *Back-propagation* que afine la solución realizando una búsqueda local [Belew y otros, 1991].

Evolución de la arquitectura de la red

Aquí, el objetivo es realizar un diseño evolutivo de la arquitectura de la red (topología y función de transferencia de las neuronas).

La arquitectura que se fije para una red es muy importante ya que determina la capacidad de aprendizaje y generalización de la misma. Para una tarea de aprendizaje determinada, una red con pocas conexiones y pocos nodos puede que no sea capaz de alcanzar errores pequeños debido a su limitaciones en el aprendizaje. En el otro lado, una red con excesivas conexiones puede tener problemas de sobre-entrenamiento y baja generalización.

En la práctica, el diseño de la arquitectura suele ser un trabajo heurístico que depende del conocimiento experto de la persona que lo realiza a través de un proceso de prueba y error. Existe, no obstante, alguna aproximación al diseño automático de arquitecturas de redes como son los algoritmos incrementales (constructivos) o decrementales (destructivos o de poda) [Fahlman y Lebiere, 1990; Frean, 1990].

Los métodos incrementales comienzan con una red pequeña y van añadiendo elementos hasta que la red pueda solucionar el problema. Por el contrario los métodos decrementales se basan en entrenar una red de mayor tamaño del necesario e ir eliminando elementos innecesarios. Un inconveniente de estos métodos, además de poder quedar atrapados en mínimos locales, es que los criterios para añadir o eliminar elementos depende del problema a resolver y de la arquitectura de la red.

De nuevo en el diseño de la arquitectura, los algoritmos evolutivos se presenta como una alternativa a dichos métodos clásicos, ya que la computación evolutiva es capaz de hacer búsquedas globales en espacios discontinuos y multimodales.

En el campo de la evolución de la arquitectura de redes neuronales [Yao,

1999] casi todos los estudios se refieren a la evolución de la topología y no se ha trabajado tanto en la evolución de las funciones de transferencia de las neuronas.

En la evolución de la arquitectura se ha de determinar la representación del genotipo y el algoritmo genético a utilizar (operadores genéticos). Existen dos formas principales de codificación utilizada para la topología de la red:

- Codificación directa (o fuerte). Según este esquema, las conexiones de la red se representan mediante una matriz. De esta forma si existen N nodos, la topología se representa mediante una matriz T de $N \times N$ donde $t_{ij} = 1$ si existe una conexión entre la neurona i y la neurona j. Cuando valor $t_{ij} = 0$ no existe ninguna conexión entre las neuronas i y j y así se tiene directamente una codificación binaria. Existen diferentes ejemplos de uso de este tipo de representación como [Miller y otros, 1989; Whitley y otros, 1990]. Esta aproximación se puede ampliar de forma que los elementos de la matriz pueden representar los pesos de las conexiones y evolucionar ambas cosas, pesos y arquitectura. Este tipo de representación presenta como ventajas su facilidad de implementación y manejo de los operadores genéticos. El problema que presenta este tipo de codificación es que no escala bien cuando se tienen arquitecturas complejas. En algunas ocasiones se puede reducir el tamaño de las matrices partiendo de restricciones propias de la red.

- Codificación indirecta (débil). En este caso, para intentar reducir la longitud de la representación, lo que se codifica en un individuo son sólo algunas de las características de la arquitectura. Por ejemplo se

puede codificar el número de capas, el número de nodos por capa, etc., el resto de parámetros los decide el algoritmo de entrenamiento. La forma en que cada conexión se codifica está predefinida de acuerdo a cierto conocimiento previo sobre el dominio del problema, o bien mediante ciertas reglas determinísticas. Este tipo de codificación es más compacto en cuanto a la representación del genotipo. Existen diferentes métodos para realizar la codificación indirecta, por ejemplo:

- Representación paramétrica: las arquitecturas se especifican mediante parámetros como número de capas, número de neuronas por capa, etc. La mayoría de los métodos se basan en este esquema. Para utilizar esta aproximación hay que desarrollar operadores genéticos específicos ya que individuos funcionalmente correctos pueden generar descendientes no funcionales. En [Harp y otros, 1990] podemos encontrar un ejemplo de este tipo de representación.

- Representación de reglas de desarrollo: en este caso se codifican las reglas que sirven para construir la arquitectura. Una regla de construcción suele ser una ecuación recursiva o una regla de generación. Ejemplos de este tipo de representación los podemos encontrar en [Kitano, 1990; Gruau, 1992].

- Representaciones co-evolutivas: en [Andersen y Tsoi, 1993] se usa una representación en la que un individuo representa una sola neurona de la red. El valor de adaptación se obtiene por medio de técnicas de asignación de crédito.

Evolución de las reglas de aprendizaje

Un algoritmo de entrenamiento puede conseguir diferentes resultados según a qué arquitectura se aplique. De hecho, el diseño de un algoritmo de entrenamiento o reglas de aprendizaje que se usan para ajustar los pesos de una red, depende de la arquitectura que se tenga. Dicho diseño se complica cuando no se tiene suficiente conocimiento a priori sobre la arquitectura de la red.

Por tanto se hace necesario el desarrollo de un método automático que vaya adaptando el aprendizaje según cual sea la arquitectura y el problema al que se aplique. De esta forma parece bastante razonable pensar en técnicas evolutivas que adapten dichas reglas de aprendizaje.

En algunos trabajos lo que se propone es realizar un ajuste evolutivo de los parámetros que rigen las reglas de aprendizaje y no de la totalidad de las reglas. Así en [Belew y otros, 1991; Kim y otros, 1996] se propone una técnica para encontrar los mejores parámetros del algoritmo *Back-Propagation* manteniendo predefinida y fija la arquitectura. En [Harp y otros, 1989; Merelo y otros, 1993] se codifican tanto los parámetros de las reglas como la arquitectura de la red y evolucionan ambos.

En los trabajos anteriores no se evoluciona las reglas de aprendizaje, sólo sus parámetros. La adaptación de las reglas hace que éstas y la red en general tengan un comportamiento dinámico, el problema es la complejidad que conlleva la representación de esta conducta dinámica. Hay que establecer ciertas restricciones para simplificar dicha representación y por tanto reducir el espacio de búsqueda. Así por ejemplo se ha de suponer que la regla de adaptación de pesos dependa sólo de información local y que las reglas de

aprendizaje sean las mismas para todas las conexiones de la red. Chalmers [Chalmers, 1990] hizo uno de los primeros estudios en esta dirección y define una regla de aprendizaje como combinación lineal de una serie de variables.

1.6. Conclusiones

En este capítulo se ha descrito el descubrimiento de conocimiento en bases de datos y con más detalle la fase de minería de datos, dado que el método que se presenta en esta memoria se encuadra dentro de ella. Se han enumerado las tareas de minería de datos así como las técnicas utilizadas para abordar éstas.

Se ha realizado un estudio de tres técnicas *soft-computing* como son la computación evolutiva, la lógica difusa y las redes neuronales. Éstas tecnologías no deben considerarse excluyentes en la resolución de problemas sino que deben cooperar entre sí para lograr sistemas híbridos de componentes especializados. Esta máxima es la que se sigue en el desarrollo de los métodos que se van a presentar, ya que son hibridaciones de las tres.

De la computación evolutiva se han descrito brevemente los diferentes paradigmas, así como los componentes generales de un algoritmo evolutivo. Como tipo especial de algoritmos evolutivos, se han analizado las características de los métodos co-evolutivos y cooperativos-competitivos.

Dentro de la lógica difusa se han estudiado los fundamentos teóricos necesarios para realizar inferencias a partir de conocimiento experto.

En cuanto a las redes neuronales se ha explicado su arquitectura, los métodos de aprendizaje y una breve reseña histórica. El capítulo finaliza con

un estudio más detallado de su diseño evolutivo, como ejemplo de hibridación de técnicas *soft-computing*.

Capítulo 2

Redes de Funciones de Base Radial

En el capítulo anterior, se describieron las redes neuronales artificiales como un paradigma *soft-computing* de aprendizaje y procesamiento automático inspirado en el sistema nervioso animal.

Dado que el método que se presentará es un método para el diseño evolutivo de Redes de Funciones de Base Radial, en este capítulo se estudiarán dicho tipo de redes, su arquitectura y las técnicas utilizadas para su diseño. Entre estas técnicas de diseño destacan las numéricas, las de *clustering*, las incrementales/decrementales y las evolutivas.

2.1. Funciones de base radial

Las Funciones de Base Radial (RBFs) se definen por primera vez en los trabajos de Powell y Haykin [Powell, 1985; Haykin, 1999]. Una Función de Base Radial es una función de segundo orden no lineal que devuelve un valor

que representa la distancia a un determinado patrón de referencia.

Una RBF, $\phi_i : R^n \rightarrow R$, se puede expresar como (2.1):

$$\phi_i(\vec{x}) = \phi(\parallel \vec{x} - \vec{c_i} \parallel / r_i) \tag{2.1}$$

La función tiene una salida simétrica, y se incrementa (o decrementa) de forma monótona con respecto a un centro, $\vec{c_i} \in R^n$. Tiene un factor de escala, $r_i \in R$, conocido como radio y el operador $\parallel \parallel$ se suele tomar como la distancia Euclídea en R^n (ecuación 2.2),

$$\sqrt{\sum_{j=1}^{n}(x_j - c_{ij})^2} \tag{2.2}$$

siendo n la dimensión de los vectores $\vec{x}$ y $\vec{c_i}$.

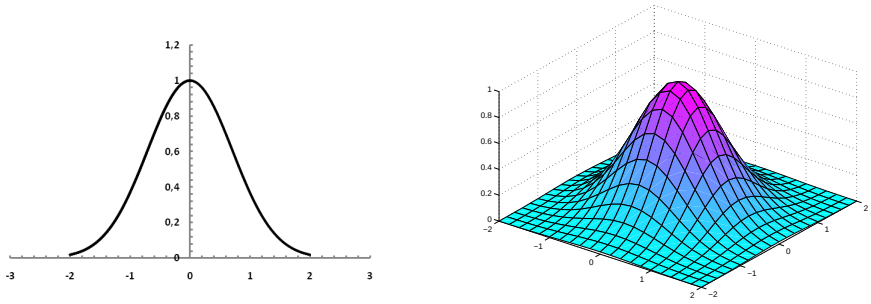
La activación de estas funciones es proporcional a la cercanía, medida con la norma euclídea, entre el patrón de entrada y el centro de la RBF, y depende de la función ϕ elegida. En resumen se puede decir que una RBF, ϕ_i, está definida por tres parámetros:

- Centro ($\vec{c_i}$), que es un vector que coincide en dimensión con el espacio de entradas.
- Radio (r_i), que permite escalar la distancia de los puntos de entrada con respecto al centro.
- Tipo de función de activación.

Existen distintas funciones de activación para las RFBs, tales como gaussianas, *splines*, inversas multicuadráticas, etc. En [Rojas y otros, 1997]

se realiza un estudio sobre las formas que puede adoptar una RBF, concluyendo que una de las mejores elecciones es una función gaussiana. En la ecuación 2.3 se muestra la fórmula para una RBF con forma gaussiana y en la figura 2.1 se muestra una RBF gaussiana con $c = 1$ y $r = 0$.

$$\phi_i(\vec{x}) = e^{-(\|\vec{x}-\vec{c}_i\|/r_i)^2} \tag{2.3}$$



(a) *2D* (b) *3D*

Figura 2.1: *RBF gaussiana con c=0 y r=1*

La función gaussiana decrece de forma monótona conforme se aleja del centro. Por tanto las RBFs gaussianas son locales, es decir, dan una respuesta significativa sólo en aquellos puntos que son cercanos al centro y por tanto su respuesta es finita.

A partir de ahora, se va a considerar la función gaussiana siempre que se hable de RBFs.

2.2. Arquitectura de una red de funciones de base radial

Las funciones de base radial constituyen una clase de funciones que, en principio, se podrían utilizar en cualquier modelo (lineal o no lineal) y en nodos de cualquier tipo de redes (de una capa o de múltiples capas) [Orr, 1995]. Sin embargo desde que Broomhead y Lowe presentaron su trabajo [Broomhead y Lowe, 1988], se ha asociado este tipo de funciones sólo a un tipo de red conocida por utilizarlas, las Redes de Funciones de Base Radial (RBFNs) y se considera una red de una sola capa oculta.

La mayor contribución a la teoría, diseño y aplicaciones de las RBFNs se debe a trabajos como [Moody y Darken, 1989; Poggio y Girosi, 1990] donde aparecen las primeras implementaciones de este tipo de redes. Uno de los principales objetivos de estos autores era construir RNAs que fueran capaces de aprender en menos tiempo que otro tipo de redes como los perceptrones multicapa.

Por tanto, una RBFN se define como una red neuronal con tres capas (figura 3.2):

- Capa de entrada, formada por n nodos. Las neuronas de esta capa son un mero receptor para los datos de entrada, de forma que las transmiten a la siguiente capa sin realizarles ningún procesamiento. Cada neurona de entrada se corresponde con una característica de un patrón de entrada.
- Capa oculta con las m neuronas. Éstas neuronas de la capa oculta se activan mediante una función de base radial, por lo que a cada una

de dichas neuronas también se le conoce como RBF. La salida de la función base será alta cuando el centro de la misma y el patrón de entrada sean similares, siempre teniendo en cuenta el factor de escala que representa el radio. La transformación en esta capa es local y no lineal sobre sus entradas y depende del centro y el radio de cada RBF. Los pesos w_{ij} indican la contribución de la RBF a su respectivo nodo de salida.

- Capa de salida, que puede tener uno o más nodos. Los nodos de la capa de salida de la red implementan la suma ponderada de las salidas de los nodos de las RBFs de la capa oculta (ecuación 2.4).

$$f(\vec{x}) = \sum_{i=1}^{m} w_{ij}\phi_i(\vec{x}) \tag{2.4}$$

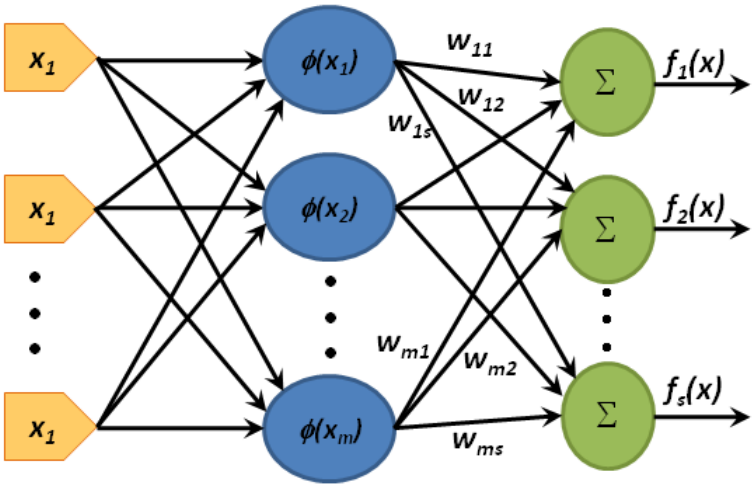


Figura 2.2: *Arquitectura típica de una RBFN*

Las RBFNs son redes con conexiones hacia adelante, la propagación de las señales va desde la capa de entrada hacia la salida y no existen

realimentaciones de una capa hacia otra anterior o hacia ella misma.

Este tipo de red tiene tal y como se ha visto una topología simple (sólo una capa oculta) y su respuesta es local, estas características les confieren a las RBFNs un grado de interpretabilidad mayor que el de la mayoría de las RNAs. De hecho, la equivalencia funcional entre RBFNs y sistemas interpretables como los sistemas de inferencia difusa se demuestra en [Jang y Sun, 1993] y otros trabajos proponen métodos de extracción de reglas difusas interpretables a partir de RBFN entrenadas [Jin y Sendhoff, 2003]. Para poder extraer reglas difusas desde la RBFN, el número de RBFs deber ser pequeño y éstas no deben ser muy similares entre sí.

Además de su topología simple y su capacidad de respuesta local, otra de las importantes características de las RBFNs es su capacidad de aproximación universal para toda función continua y acotada [Poggio y Girosi, 1990; Park y Sandberg, 1991, 1993].

Como ya se ha mencionado las primeras investigaciones sobre este tipo de redes aparecen a finales de los años ochenta [Powell, 1985; Broomhead y Lowe, 1988; Moody y Darken, 1989]. Desde entonces se ha probado la eficiencia de este tipo de redes en muchas áreas tales como clasificación de patrones [Buchtala y otros, 2005], aproximación de funciones [Park y Sandberg, 1991], predicción de series temporales [Whitehead y Choate, 1996] y otras aplicaciones específicas como asignación de créditos bancarios [Lacerda y otros, 2005], detección de intrusos en redes [Zhang y otros, 2005], control de procesos [Huang y otros, 2008], reconocimiento de rostros [Er y otros, 2005], predicción de series financieras [Sun y otros, 2005], control de procesos [Huang y otros, 2008] y diagnosis médica [Maglogiannis y otros, 2008; Marcos y otros, 2008], o reconocimiento de imágenes [Siddiqui y otros,

2009] entre otras.

2.3. Diseño de redes de funciones de base radial

En una RBFN el número de nodos en la capa de entrada viene determinado por el número de características o atributos en el problema. Hay casos en los que el número de características es muy alto y entonces sólo se toman las más representativas. El número de elementos de la capa de salida también depende del tipo de problema que se esté resolviendo.

Por tanto, el objetivo en el proceso de diseño de cualquier RBFN, es determinar el número (m) y tipo (ϕ) de las RBFs, sus parámetros, centros ($\vec{c}_i$) y radios (r_i) y los pesos óptimos (w_i) de las conexiones entre las RBFs y las neuronas de la capa de salida.

El tipo de función base es algo que se suele fijar de antemano y por tanto no se considera como parámetro a tratar dentro del aprendizaje. En muchas ocasiones también se fija el número de neuronas en la capa oculta. Esto implica que, normalmente, en los algoritmos de diseño de RBFNs, el aprendizaje se dirija a la determinación de los centros, radios y pesos.

La determinación de los centros y los radios, se realiza mediante la optimización en el espacio de entrada, ya que cada neurona va a representar una zona diferente en dicho espacio. Para la determinación de los pesos, la optimización se realiza en base a las salidas que se desea obtener (salidas deseadas), ya que la red se utiliza para aproximar las relaciones entre el conjunto de variables de entrada y salida que definen el problema. Los algoritmos tradicionales de aprendizaje actúan en dos fases:

- Determinación de los centros y los radios de las RBFs.

- Cálculo de los pesos de las conexiones entre las RBFs y las neuronas de la capa de salida.

A continuación se realiza una revisión de los principales métodos y técnicas que se han utilizado para el diseño de RBFNs. Dichos métodos se pueden agrupar en métodos numéricos, algoritmos de *clustering*, técnicas incrementales/decrementales y por último los métodos evolutivos.

2.3.1. Métodos numéricos

Los métodos numéricos son métodos matemáticos basados en resolución de ecuaciones o técnicas de gradiente.

Algoritmos basados en el gradiente

En el caso de las RBFNs, al igual que en el diseño de otro tipo de redes, se pueden utilizar técnicas de *gradiente* que actualizan los parámetros de forma iterativa ante cada nuevo patrón del conjunto de entrenamiento.

El objetivo es minimizar el error entre la salida de la red y la salida real, por lo que el proceso de aprendizaje está guiado por la minimización de la función del error de aproximación, mostrada en la ecuación 2.5

$$E = \sum_{i=1}^{n} (f(\vec{x}_i) - y_i)^2 \tag{2.5}$$

donde $f(\vec{x}_i)$ es la salida de la red e y_i la salida real.

En el caso de utilizar funciones gaussianas, se tienen las ecuaciones 2.6, 2.7 y 2.8 [Ghost y otros, 1992], para actualizar el vector de pesos, centro y radio respectivamente:

$$\Delta w_j = \alpha_1(y_i - f(\vec{x}_i))\phi(\vec{x}_i) \tag{2.6}$$

$$\Delta \vec{c}_j = \frac{\alpha_2(y_i - f(\vec{x}_i))\phi(\vec{x}_i) \parallel \vec{x}_i - \vec{c}_j \parallel w_j}{r_j^2} \tag{2.7}$$

$$\Delta r_j = \frac{\alpha_3(y_i - f(\vec{x}_i))\phi(\vec{x}_i) \parallel \vec{x}_i - \vec{c}_j \parallel^2 w_j}{r_j^3} \tag{2.8}$$

donde α_1, α_2 y α_3 son constantes que definen la variación a aplicar o velocidad de aprendizaje. Las ecuaciones anteriores representan una particularización del algoritmo *LMS (Least Mean Squares)* [Widrow y Hoff, 1960] para las RBFNs.

En este método de aprendizaje supervisado se inicializan los parámetros de la red de forma aleatoria con valores cercanos a 0. La desventaja que presenta esta técnica es que se puede quedar atrapado en mínimos locales.

Otros ejemplos de estos métodos se encuentran en [Ampazis y Perantonis, 2002; Neruda y Kudová, 2005].

Algoritmos de determinación de pesos basados en la resolución de matrices

Una vez fijados los centros y los radios de las neuronas de la capa oculta se pueden utilizar algoritmos para determinar los pesos que conectan dichas

RBFs con las neuronas de la capa de salida.

Estos algoritmos intentan encontrar un conjunto de pesos óptimo que aproximen la salida de la red a la salida real. El conjunto de pesos óptimo se calculará a partir de la ecuación 2.9, expresada en notación matricial.

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_m \end{bmatrix} \begin{bmatrix} \phi_{11} & \dots & \phi_{1m} \\ \phi_{21} & \dots & \phi_{2m} \\ \vdots & \vdots & \vdots \\ \phi_{n1} & \dots & \phi_{nm} \end{bmatrix} \tag{2.9}$$

La ecuación 2.9 se puede expresar de forma más compacta según la expresión 2.10:

$$\vec{y} = \Phi\vec{w} \tag{2.10}$$

donde $\vec{y}$ es el vector de la salida objetivo, $\vec{w}$ es el vector de pesos y Φ es la matriz de activación de las RBFs. Los pesos, a partir de la ecuación vista, se calculan multiplicando el vector de salida por la inversa de la matriz de activación, ecuación 2.11.

$$\vec{w} = \vec{y}\Phi^{-1} \tag{2.11}$$

Los métodos más utilizados determinan el vector de pesos resolviendo sistemas de ecuaciones. Entre ellos destacan la *descomposición de Cholesky* [Golub y Van Loan, 1996], la *descomposición en valores singulares (SVD)* [Cliff, 1983; Golub y Van Loan, 1996] y el *método de los mínimos cuadrados ortogonales (OLS)* [Chen y otros, 1991].

Descomposición de Cholesky

La descomposición de Cholesky [Golub y Van Loan, 1996] es la técnica más rápida para la resolución de sistemas de ecuaciones lineales, pero sólo se puede aplicar a aquellos sistemas que estén descritos mediante una matriz cuadrada, simétrica y definida positiva.

Se han de realizar una serie de transformaciones en la ecuación 2.10 para que se cumplan las restricciones. Una vez que se han realizado las transformaciones la matriz de activación debe verificar otra serie de condiciones que dependen de las características de las RBFs situadas: las RBFs deben cubrir todo el espacio de entrada y ser independientes para que el sistema de ecuaciones se pueda resolver mediante la descomposición de Cholesky. Es decir, si existen patrones no cubiertos por ninguna RBF o existe un solapamiento excesivo entre RBFs se producen matrices de activación singulares que esta técnica no es capaz de resolver.

Descomposición en valores singulares

El método de descomposición en valores singulares (SVD) [Cliff, 1983; Golub y Van Loan, 1996] se ha utilizado en procesamiento de señales, compresión de datos y para resolver sistemas de ecuaciones lineales. Esta técnica se puede utilizar cuando las RBFs estén mal colocadas y se produzcan matrices de activación singulares o casi singulares. Cuando dos funciones bases son casi idénticas (están muy solapadas) en la matriz de activación se producen dos columnas prácticamente iguales, mientras que si una función base no se activa para casi ningún punto, se producirá una columna casi nula en la matriz.

La técnica SVD permite que se resuelva cualquier sistema de ecuaciones,

y se puede tener una reducción del error en la salida de la red. También se puede utilizar para eliminar alguna función base cuyo valor singular asociado tuviera una magnitud pequeña (la RBF no está cubriendo casi ningún punto) y el error de aproximación no se vería afectado.

El inconveniente que presenta esta técnica es que por la forma en que selecciona las funciones base más importantes para la red, no tiene en cuenta la salida esperada del sistema para cada vector de entrada. Si en la red existen dos RBFs muy solapadas, el método eliminaría cualquiera de las dos sin evaluar la influencia que dicha modificación causaría en el error de aproximación de la red.

Método de los mínimos cuadrados ortogonales

Otra opción para resolver un sistema de ecuaciones lineales es el método de los mínimos cuadrados ortogonales (OLS) [Chen y otros, 1991]. Esta técnica se puede aplicar, al igual que la anterior aunque las RBFs estén mal colocadas (con zonas de espacio sin cubrir y solapamientos).

OLS es una técnica iterativa que selecciona en cada iteración la columna de la matriz de activación Φ que más contribuye a la disminución del error de aproximación del modelo. El método va transformando las columnas de la matriz de activación en un conjunto de vectores ortogonales. De esta forma el método va seleccionando secuencialmente los centros de la RBF que son ortogonales con la función anteriormente elegida. Para cada RBF elegida se mide la contribución de ésta a la disminución del error. El método se ejecuta hasta que se alcanza un nivel de error aceptable o cuando el número de RBFs elegidas es el deseado.

Se han desarrollado algunas hibridaciones del método OLS con técnicas

de *regularización* [Orr, 1993] produciendo el algoritmo *ROLS* [Chen y otros, 1996], e hibridaciones de éste último con algoritmos genéticos [Chen y otros, 1999].

Técnica de regularización

La regularización es una técnica [Orr, 1993, 1995] que favorece unas soluciones sobre otras añadiendo un término de penalización a la función de costo, o error en este caso, a minimizar.

Dependiendo de la función a minimizar y del término de penalización surgen varias técnicas, entre las que se puede destacar la *regresión límite* [Orr, 1996].

La técnica de regularización utiliza la *matriz de proyección* P, cuya expresión aparece en la ecuación 2.12:

$$P = I_n - \Phi A^{-1} \Phi^T \tag{2.12}$$

donde I_n es la matriz identidad con dimensión n y $A = \Phi^T \Phi$.

La matriz P representa la proyección de vectores de un espacio n-dimensional, donde n es el número de patrones del conjunto de entrenamiento, en un espacio m-dimensional, siendo m el número de funciones base. A partir de esta matriz se pueden definir muchos conceptos importantes como la reducción del error, los efectos tras la inclusión o eliminación de una RBF, etc.

Regresión Límite

La técnica de regresión límite [Orr, 1996] utiliza al igual que la regularización un término de penalización en la función de error a minimizar. De esta forma se define una función de costo a minimizar tal y como la descrita en la ecuación 2.13:

$$C = \sum_{i=1}^{n} \big(y_i - f(\vec{x}_i)\big) + \lambda \sum_{j=1}^{m} w_j^2 \tag{2.13}$$

donde λ es el parámetro de regularización. Con la inclusión de este término se penalizarán redes con pesos muy altos (dependiendo también de λ). El objetivo es conseguir redes con salidas suaves, al tener pesos moderados. Con estas premisas y usando la matriz de proyección se define el vector de pesos óptimo como se indica en la ecuación 2.14,

$$\vec{w} = A^{-1}\Phi^T\vec{y} \tag{2.14}$$

siendo ahora A:

$$A = \Phi^T\Phi + \lambda I_n \tag{2.15}$$

2.3.2. Técnicas de clustering

Las técnicas de *clustering* se desarrollaron originariamente para resolver problemas de clasificación [MacQueen, 1967; Hartigan, 1975] aunque después se han aplicado a diferentes campos en los sistemas difusos [Bezdek, 1981] y redes neuronales artificiales [Chakaravathy y Ghosh, 1996; Karayiannis y

Mi, 1997], entre otros paradigmas.

Un algoritmo de *clustering* divide un conjunto de n observaciones en c grupos o *clusters* diferentes. El número de grupos puede estar establecido o puede ser calculado por el propio algoritmo. A continuación el algoritmo asocia cada ejemplo con un grupo de forma que los elementos de cada grupo comparten una serie de características y se diferencian de los de los otros grupos.

Estos algoritmos se suelen utilizar en el diseño de RBFNs para inicializar los centros de las funciones base, dado que permiten dividir el espacio de los datos de entrada en clases. De esta forma el número de clases o grupos en que se divide el espacio de entrada coincide con el número de RBFs y se sitúa el centro de cada una en el centro geométrico de cada uno de los grupos identificados. Así, los algoritmos de *clustering* estudian el espacio de entrada y permiten inicializar los centros de las funciones base, aunque no los optimicen.

Dentro de los algoritmos de *clustering* se puede establecer una clasificación en:

- Algoritmos de *clustering* no supervisados: la característica común a este tipo de algoritmos es que el mecanismo de aprendizaje (de asignación de grupos) no tiene en cuenta ninguna información aportada por la/s variable/s de salida. Los algoritmos más importantes dentro de esta categoría son: el algoritmo de *K-medias* [Duda y Hart, 1973], algoritmos de las *K-medias difuso* [Bezdek, 1981] y algoritmo *ELBG* [Russo y Patanè, 1999].
- Algoritmos de *clustering* supervisados: en estos se tiene en cuenta

la información suministrada por la/s variable/s de salida. Dentro de este grupo las diferencias se centran en la forma en que utilizan dicha información. Ejemplos importantes de este tipo son: algoritmo de *clustering difuso condicional* [Pedrycz, 1996, 1998], algoritmo de *estimación de grupos alternante (ACE)* [Runkler y Bezdek, 1999] y algoritmo de *clustering para aproximación de funciones* [González y otros, 2002].

A continuación se explican estos algoritmos de *clustering*.

Algoritmo de las k-medias

Esta técnica [Duda y Hart, 1973] divide el conjunto de vectores de entrada $X = \vec{x}_i : i = 1, \ldots, n$ en c grupos. Estos grupos proporcionan una partición de Voronoi $P = P_1, \ldots, P_c$ y cada grupo o *cluster* P_j se define mediante un vector representante del grupo o prototipo $\vec{p}_j$. Para realizar las particiones utiliza la distancia entre vectores, de forma que se define como función objetivo la minimización de D (ecuación 2.16):

$$D = \sum_{j=1}^{c} \sum_{i=1}^{n} \parallel \vec{x}_i - \vec{p}_j \parallel \tag{2.16}$$

donde $\vec{x}_1 \ldots, \vec{x}_n$ son los vectores que se tienen que agrupar, $\vec{p}_1 \ldots, \vec{p}_c$ son los prototipos de cada grupo y $\parallel \parallel$ es la distancia euclídea. Además se define una función, $\mu_{P_j}(\vec{x}_i)$, de pertenencia al vector de entrada $\vec{x}_i$ al grupo representado por el prototipo $\vec{p}_j$ (ecuación 2.17).

$$\mu_{P_j}(\vec{x}_i) = \begin{cases} 1 & \text{si} \quad \| \vec{x}_i - \vec{p}_j \|^2 < \| \vec{x}_i - \vec{p}_l \|^2 \quad \forall l \neq j \\ 0 & \text{en otro caso} \end{cases} \tag{2.17}$$

La función de pertenencia produce una partición de Voronoi P del conjunto de vectores de entrada tal y como se muestra en la ecuación 2.18,

$$P = \bigcup_{j=1}^{c} P_j \quad \text{siendo} \quad P_i \cap P_j = \oslash \quad \forall j \neq i \tag{2.18}$$

donde P_j se define mediante la ecuación 2.19.

$$P_j = \{\vec{x}_i : \mu_{P_j}(\vec{x}_i) = 1\} \tag{2.19}$$

Una partición se puede describir mediante la matriz de partición U, de dimensión $c \times n$, cuyos elementos son las funciones de pertenencia $\mu_{P_j}(\vec{x}_i)$ (ecuación 2.20).

$$U = \left\{ \mu \in \{0,1\} | \sum_{i=1}^{c} \mu_{P_i}(\vec{x}_k) = 1 \ \forall k \quad y \quad 0 < \sum_{k=1}^{n} \mu_{P_i}(\vec{x}_k) < n \ \forall i \right\} \tag{2.20}$$

El algoritmo de las K-medias (figura 2.3) empieza realizando inicializaciones generales como la asignación aleatoria de los prototipos $\vec{p}_j$ a las particiones. Después entra en el ciclo principal en el que va determinar a qué partición pertenece cada vector, calculando su función de pertenencia $\mu_{P_j}(\vec{x}_i)$ (ecuación 2.17). A continuación recalcula los prototipos de las particiones utilizando la ecuación 2.21

$$\vec{p}_j = \frac{\sum_{i=1}^{n} \mu_{P_j}(\vec{x}_i)\vec{x}_i}{\sum_{i=1}^{n} \mu_{P_j}(\vec{x}_i)} \tag{2.21}$$

```
Inicializa δ = ∞
Asignar aleatoriamente el conjunto inicial de prototipos p_j
Mientras (|δ_ant − δ|/δ >= ε) Hacer
     Asignar δ_ant = δ
     Calcular las funciones de pertenencia μ_Pi(x_k) (usando 2.17)
     Calcular los prototipos p_j (usando 2.21)
     Calcular δ (usando 2.22)
Fin_Mientras
```

Figura 2.3: *Algoritmo de las K-medias*

La condición de parada del algoritmo consiste en comparar la distorsión δ de la partición actual con la distorsión de la misma partición en la iteración anterior δ_{ant}. Si el cambio es menor que un ϵ prefijado el algoritmo acaba. En la ecuación 2.22 se muestra el cálculo de la distorsión δ,

$$\delta = \sum_{j=1}^{c} \delta_j \tag{2.22}$$

donde δ_j es la distorsión producida en cada *cluster* y se obtiene a partir de la ecuación 2.23

$$\delta_j = \sum_{\vec{x}_i \in P_j} \| \vec{x}_i - \vec{p}_j \|^2 \tag{2.23}$$

Tal y como funciona el algoritmo de las K-medias, los datos que son

similares, es decir, que están cerca geométricamente, pertenecerán al mismo grupo, de la misma forma es difícil que un dato lejano al prototipo del grupo forme parte de dicho grupo. No obstante presenta una serie de inconvenientes tales como:

- No detecta grupos vacíos o degradados.
- Puede acabar con varios prototipos idénticos.
- Como solución final encuentra el mínimo local más cercano a la partición inicial.

El inconveniente más serio es el último, ya que la solución final va a depender de la partición inicial y además no se garantiza que se llegue a la partición óptima.

Algoritmo de las k-medias difuso

Este algoritmo [Bezdek, 1981] es una generalización del algoritmo de las K-medias. La modificación que se realiza es relativa a los valores que las funciones de pertenencia pueden tomar. Mientras que en el algoritmo de las K-medias las funciones de pertenencia estaban definidas en el rango $\{0, 1\}$, ahora cada grupo se considera un conjunto difuso, por lo que las funciones de pertenencia están definidas en el rango $[0, 1]$. Esto implica que un ejemplo de entrada puede estar asignado a varios grupos con distinto grado de pertenencia. Dichos grados de pertenencia informan sobre la cercanía del vector de entrada a los prototipos de tales grupos.

En este algoritmo desaparece el inconveniente de crear grupos vacíos que presentaba el algoritmo de las K-medias, no obstante sigue manteniendo

los otros dos problemas: pueden aparecer grupos idénticos y el algoritmo termina en el mínimo local más cercano a la partición inicial. Hay que resaltar que el algoritmo de las K-medias difuso es más costoso en tiempo de ejecución.

Algoritmo ELBG (o *enhanced LBG*)

Con este algoritmo [Russo y Patanè, 1999] se pretenden solucionar los problemas que aparecen en con los algoritmos de las K-medias y de las K-medias difuso, en cuanto a que sólo alcanzan mínimos locales próximos al punto de partida.

Este algoritmo pretende determinar qué *clusters* están mal colocados y dónde se deberían mover para así escapar de la proximidad de un mínimo local y obtener una distribución de *clusters* independiente de la configuración inicial.

El algoritmo constituye una extensión del algoritmo de las K-medias basada en el teorema de *distorsión total*: "Cada grupo tiene una contribución igual a la distorsión total en una cuantización óptima de alta resolución".

Se puede definir el concepto de *utilidad* de un prototipo utilizando el teorema de distorsión total, tal y como muestra la ecuación 2.24.

$$u_j = \frac{\delta_j}{\bar{\delta}} \tag{2.24}$$

donde $\bar{\delta}$ representa la distorsión media de la partición (ecuación 2.25)

$$\bar{\delta} = \frac{1}{c}\sum_{i=1}^{c}\delta_i \tag{2.25}$$

El teorema de distorsión total implica que en una partición óptima de los vectores de entrada todos los prototipos tendrían la misma utilidad. De esta forma, el algoritmo ELBG trata de alcanzar esta condición moviendo los prototipos que tienen una utilidad menor que 1, hacia grupos con prototipos de utilidad mayor que 1.

Algoritmo de clustering difuso condicional

En este caso como algoritmo supervisado que es no sólo va a tener en cuenta las variables de entrada sino también la/s de salida.

El objetivo principal de este algoritmo [Pedrycz, 1996, 1998] es desarrollar *clusters* teniendo en cuenta las semejanzas en el espacio de entrada así como sus valores respectivos asumidos en el espacio de salida. Las bases de este algoritmo son las mismas que las del algoritmo de las K-medias difuso, por lo que se considera una extensión de éste. La parte condicional del mecanismo de agrupación reside en las variables de salida $y_1, y_2, \ldots, y_n$ de los correspondientes vectores de de entrada. De esta forma la variable de salida y_k del vector de entrada $\vec{x}_k$ describe el nivel en que este vector interviene en la construcción del *cluster*.

Algoritmo de estimación de grupos alternante (ACE)

En el resto de algoritmos de *clustering* analizados se realiza una determinación inicial de los parámetros que caracterizan los métodos. Por

ejemplo, la forma de las funciones de pertenencia, la actualización de los prototipos de los grupos, etc. El algoritmo de estimación de grupos alternante [Runkler y Bezdek, 1999] se presenta como una herramienta donde los parámetros anteriormente citados se pueden elegir directamente por el usuario. Para fijar los parámetros que definen el algoritmo de *clustering* se determinan los conjuntos de entrada y salida. De esta forma se pueden implementar los algoritmos vistos anteriormente con distintas configuraciones. En este caso las funciones de pertenencia se inspiran los sistemas neuro-difusos.

Algoritmo de clustering para aproximación de funciones

La mayoría de los métodos presentados anteriormente se diseñaron para la resolución de problemas de clasificación y no se adaptaban bien al problema de aproximación funcional. El método de *clustering* para la aproximación de funciones [González y otros, 2002] incorpora una serie de características que lo adecúan al problema de aproximación.

Este algoritmo mide la variabilidad de la salida de la función objetivo durante el proceso de *clustering* y aumenta el número de *clusters* en aquellas zonas de entrada donde la salida de la función sea más variable.

2.3.3. Algoritmos para la inicialización de los radios de las funciones base

Una vez que se han determinado los valores iniciales de los centros $\vec{c_i} = (c_p, \ldots, c_p)$ de las funciones de base radial, se han de determinar los radios r_i de dichas funciones. Los radios se deben establecer de tal manera que se

cubra convenientemente el espacio de entrada, que el solapamiento entre las zonas de activación de las RBFs sea lo menor posible y que todo el espacio de entrada quede cubierto.

Una alternativa clásica para inicializar los radios consiste en calcular el mismo valor para los radios de todas las funciones base. En [Park y Sandberg, 1991] [Park y Sandberg, 1993] se calcula el valor inicial como $r = d/\sqrt{2L}$, donde d es la distancia máxima entre los centros y L es el número de centros. A pesar de la simplicidad de este mecanismo se prueba que es suficiente para obtener un aproximador universal.

El comportamiento de la red puede mejorar si se establece un radio individual para cada función base, esto se demuestra en [Musavi y otros, 1992].

El radio de las funciones de base radial se puede establecer mediante diferentes heurísticas. Entre las más utilizadas están:

- Heurística de los k vecinos más cercanos (KNN). Fija el valor del radio de cada función base a un valor igual a la distancia media a los centros de las k funciones base más cercanas [Moody y Darken, 1989]. Cuando $k = 1$ se conoce como heurística del vecino más cercano.

- Heurística de la distancia media de los vectores de entrada más cercanos (CIV). En este caso, el radio de la i-ésima RBF se determina se acuerdo con la ecuación 2.26 [Karayiannis y Mi, 1997],

$$r_j = \frac{\sum_{\vec{x}_i \in C_j} d(\vec{c_j}, \vec{x_i})}{|C_j|} \tag{2.26}$$

donde C_j es el conjunto de vectores de entrada que están más cerca de $\vec{c_j}$ que de cualquier otro centro, $|C_j|$ es el número de vectores que componen dicho conjunto y $d(\vec{c_j}, \vec{x_i})$ es la distancia entre el centro $\vec{c_j}$ y el patrón de entrada $\vec{x_i}$. Esta heurística produce menor solapamiento que la anterior aunque $k = 1$.

2.3.4. Algoritmos incrementales y decrementales

Tal y como se mencionó con anterioridad, cuando se aborda el diseño de RNAs en general, y por tanto el diseño de RBFNs [Sanchez, 2002] en particular, hay que determinar la topología y los parámetros óptimos.

Los algoritmos incrementales y decrementales, también conocidos como métodos secuenciales, tratan de determinar el número óptimo de RBFs que debe tener la red.

Un método es incremental o constructivo cuando comienza con una red pequeña, con pocas o una RBF, y va añadiendo RBFs hasta que logra una buena solución. Por el contrario, el método decremental (destructivo o de poda) parte de una red compleja y va eliminando RBFs que considera innecesarias de forma que se llega a una red menos compleja y que proporciona una buena solución. El principal problema que presentan estos métodos es que pueden quedar atrapados en mínimos locales.

Uno de los algoritmos más utilizados es el algoritmo *RAN (Resource Allocation Network)* [Plat, 1991] y algunas de sus modificaciones. El método RAN intenta mejorar los diseños de RBFNs que se hacían en la época. Es un método incremental, va a partir de un número bajo de RBFs y utiliza información local del entorno para ir añadiendo RBFs de forma que se

construya una red capaz de solucionar el problema. La información local que el algoritmo analiza cíclicamente son las muestras o patrones del conjunto de entrenamiento. Dada una muestra, se introducirá una nueva RBF centrada en dicha muestra, cuando se cumplan las dos condiciones expresadas en 2.27 y 2.29:

1. Que la distancia del patrón de entrada al centro de la RBF más cercana esté por encima de un umbral,

$$\| \vec{x}_k - \vec{c}_n \| > \delta(t) \tag{2.27}$$

donde $\vec{x}_n$ es el vector de entrada actual, $\vec{c}_n$ es el centro de la RBF ϕ_n más cercana al vector de entrada según la distancia euclídea $\| \|$. $\delta(t)$ es un umbral de resolución variable respecto al número de patrones que se han presentado, donde t indica el número de patrón actual. Inicialmente el umbral toma el valor máximo que pueda tomar $\delta(t) = \delta_{max}$, que puede ser igual al tamaño del espacio de entrada. Su valor va disminuyendo hasta que se alcanza un valor δ_{min}. Para ir disminuyendo el valor se utiliza la expresión 2.28

$$\delta(t) = max[\delta_{max} exp(-t/\tau), \delta_{min}] \tag{2.28}$$

siendo τ una constante.

2. Que la diferencia entre la salida de la red $f(\vec{x}_k)$ y el valor objetivo dado por el patrón y_k sea superior a un valor ϵ que representa la precisión de la salida de la red.

$$|y_k - f(\vec{x}_k)| > \epsilon \tag{2.29}$$

Si no se cumplen las dos condiciones anteriores, entonces se utilizan algoritmos de gradiente para ajustar los pesos y los centros de la red. El sistema empieza con una red con mucho error y va reduciendo éste añadiendo RBFs.

La eficiencia de este método se mejora en *RANEKF* [Kadirkamanathan, 1993] que utiliza un método de minimización denominado *Filtro de Kalman Extendido, EKF*, en vez de algoritmos de gradiente para ajustar los parámetros de la red. Este método obtiene mejores resultados que RAN en aproximación de funciones y predicción de series temporales.

Otro enfoque parecido es el propuesto por el algoritmo *Growing Radial Basis Networks, GRBN* [Karayiannis y Mi, 1997]. En este algoritmo se plantea un mecanismo en el que se añaden más funciones bases en aquellas zonas en donde existen funciones base que cometen mucho error. Esto implica que la complejidad de la red va creciendo hasta que se cumpla una condición de parada que evalúa el compromiso entre el error de aproximación y la complejidad de la red.

Los métodos anteriores expuestos van añadiendo funciones base pero no eliminan ninguna y puede ser interesante eliminar aquellas que aporten poco para lograr una buena salida de la red.

El algoritmo *Minimal RAN, M-RAN* [Yingwei y otros, 1997] incorpora un mecanismo de poda de aquellas RBFs que contribuyen poco a la salida de la red. A la hora de introducir RBFs se tiene que cumplir que el error cuadrático medio para un grupo de patrones ha de ser superior a un valor mínimo.

En [Rosipal y otros, 1998] también se presenta una modificación del

algoritmo RAN, donde se incluye una descomposición QR de Givens *RAN-GQRD* para calcular los pesos de la red y también realiza una poda de funciones base *RAN-P-GQRD* de forma que se reduce la complejidad final de la red.

Otra posibilidad para implementar un mecanismo de poda en el algoritmo RAN, consiste en usar la descomposición SVD de la matriz de activación de la red, junto con la transformación QR-CP para determinar cuáles son las funciones bases menos determinantes [Salmerón y otros, 2001].

En [Rojas y otros, 2002] se presenta el algoritmo *PG-RBF Sequential Learning Algorithm* en que se emplean hasta tres criterios para identificar funciones base que no contribuyen a la salida de la red.

Otro tipo de funciones que se pueden utilizar para determinar RBFs poco importantes son los métodos OLS o SVD, e incluso hibridaciones de OLS con algoritmos evolutivos.

En [Chen y otros, 1990] se usa el criterio de información de Akaike para determinar el número de RBFs en la capa oculta, estableciendo un compromiso entre la complejidad de la red y su eficiencia (va incrementando o decrementando el número de RBFs).

Otros algoritmos con este enfoque secuencial (incremental o decremental) se encuentran en [Holcomb y Morari, 1991; Lee y Kil, 1991; Musavi y otros, 1992; Sundararajan y otros, 1999; Peng y otros, 2006].

2.3.5. Diseño evolutivo

Como ya se vio en el capítulo 1 un paradigma importante de diseño de redes neuronales artificiales es el evolutivo y sus variantes co-evolutivo y cooperativo-competitivo. Estos paradigmas se han utilizado también en el caso particular de las RBFNs.

Concretamente, la determinación evolutiva de los parámetros óptimos de una RBFN se aborda en trabajos tales como [Harpham y otros, 2004; Buchtala y otros, 2005]. La evolución de RBFNs se centra en tres grandes áreas [Yao, 1999; Harpham y otros, 2004]:

- Evolución de la arquitectura de la RBFN. Dicha evolución implica obtener el número de RBFs necesarias. Este problema ha sido abordado usualmente, junto con la evolución del resto de los parámetros, en aproximaciones tales como [Burdsall y Giraud-Carrier, 1997; Sergeev y otros, 1998; Xue y Watton, 1998].

- Evolución de los parámetros de la RBFN, centros, radios y pesos de las RBFs. El uso de algoritmos evolutivos para optimizar los pesos de las conexiones puede eliminar la posibilidad de converger a un mínimo local pero no es usual utilizar un AE para optimizar dicho parámetro de forma aislada sino junto con otros parámetros de la RBF [Sumathi y otros, 2001; Sheta y De Jong, 2001; Jiang y otros, 2003]. En [Vesin y Gruter, 1999; Dawson y otros, 2000] se pueden encontrar propuestas que evolucionan sólo el centro y el radio de las funciones base.

- Optimización del conjunto de datos. En estos casos lo que se intenta es seleccionar un subconjunto del conjunto de datos de entrenamiento

de forma que se reduzca la dimensionalidad del aprendizaje [Billings y Zheng, 1995; Sergeev y otros, 1998]. Por otro lado, la selección de los atributos más relevantes para el diseño de la RBFN no está muy estudiado en la bibliografía especializada [Fu y Wang, 2003] y se puede realizar mediante AEs [Fu y Wang, 2002; Ferreira y otros, 2003].

En el diseño evolutivo de las RBFNs la mayoría de las propuestas presentan algoritmos híbridos donde el AE optimiza los centros y los radios de las RBFs y en una segunda fase usan métodos numéricos para calcular los pesos de las conexiones [Chaiyaratana y Zalzala, 1998; Sergeev y otros, 1998; Xue y Watton, 1998; Vesin y Gruter, 1999; Moechtar y otros, 1999; Chen y otros, 1999; Dawson y otros, 2000].

En cualquier algoritmo evolutivo y por tanto en el diseño evolutivo de RBFNs, hay que considerar dos aspectos:

- La forma en que se representan y manejan las soluciones.
- La forma de calcular la bondad de una solución candidata.

Si nos centramos en la representación, la mayoría de las propuestas de diseño evolutivo de RFBNs codifican en un individuo una RBFN completa y la población de RBFNs evoluciona mediante un conjunto de operaciones [Harpham y otros, 2004; Rivas y otros, 2004; Lacerda y otros, 2005]. Dicho esquema de representación es conocido como enfoque *Pittsburgh*. Otros trabajos que también utilizan dicho esquema de representación son: con codificación binaria [Sergeev y otros, 1998; Vesin y Gruter, 1999; Moechtar y otros, 1999; Dawson y otros, 2000; Sumathi y otros, 2001; Du y Zhang, 2008],

con codificación entera [Billings y Zheng, 1995] o mediante codificación real [Leung y otros, 2002; Esposito y otros, 2000a].

No obstante, y según Potter [Potter y De Jong, 2000] el diseño evolutivo presenta algunas dificultades en la resolución de cierto tipo de problemas, especialmente cuando un individuo representa una solución completa compuesta de subcomponentes independientes. Una alternativa que se presenta al esquema clásico *Pittsburgh* es la estrategia cooperativa-competitiva [Whitehead y Choate, 1996] y la co-evolutiva [Potter y De Jong, 2000], que ofrecen un marco donde cada individuo de la población representa sólo una parte de la solución, los individuos evolucionan en una o varias poblaciones y entre todos forman la solución global. Estas estrategias son computacionalmente menos complejas, dado que un individuo no representa una solución completa sino una parte de ésta. Los dos problemas que se han de resolver para aplicar dichas estrategias son la asignación de crédito (*fitness*) que debe valorar la contribución de cada individuo para la solución global y el mecanismo que se utilice para garantizar la diversidad de los individuos en la población.

En la bibliografía existen algunas propuestas que hacen referencia a estas estrategias para el diseño de RBFNs [Whitehead y Choate, 1996; Topchy y otros, 1997; Li y otros, 2008].

La más tradicional es la de [Whitehead y Choate, 1996] donde un individuo representa una RBF y la población la red completa. La asignación de crédito para un individuo se calcula en función del peso de la RBF.

De la misma forma en [Topchy y otros, 1997] se describe un algoritmo en el que cada individuo es una RBF y la asignación de crédito es calculada

de acuerdo con la eficiencia o la aportación de la RBF a la salida de la red.

Otra propuesta de algoritmo cooperativo-competitivo se presenta en [Rivera y otros, 2007] donde la asignación de crédito de los individuos, RBFs, se define a partir de tres parámetros y usa un sistema basado en reglas difusas para decidir qué operadores aplicar a los individuos.

En [Li y otros, 2008] se presenta un método de diseño co-evolutivo, en este caso existen un conjunto de sub-poblaciones que interactúan y evolucionan. Cada individuo de la población representa un componente particular (grupo de RBFs) de la RBFN. En este caso la asignación de crédito de un individuo de una sub-población se calcula de acuerdo con la representatividad de la sub-población frente a las otras.

Si en el cálculo de la calidad de las soluciones candidatas, se maneja un sólo objetivo (por ejemplo el error) para optimizar las RBFNs, puede ocurrir que se obtengan redes muy complejas, con un alto número de RBFs. Esto ocurre porque es más fácil reducir el error con redes que tienen más neuronas que con aquellas que tienen menos. Para intentar optimizar varios objetivos, se pueden utilizar los algoritmos evolutivos multi-objetivo [Deb, 2001], así dichos algoritmos permiten optimizar simultáneamente múltiples objetivos tales como error y complejidad, por ejemplo. En este tipo de algoritmos evolutivos todos los objetivos que se deban de optimizar se han de tener en cuenta a la hora de calcular el *fitness* y las relaciones entere los individuos. El objetivo ahora no es encontrar un única solución óptima, sino obtener un conjunto de soluciones óptimas (soluciones no-dominadas) con calidades similares. Existen diferentes algoritmos evolutivos multi-objetivo aplicados al diseño de RBFNs tal y como los propuestos en [González y otros, 2003; Yen, 2006; Guillén y otros, 2007; Teixeira y otros, 2008].

Todas las propuestas multi-objetivo para el diseño de RBFNs usan un esquema de codificación *Pittsburgh* debido a la dificultad que presenta la comparación de soluciones parciales, tal y como se requiere en los enfoques co-evolutivos y cooperativos-competitivos.

2.4. Conclusiones

En el capítulo anterior se describieron características generales de las redes neuronales y en éste el estudio se ha centrado en las Redes de Función de Base Radial. Se han descrito las RBFNs que son las que utilizan las neuronas de la capa oculta de este tipo de redes y a las que se debe su nombre. A continuación se ha explicado la arquitectura típica de este tipo de redes.

Para el desarrollo de un modelo es necesario realizar un estudio de otros modelos ya desarrollados, por lo que se ha realizado un estudio bibliográfico de los métodos más utilizados para el diseño de RBFNs. Entre los más importantes se han destacado los numéricos, los de *clustering*, los incrementales/decrementales y los evolutivos.

Dentro del diseño evolutivo, que es el que se va a utilizar en el trabajo de esta memoria, la mayoría de las propuestas codifican en un individuo una RBFN completa y la población de RBFNs evolucionan mediante un conjunto de operaciones (enfoque *Pittsburgh*). Son menos frecuentes las propuestas donde cada individuo de la población representa sólo una parte de la solución, los individuos evolucionan en una o varias poblaciones y entre todos forman la solución global (enfoques co-evolutivos y cooperativos-competitivos). Éstos últimos son computacionalmente menos complejos que

los de tipo *Pittsburgh*, dado que un individuo no representan una solución completa sino una parte de ésta.

Capítulo 3

CO^2RBFN: algoritmo evolutivo cooperativo-competitivo para el diseño de Redes de Funciones de Base Radial

En este capítulo se presenta el algoritmo, CO^2RBFN, un algoritmo evolutivo, con un enfoque cooperativo-competitivo, para el diseño de RBFNs, aplicado a resolver el problema de clasificación de patrones. En el algoritmo, se hibridan diferentes paradigmas *soft-computing*, como son las redes neuronales, los algoritmos evolutivos y la lógica difusa. Esta última se utiliza porque el algoritmo implementa un sistema basado en reglas difusas para representar conocimiento experto que permita decidir qué operadores aplicar a los individuos.

En el entorno cooperativo-competitivo propuesto, cada individuo es una función base y la población entera es responsable de la solución final. Para medir la asignación de crédito de los individuos o RBFs se consideran tres factores: aportación a la salida de la red, error y solapamiento con otras RBFs.

El capítulo comienza con una descripción del problema de clasificación, después se analizan problemas a abordar en el diseño evolutivo de RBFNs y se estudian diferentes métricas para determinar una medida que sea sencilla de implementar y con la que se pierda la mínima cantidad de información en problemas de clasificación con variables nominales. Una vez vistos estos aspectos preliminares se pasa a detallar el algoritmo.

Una vez presentado el algoritmo, se aplicará a la resolución de problemas de clasificación. La precisión y la complejidad de la red obtenida por el modelo se comparará con los resultados obtenidos mediante otros métodos.

3.1. El problema de clasificación

En clasificación el atributo de salida es conocido como *clase* y por tanto, el objetivo es aprender un modelo, a partir de los ejemplos, que sea capaz de establecer la clase asociada a un conjunto de valores de los atributos de entrada. Desde un punto de vista formal, el objetivo es aprender una función

$$F : \vec{e} \longrightarrow dom(y) \tag{3.1}$$

denominada clasificador, que represente la correspondencia entre los atributos de entrada y la clase. En este caso y es discreto y sólo puede

tomar valores o clases dentro de un conjunto finito, $dom(y) = c_1, c_2, \ldots, c_m$. La función aprendida debe ser capaz de etiquetar los ejemplos no etiquetados que se le presenten, es decir, dará un valor de y para cada valor de $\vec{e}$ presentado [Maimon y Rokach, 2005]. En la figura 3.1 se muestra un ejemplo de un problema de clasificación con tres clases distintas.

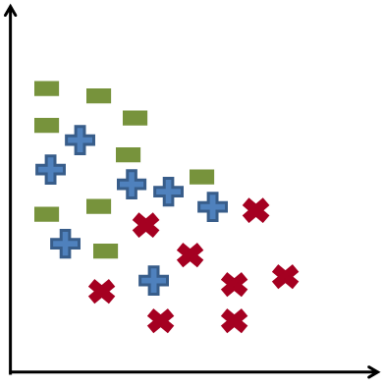


Figura 3.1: *Ejemplo de problema de clasificación con datos pertenecientes a tres clases*

El escenario anterior es válido para un problema de clasificación clásica, en el que las clases son mutuamente excluyentes, es decir, dado un ejemplo sólo puede pertenecer a una clase. En problemas de multi-clasificación puede ocurrir que haya varias clases disjuntas. Nos vamos a centrar en aquellos problemas en los que las clases son excluyentes.

Diferentes técnicas se pueden aplicar a esta tarea de clasificación con algoritmos de aprendizaje capaces de identificar el modelo que se ajuste mejor a las relaciones entre el atributo de la clase y los atributos de entrada. Una vez que se obtiene el modelo aprendido, éste se puede utilizar para predecir la clase de ejemplos no vistos.

Un factor clave en los algoritmos de aprendizaje es construir modelos que tengan una buena *capacidad de generalización*, es decir, modelos que

predigan con precisión la clase para los ejemplos no vistos que se le presenten. En resumen, el algoritmo debe aprender mediante los ejemplos del conjunto de entrenamiento un buen modelo que sea capaz de etiquetar con la clase a los ejemplos del conjunto de test.

Para medir la eficiencia de un modelo de clasificación se ha de hacer un recuento del número de ejemplos bien y mal clasificados del conjunto de test. Este dato con frecuencia se recoge en una tabla conocida como *matriz de confusión* [Chawla y otros, 2002]. En la tabla 3.1 se muestra la matriz de confusión para un problema de clasificación binario, existen sólo dos clases a las que vamos a llamar P y N, clases Positiva y Negativa.

Tabla 3.1: *Matriz de confusión para un problema con dos clases*

		Clase Predicha	
		Clase P	Clase N
Clase Real	Clase P	VP	FN
	Clase N	FP	VN

En esta tabla los elementos representan:

- VP (Verdaderos Positivos): son el número de ejemplos de la clase positiva que han sido correctamente clasificados.
- VN (Verdaderos Negativos): son el número de ejemplos de la clase negativa que han sido correctamente clasificados.
- FP (Falsos Positivos): son el número de ejemplos de la clase positiva que han sido clasificados incorrectamente.
- FN (Falsos Negativos): son el número de ejemplos de la clase negativa que han sido incorrectamente clasificados.

Las medidas más utilizadas y que permiten comparar modelos entre sí son la *precisión* o *tasa de aciertos* y la *tasa de error*, definidas en las ecuaciones 3.2 y 3.3, sobre un conjunto de ejemplos .

$$Precisión = \frac{N^o \ de \ ejemplos \ correctamete \ clasificados}{N^o \ total \ de \ ejemplos \ clasificados} \tag{3.2}$$

$$Tasa \ de \ Error = \frac{N^o \ ejemplos \ mal \ clasificados}{N^o \ total \ de \ ejemplos \ clasificados} \tag{3.3}$$

De esta forma, en el ejemplo anterior de las dos clases se tendría

$$Precisión = \frac{VP + VN}{VP + FN + FP + VN} = 1 - Tasa \ de \ Error$$

y

$$Tasa \ de \ Error = \frac{FP + FN}{VP + FN + FP + VN}$$

Los algoritmos en clasificación buscan modelos que obtengan la mayor precisión, o lo que es lo mismo el menor error, cuando se aplican al conjunto de test.

Un modelo será bueno no sólo cuando obtenga buenos resultados sobre el conjunto de entrenamiento sino cuando también obtenga buenos resultados sobre el conjunto de test. A medida que el modelo se va haciendo más complejo, su precisión va aumentando, pero puede ocurrir que a partir de un punto su precisión no mejore sino que incluso empiece a empeorar. En esta situación se dice que existe un problema de *sobre-aprendizaje* (*overfitting*), el modelo clasifica muy bien el conjunto de entrenamiento pero no generalizan bien en el conjunto de test. Cuando el modelo no es suficientemente complejo puede presentar una precisión baja tanto en el conjunto de entrenamiento

como en el de test, se dice que presenta *bajo-aprendizaje* (*underfitting*).

Algunas de las causas potenciales que hacen que aparezcan los problemas vistos anteriormente son:

- presencia de ruido debido a excepciones que aparecen en los ejemplos del conjunto de entrenamiento, ya que entran en contradicción con otros ejemplos,

- que en el conjunto de entrenamiento haya ejemplos pocos significativos lo cual causará que el modelo tenga poca capacidad de generalización.

En vista de lo anterior, si el modelo que se obtiene es poco complejo se pueden tener problemas de bajo-entrenamiento, si por el contrario es demasiado complejo puede existir sobre-entrenamiento. La complejidad ideal del modelo sería aquella con la que consiga una buena generalización, pero durante la etapa de aprendizaje del modelo no se tiene acceso a los ejemplos del conjunto de test. Lo que se puede hacer es tratar de estimar la capacidad de generalización usando los ejemplos del conjunto de entrenamiento.

El objetivo es obtener modelos que tengan una buena capacidad de generalización, para ayudar en esta tarea se usará el método de validación cruzada con 10 particiones visto en el capítulo 1 que permite estimar la capacidad del modelo, de forma que tengamos una idea de su comportamiento frente a datos desconocidos.

3.2. Preliminares para el diseño del modelo

Las RBFNs, tal y como se vio en el capítulo anterior, presentan características que hacen que sean un modelo interesante para la resolución de problemas:

- Tienen una topología simple, con solo una capa oculta.
- Las RBFs tienen una respuesta local que depende de su centro y su radio.
- Son uno de los pocos modelos de RNAs interpretables y ofrecen la posibilidad de extraer reglas a partir de ellas.
- Poseen una capacidad de aproximación universal.

El modelo que se propone en esta tesis es un modelo de diseño para este tipo de redes. El objetivo en el proceso de diseño de cualquier RBFN, puede ser determinar el número (m) de RBFs y para cada una de ellas el tipo (ϕ_i) de función y sus parámetros, centros ($\vec{c}_i$) y los radios (r_i) así como los pesos óptimos ($\vec{w}_i$) que conectan las RBFs con las neuronas de la capa de salida. Se vio en el capítulo anterior que los algoritmos de diseño no siempre determinan todos los parámetros anteriores, gran parte de ellos se centraban en determinar los pesos, centros y radios para una topología dada.

En el método que se presenta, se considera que el tipo de RBF es una gaussiana, el número de RBFs se fija a priori, por lo que el algoritmo de diseño se ocupará de determinar los centros, los radios y los pesos. El diseño se va a realizar en dos etapas: determinación de los centros y radios mediante un enfoque evolutivo y cálculo de los pesos mediante la técnica LMS.

Las RBFNs son combinaciones lineales de múltiples funciones locales no lineales (las RBFs). Se puede decir que aproximan relaciones complejas a partir de aproximaciones locales menos complejas de forma análoga a la división de un problema en varios sub-problemas más simples [Park y Sandberg, 1991]. Dada esta característica de respuesta local de las RBFs, se considera que una buena opción es elegir un enfoque cooperativo-competitivo en el diseño del algoritmo evolutivo, ya que en este paradigma los individuos compiten por representar su parte del entorno y colaboran entre ellos para alcanzar la solución final.

El objetivo del modelo propuesto es obtener RBFNs simples, precisas y que generalicen bien. Es decir, redes que con pocas RBFs sean capaces de cubrir el espacio del problema, con un mínimo de solapamiento entre ellas y que dichas redes sean capaces de proporcionar una respuesta adecuada ante cada nueva entrada que se les presente.

Teniendo en mente los objetivos anteriores se diseñan sus componentes:

- Una función *fitness* (asignación de crédito) que considera tres factores: aportación de la RBF a la salida global de la red, error local cometido por la RBF y solapamiento de la RBF con otras. Con esto se pretende obtener RBFs importantes para la salida de la red, que cometan poco error y con el mínimo solapamiento entre ellas.

- Diseño de unos operadores evolutivos específicos que analizan el entorno de las RBFs.

- Uso de una medida de distancia que, con la mínima pérdida de información, se pueda aplicar tanto a valores numéricos como nominales.

- Utilización de un SBRD para decidir en base a conocimiento experto qué operador aplicar a una RBF en un momento dado.

El diseño se va a optimizar para resolver problemas de clasificación. En un entorno de clasificación, la RBFN tiene que realizar la traslación desde un espacio de entrada X^n a un conjunto finito de clases C con k clases, de la forma indicada en la ecuación 3.4:

$$S = \left\{(\vec{x}_u, y_u) | \vec{x}_u \in X^n,\ y_u \in C,\ u = 1. \ldots, p\right\} \tag{3.4}$$

donde $\vec{x}_u$ es el vector de características (patrón) e y_u es la clase a la que pertenece.

Normalmente, en un entorno de clasificación se suelen considerar los siguientes aspectos:

- El número de salidas de la RBFN se corresponde con el número de clases ($s = k$) y cada clase y_u se asigna a un nodo de salida (figura 3.2).
- Para entrenar la red, la pertenencia de la clase y_u se codifica como un vector binario $\vec{z}_u \in \{0, 1\}^k$ con la relación $\vec{z}_u^i = 1$ si y sólo si $y_u = i$, y $\vec{z}_u^i = 0$ en otro caso.
- Después del entrenamiento, la salida del nodo j, $f_j(\vec{x}_u)$ para un vector $\vec{x}_u$ se puede interpretar como la probabilidad de pertenencia a la clase que representa dicho nodo de salida. Usualmente, la clase que la red da como salida es la correspondiente al nodo de salida que presenta mayor nivel de activación.

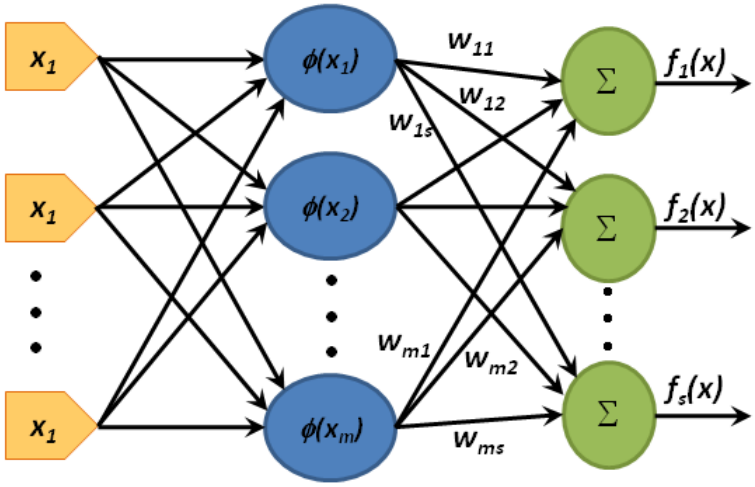


Figura 3.2: *Arquitectura típica de una RBFN*

- Cada RBF ϕ_i cubre un conjunto de patrones de la misma clase y_u. Si los pesos de la red se han entrenado correctamente, la RBF ϕ_i tendrá un valor alto de activación para los patrones de entrada que pertenezcan a la clase y_u, y de esta forma el patrón será bien clasificado por la red.

Los aspectos anteriores junto con las características mencionadas de topología simple y respuesta localizada, confieren a las RBFNs un grado de interpretabilidad mayor que el de otro tipo de RNAs, como se vio en el capítulo 2. También se vio que para poder extraer reglas interpretables, el número de RBFs en la red no podía elevado y éstas no deberían ser muy similares entre sí. En el algoritmo que se propone se intentan optimizar estas dos características.

El principal problema en este tipo de redes se produce en bases de datos con la alta dimensionalidad. Cuando la dimensionalidad del espacio de entrada es alta, el número de neuronas de la capa de entrada tiende ser alto,

esto provoca que el número de neuronas en la capa oculta, RBFs, necesarias para tener una buena generalización, crezca de forma exponencial.

El algoritmo propuesto, se ha probado también en bases de datos con alta dimensionalidad.

3.3. Métrica utilizada para los atributos numéricos y nominales

Un problema típico al que se enfrentan los modelos de clasificación consiste en trabajar con bases de datos que no sólo contengan atributos numéricos (reales o enteros) sino también que tengan atributos nominales. Una de las características clave de cualquier modelo de diseño de RBFNs es la evaluación de los patrones, lo que implica el cálculo de distancias entre patrones y los centros de las RBFs. Cuando se trata de atributos numéricos es habitual la utilización de la distancia Euclídea, con la que se consiguen resultados satisfactorios. Esta métrica sólo está definida para los atributos numéricos por lo que se necesita otra medida diferente para trabajar con los atributos nominales. Definir una medida de similaridad para atributos nominales es menos trivial [Esposito y otros, 2000b] y básico en un método adecuado de diseño de RBFNs.

Una medida simple, pero comúnmente usada, es la *Overlap Metric (OM)* [Stanfill y Waltz, 1986], también conocida como distancia *Hamming* en el caso de atributos binarios. En el entorno de esta métrica, para dos posibles valores, la distancia es cero cuando los valores son idénticos y uno en otro caso. Esta medida, por tanto, implica una pérdida de información dado que

considera que todos los valores que son diferentes están a la misma distancia, y no tiene en cuenta diferentes grados a la hora de medir las diferencias.

Existen medidas alternativas como son la distancia *VDM (Value Difference Metric)* [Wilson y Martinez, 1997], y la distancia *ADM (Adaptive Dissimilarity Matrix)* [Cheng y otros, 2004], entre otras. Para la distancia VDM dos valores están cercanos si tienen clasificaciones similares (es decir, sus clases de salida siguen correlaciones similares). Por otro lado, la distancia ADM tiene en cuenta la posible correlación entre atributos. En [Cheng y otros, 2004] un modelo de RBFN se usa para analizar la eficiencia de las medias OM, VDM y ADM y demuestra que las medidas VDM y ADM superan a la medida OM. Sin embargo, comparando los resultados alcanzados por VDM y ADM no se puede sacar una conclusión sobre cuál de las dos es más eficiente.

En el método que se propone se va a utilizar la distancia VDM para tratar los atributos nominales, dado que desde un punto de vista computacional es más simple que ADM e igual de eficiente. Por otro lado se han de procesar también atributos numéricos, por lo que la distancia elegida para tratar ambos tipos de atributos es la HVDM [Wilson y Martinez, 1997], que usa la distancia euclídea para los atributos numéricos y la VDM para los nominales tal y como se muestra en la ecuación 3.5:

$$HVDM(x,y) = \sqrt{\sum_{a=1}^{n} d_a^2(x,y)} \tag{3.5}$$

donde n es el número de atributos o características de los patrones, y la función $d_a(x,y)$ devuelve la distancia entre dos valores x e y para el atributo

a según la ecuación 3.6

$$d_a(x,y) = \begin{cases} 1 & \text{si x ó y son desconocidos} \\ VDM_a(x,y) & \text{si a es nominal} \\ E_a(x,y) & \text{si a es numérico} \end{cases} \tag{3.6}$$

$E_a(x,y)$ es la conocida distancia euclídea, en este caso para el atributo a (ecuación 3.7):

$$E_a(x,y) = \sqrt{\sum_{a=1}^{n}(x_a - y_a)^2} \tag{3.7}$$

La distancia VDM, apropiada para los atributos nominales, se muestra en la ecuación 3.8:

$$VDM_a(x,y) = \sum_{c=1}^{k}\left|\frac{N_{a,x,c}}{N_{a,x}} - \frac{N_{a,y,c}}{N_{a,y}}\right|^q = \sum_{c=1}^{k}\left|P_{a,x,c} - P_{a,y,c}\right|^q \tag{3.8}$$

donde

- $N_{a,x}$ es el número de instancias en el conjunto de entrenamiento que tiene valor x en el atributo a;
- $N_{a,x,c}$ es el número de instancias en el conjunto de entrenamiento que tienen valor x en el atributo a y c como clase de salida;
- k es el número de clases de salida en el dominio del problema;
- q es una constante (se usa como valor $q = 1$);

- $P_{a,x,c}$ es la probabilidad condicional de que la clase de salida sea c, siendo x el valor del atributo a, es decir, $P(c/a)$. $P_{a,x,c}$ se define como se indica en la ecuación 3.9:

$$P_{a,x,c} = \frac{N_{a,x,c}}{N_{a,x}} \tag{3.9}$$

3.4. Algoritmo cooperativo-competitivo para el diseño de redes de funciones de base radial: CO²RBFN

El algoritmo CO²RBFN (Algoritmo Cooperativo Competitivo para el diseño de RBFNs), es un algoritmo híbrido (figura 3.3) que combina diferentes técnicas *soft-computing* tales como redes neuronales, programación evolutiva con un enfoque cooperativo-competitivo y sistemas basados en reglas difusas. La estrategia evolutiva permite determinar los parámetros de las RBFs y utiliza un SBRD para determinar el operador genético a aplicar a una determinada RBF.

Como se ha mencionado anteriormente, el algoritmo tiene como objetivo diseñar RBFNs simples y precisas, es decir, redes compuestas por pocas RBFs (con bajo grado de solapamiento) que sean capaz de representar el conocimiento de los patrones y trabajen juntas para obtener una red con un adecuado nivel de generalización.

Con este objetivo en mente, la propuesta sigue una estrategia cooperativa-competitiva en la que cada individuo de la población representa una RBF (en este caso es una función gaussiana) y la población entera es la responsable

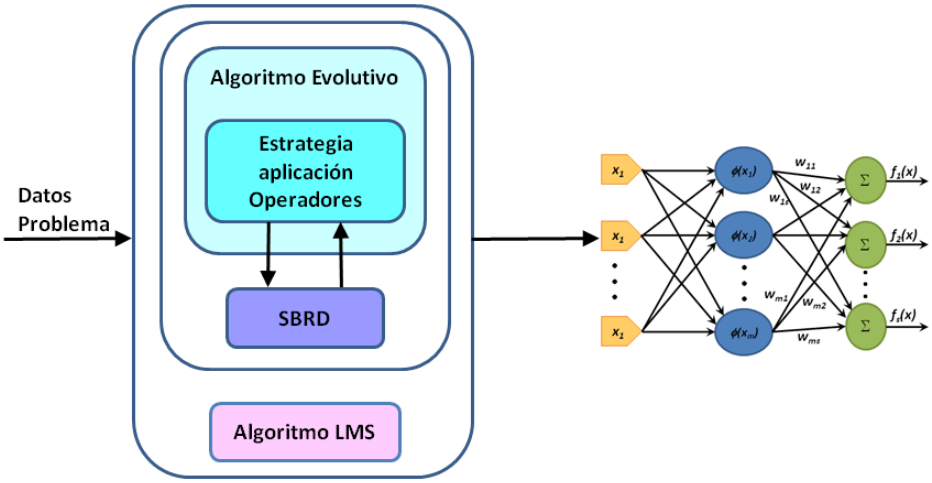


Figura 3.3: *Arquitectura de CO^2RBFN*

de la solución final. Este paradigma ofrece un marco en el que un individuo representa sólo una parte de la solución, de forma que los individuos cooperan para alcanzar una buena solución (una RBFN que generalice bien para nuevos patrones), pero también compiten por su supervivencia, dado que los individuos con peor comportamiento serán eliminados de la población. Gracias a este escenario (de cooperación-competición), se refuerza la explotación por zonas (RBFs con respuesta local), que la mayoría de los ejemplos estén representados (mediante alguna RBF) y se minimiza el solapamiento entre RBFs. Esta guía de diseño en el algoritmo propuesto mejora la interpretabilidad de la RBFN obtenida.

El modelo evolutivo de la propuesta tiene las siguientes características:

- el algoritmo utiliza un esquema de codificación real, en el que cada cromosoma representa el centro (con un número de coordenadas igual al número de características de los ejemplos) y el radio de una RBF;

- el operador de mutación, en sus dos variantes, es la única fuente de modificación de las soluciones;
- no existe operador genético de cruce;
- como estrategia de reemplazamiento, se compara el padre con el hijo y se selecciona para la siguiente generación al que tenga un mejor comportamiento en la red.

Después de elegir el enfoque evolutivo y algunos de sus componentes, se diseña el resto: *fitness*, operadores evolutivos, medida de distancia y SBRD, todos con el fin de lograr el objetivo propuesto.

En el entorno cooperativo-competitivo, como ya se mencionó, el fitness de cada individuo es conocido como asignación de crédito. Para medir la asignación de crédito de un individuo, se proponen tres factores que evalúan el comportamiento de cada RBF en la red. Dichos factores contemplan la aportación de la RBF a la salida global de la red, el error cometido por la RBF y su posible solapamiento con otras RBFs. En este sentido, nuestra función fitness combina conceptos tales como cooperación, especialización y niching [Buchtala y otros, 2005].

Se definen cuatro operadores evolutivos que van a poder ser aplicados a una RBF: un operador que elimina una RBF, dos operadores de mutación, y finalmente un operador que mantiene los parámetros de la RBF. Con dichos operadores se intenta lograr un adecuado equilibrio entre explotación y exploración del espacio de búsqueda y también se preservan las mejores RBFs.

Para decidir la probabilidad de aplicación de los operadores sobre una

determinada RBF, el algoritmo usa un SBRD que representa conocimiento experto en el diseño de RBFNs. Los factores propuestos para la asignación de crédito se usan como parámetros de entrada para el SBRD y las salidas determinan la probabilidad de aplicación de cada uno de los operadores.

Finalmente, es importante resaltar de nuevo que el algoritmo propuesto utiliza como medida la distancia HVDM que permite, con la mínima pérdida de información, trabajar tanto con atributos numéricos como nominales, usuales ambos en problemas de clasificación.

Se intenta que las modificaciones de la red a través las acciones del algoritmo no sean bruscas de tal forma que se mantenga una conducta o línea evolutiva a lo largo de la ejecución del algoritmo [Yao, 1999]. También se va a permitir cierto solapamiento entre las RBFs de forma que la salida de la red sea más suave y continua [Musavi y otros, 1992].

Todos los componentes mencionados consiguen que el algoritmo, CO²RBFN, diseñe RBFNs con un adecuado equilibrio entre precisión y simplicidad.

Los principales pasos del algoritmo se muestran en el pseudocódigo de la figura 3.4.

```
Inicialización de la RBFN
Mientras (No Fin) Hacer
    Entrenamiento de la RBFN
    Evaluación de las RBFs
    Aplicación de los operadores a las RBFs
    Introducción de nuevas RBFs
Fin_Mientras
```

Figura 3.4: *Principales pasos de CO² RBFN*

A continuación se describen detalladamente cada uno de los componentes de CO^{2}RBFN.

3.4.1. Inicialización de la red

Como se ha comentado anteriormente, el tamaño de la población se fija a priori, de forma que el número de neuronas m, con las que el algoritmo va a trabajar, es un parámetro de entrada. El tipo de función base también está determinado y va a ser una gaussiana.

Por tanto, en esta primera etapa, el algoritmo determinará un valor inicial para el vector de centros de las RBFs, el radio de las mismas y el vector de pesos de las conexiones entre las RBFs y los nodos de la capa de salida.

Cada patrón del conjunto de entrenamiento tiene la forma $\vec{e}_j = (\vec{x}_j, y_j)$, donde $\vec{x}_j$ es el vector de características de entrada del patrón e y_j es la clase de salida que representa.

El número de coordenadas del vector de centro de una RBF coincide con el número de atributos de entrada de los ejemplos. Por lo que el algoritmo lo que va a hacer para inicializar los centros de las RBFs es situarlas en m patrones de entrada, de forma aleatoria, es decir, se escogen aleatoriamente m patrones del conjunto de entrenamiento, de forma que el centro $\vec{c}_i$ de cada RBF ϕ_i se sitúa en uno de esos patrones, intentando que las diferentes clases queden representadas por las RBFs. Esta idea se recoge en la ecuación 3.10. Es decir, si el número de neuronas fijado es menor que el número de clases k del problema, se sitúan en patrones aleatorios pero de distintas clases; si el número de neuronas es mayor que el número de clases, se intenta

primero situarlas en patrones de clases diferentes y cuando ya exista una neurona situada en patrones de todas las clases, se completan eligiendo nuevos patrones de clases ya repetidas.

$$\vec{c}_i = \vec{x}_j \ \forall i, j = 1, \ldots, m$$
$$\text{tal que} \begin{cases} \text{si } m \leq k \text{ entonces } y_j \neq y_z \ \forall j, z = 1, \ldots, m \\ \text{si no } y_j \neq y_z \ \forall j, z = 1, \ldots, k \\ \quad \text{y } \forall y_h \, (h = k+1, \ldots, m) \ \exists y_z \, (z = 1, \ldots, m) \Big| y_h = y_z \end{cases} \tag{3.10}$$

Esta inicialización de los centros tiene una componente informada, en cuanto que sitúa a las RBFs en puntos del espacio de entrada y además, en la medida de lo posible, en puntos pertenecientes a diferentes clases, con lo que las distintas clases estarán representadas. Por otro lado presenta una componente aleatoria en cuanto que elige aleatoriamente un patrón de los de cada clase. Se consigue así que las RBFs estén bien distribuidas por todas las clases.

El algoritmo fija el mismo radio para todas las RBFs, $r_i = r \ \forall i = 1, \ldots, m$. El radio r_i para cada RBF ϕ_i, se inicializa a la mitad de la distancia media entre los centros ya fijados tal y como se indica en la ecuación 3.11.

$$r = \frac{\sum_{i,j=1; i \neq j}^{m} HVDM(\vec{c}_i, \vec{c}_j)}{m(m-1)} \tag{3.11}$$

Al iniciar el radio de esta forma se intenta partir con un radio de anchura media de forma que las RBFs no queden muy solapadas.

Se decide fijar el mismo radio para todas las RBFNs por resultar un mecanismo sencillo para el proceso de inicialización y conseguir un buen resultado en el desarrollo del algoritmo evolutivo.

Cada uno de los pesos w_{ij} que conecta una RBF ϕ_i con un nodo de la capa de salida j se inicializa a 0. La inicialización de los pesos a un valor bajo es lo que se suele hacer cuando se va a utilizar el algoritmo LMS para el entrenamiento de los mismos [Lipmann, 1987], y éste es el que va a utilizar el algoritmo que se está exponiendo.

Los métodos empleados para la inicialización de la red, tienen componentes aleatorias e informadas, son simples de implementar y el coste computacional asociado es bajo.

3.4.2. Entrenamiento de la red

Durante esta etapa, se entrenan los pesos que conectan las RBFs de la capa oculta con las neuronas de la capa de salida.

En el capítulo 2 se describieron diferentes técnicas para la determinación de los pesos. Se pueden clasificar dichos métodos en dos tipos, según su funcionamiento:

- Métodos que utilizan técnicas basadas en el gradiente: son métodos clásicos en redes neuronales para la determinación de los pesos [Lipmann, 1987]. Su funcionamiento consiste en ir presentando a la red las distintas muestras del conjunto de entrenamiento, para cada muestra se obtiene la diferencia entre la salida de la red y la salida real. Esta diferencia es la información de gradiente que se obtiene y se

utilizará para determinar el cambio en los pesos de forma que la salida de la red se aproxime a la real. Entre este tipo de métodos destaca la regla delta o algoritmo LMS [Widrow y Hoff, 1960].

- Métodos que utilizan técnicas de resolución de matrices: este grupo de técnicas se basan en plantear la determinación de los pesos como un problema de resolución de sistemas de ecuaciones. Entre estas técnicas cabe destacar la descomposición de Cholesky [Golub y Van Loan, 1996], método SVD [Golub y Van Loan, 1996] o OLS [Chen y otros, 1991], también descritos con anterioridad.

Los métodos del primer tipo son métodos sencillos y de baja complejidad, aunque si se necesita mucha exactitud en la determinación de los pesos son más costosos. Por otro lado, los métodos del segundo tipo aportan soluciones de gran exactitud pero tienen un coste computacional alto y además necesitan, para poder ser aplicados, que se cumplan una serie de precondiciones (en cuanto a la distribución de las funciones bases, el solapamiento entre éstas, etc.) que no siempre se satisfacen.

Los algoritmos de diseño de RBFNs, en particular, suelen utilizar más dos de los métodos anteriores, LMS y SVD. Incluso en [Whitehead y Choate, 1996] propone un algoritmo evolutivo en donde se usan los dos métodos: el algoritmo LMS se utiliza, al ser menor costoso durante todas las generaciones del algoritmo menos en la última que utiliza el SVD para determinar de una forma más exacta la configuración definitiva de los pesos.

En el algoritmo que estamos describiendo, se utiliza el algoritmo LMS. Es decir, en cada generación el algoritmo determina los vectores de pesos de los individuos mediante el algoritmo LMS.

Algoritmo LMS

El algoritmo LMS (*Least Mean Square*) o algoritmo de la regla delta fue propuesto por Widrow, [Widrow y Hoff, 1960]. Básicamente consiste en aplicar una regla lineal para la corrección del error tras el procesamiento de una muestra del conjunto de entrenamiento (figura 3.5).

```
Para k=1 Hasta n Hacer
    Tomar el patrón k (entrada,objetivo)(x_k, y_k)
    Calcular la salida actual de la red f(x_k)
    Adaptar los pesos según ecuación 3.12
Fin_Para
```

Figura 3.5: *Algoritmo LMS*

El algoritmo LMS adapta el vector de pesos mediante la ecuación 3.12:

$$\vec{w}_{k+1} = \vec{w}_k + \alpha \frac{e_k \vec{x}_i}{|\vec{x}_i|^2} \tag{3.12}$$

donde k indica el número de generación en la que se está, $\vec{w}_{k+1}$ es el valor actualizado del vector de pesos, $\vec{w}_k$ es el valor actual del vector de pesos, $\vec{x}_i$ es el vector de entrada actual, e_k es el error lineal actual definido como la diferencia entre la respuesta deseada y_i y la salida de la red $f(\vec{x}_i)$. El parámetro α es la velocidad de aprendizaje y mide el tamaño de la rectificación que se va a realizar.

De la elección del parámetro α no solo depende la velocidad de aprendizaje sino también la estabilidad del algoritmo [Widrow y Lehr, 1990]. Para vectores de entrada independientes en el tiempo, en la mayoría de los

casos prácticos, se asegura la estabilidad si $\alpha \in (0, 2)$, pero con valores superiores a 1 se pueden producir problemas de sobre-corrección por lo que se suele tomar un $\alpha \in$ (0.1,1).

El algoritmo LMS trabaja sobre un espacio de búsqueda determinado y realiza una correccion proporcional a α del vector de pesos. Durante su funcionamiento, provoca desplazamientos por el espacio de búsqueda en el que trabaja y α representa el valor del desplazamiento. De esta forma de operar se puede extraer que siempre que se aplica LMS no tiene por qué obtenerse una solución mejor que la anterior, ya que puede ocurrir un desplazamiento a una zona de mayor error y que para obtener una solución rápida y precisa sería conveniente que el valor de α fuese variable. De esta forma, α debería tener un valor mayor en las primeras iteraciones para avanzar rápidamente y cuando se esté cerca de una solución óptima, α debería tomar valores más pequeños, para que así los desplazamientos no salgan de la zona.

Adaptación de LMS

CO²RBFN utiliza la técnica LMS para calcular los pesos que conectan las RBFs a los nodos de la capa de salida. Esta técnica explota información local acerca del comportamiento de las RBFs, ya que va ajustando los pesos intentando aproximar la salida obtenida por la red a la salida real, ante cada patrón de entrada que se le presenta.

Se ha elegido la técnica LMS porque requiere un coste computacional bajo, en contra no se obtienen las soluciones más precisas, pero no se necesita obtener soluciones precisas del vector de pesos, sino una configuración aceptable que ayude a ver cuál es la importancia relativa de las funciones

base.

La fórmula adaptada del algoritmo LMS, al caso particular de CO^{2}RBFN es la que se muestra en la ecuación 3.13:

$$\vec{w}_{k+1} = \vec{w}_k + \alpha \frac{e_k \phi_i(\vec{x}_k)}{|\phi_i(\vec{x}_i)|^2} \tag{3.13}$$

donde $e_k = y_k - f(\vec{x}_k)$ es el error, y $|\ |$ es el módulo del vector, es decir, $|\phi_i(\vec{x}_i)|^2 = \sum_{i=1}^{m} \phi_i(\vec{x}_k)$ siendo m el número de RBFs.

Viendo la ecuación 3.13, el error es el que determina la dirección en la que los pesos se van adaptando, el resto de elementos determina el tamaño de los desplazamientos.

El parámetro α se fija con un valor de 0.2, valor recomendado en la bibliografía especializada. Como se ha comentado anteriormente interesa que el valor del parámetro α sea variable. En la parte de entrenamiento, en el algoritmo propuesto, se aplica una primera vez la adaptación de LMS con todos los patrones y con un valor α=0.4 (el valor fijado multiplicado por dos). De esta forma se hacen modificaciones más grandes en los pesos y se avanza rápidamente y después se vuelve a aplicar con un valor de α =0.2 sobre la mitad de patrones de entrenamiento. Cuando ya se está cerca de una solución óptima se utilizan valores más bajos para que así los desplazamientos queden dentro de la zona.

Las muestras que se van presentando al algoritmo LMS no se le presentan de forma ordenada, ya que podría ocurrir que las últimas muestras determinaran la dirección de la búsqueda, por lo que el orden en el que el algoritmo procesa las muestras es aleatorio.

3.4.3. Evaluación de las funciones base

En este entorno cooperativo-competitivo donde la solución final depende del comportamiento de varios componentes, tal y como se vio, al *fitness* de un individuo se le conoce como asignación de crédito. Dicha asignación de crédito evalúa el comportamiento de cada función base dentro de la red.

En el entorno de cooperación-competición, las RBFs cooperan de forma que entre todas consigan cubrir todo el espacio del problema y además que esto se logre de forma que éstas estén poco solapadas (así se consigue una buena solución final con el menor número de neuronas). Por otro lado, las RBFs compiten por permanecer en la población, de forma que aquellas que tengan un mal comportamiento serán eliminadas.

Las ideas anteriores deben tenerse en cuenta a la hora de diseñar la asignación de crédito. De esta forma en el modelo propuesto se considera que ésta está formada por tres parámetros: aportación, error y solapamiento.

Para cada RBF ϕ_i, se definen estos tres parámetros:

- Aportación, a_i de la RBF. Este parámetro se considera para premiar aquellas RBFs que tengan una contribución alta a la salida de la red. Una RBF tiene una contribución alta cuando los valores del vector de pesos, que la conectan con los nodos de la capa de salida, son altos. Pero ese indicador no es único, una RBF debe cubrir una zona amplia del espacio, es decir, debe abarcar dentro de su radio un buen número de puntos del espacio. De esta forma una RBF que tenga mucho peso pero no cubra muchos puntos aporta menos a la salida de la red que otras que tenga un peso un poco menor pero cubran una zona más

amplia del espacio.

La aportación se calcula teniendo en cuenta tanto el peso de la RBF $\vec{w}_i$, como el número de patrones del conjunto de entrenamiento que están dentro de su radio, pi_i, tal y como se indica en la ecuación 3.14:

$$a_i = \begin{cases} |w_i| & \text{si} \quad pi_i \geq q \\ |w_i| * (pi_i/q) & \text{en otro caso} \end{cases} \tag{3.14}$$

donde donde q es la media de los valores pi_i menos la desviación típica de dichos valores pi_i y w_i es la coordenada máxima de su vector de pesos, dado que es la que representa la clase que determina la RBF.

Así, una RBF con poco peso y que cubra pocos patrones tendrá una aportación baja, de forma que se premia a las RBFs con alto peso y que cubran una zona del espacio amplia.

- Error, e_i, que comete la RBF. Este parámetro, se obtiene, como la proporción de patrones mal clasificados dentro del radio de la RBF. La ecuación 3.15 muestra el cálculo del error. En ella $pimc_i$ y pi_i son el número de patrones mal clasificados y el número total de patrones respectivamente, que están dentro del radio de la RBF.

$$e_i = \frac{pimc_i}{pi_i} \tag{3.15}$$

- Solapamiento, s_i, de la RBF con otras RBFs. El solapamiento se cuantifica usando el parámetro s_i. En el cálculo de este parámetro se tiene en cuenta la metodología de *fitness sharing* [Goldberg y Richardson, 1987], la cual ayuda a mantener la diversidad de la población. Este factor se expresa tal y como muestra la ecuación 3.16,

$$s_i = \sum_{j=1}^{m} s_{ij} \qquad s_{ij} = \begin{cases} (1 - \|\phi_i - \phi_j\|/r_i) & \text{si } \|\phi_i - \phi_j\| < r_i \\ 0 & \text{en otro caso} \end{cases} \tag{3.16}$$

donde s_{ij} mide el solapamiento entre la RBF ϕ_i y la ϕ_j, $j = 1 \ldots m$.

Los tres parámetros anteriormente descritos para evaluar la asignación de crédito, son sencillos de calcular y garantizan una buena disposición en el espacio de las RBFs, de forma que se cubra el espacio con un mínimo de solapamiento.

3.4.4. Operadores del algoritmo evolutivo

En el proceso evolutivo de diseño de RBFNs es necesario: un operador que pueda eliminar RBFs que tengan un mal comportamiento, un operador que no modifique nada en una RBF con un comportamiento aceptable y operadores de mutación que realicen modificaciones leves en las RBFs. Para la mutación se proponen dos operadores: con información y sin ella.

Hay que destacar que los operadores se aplican a todos los individuos (todas las RBFs) de la población.

A continuación se describen con detalle cada uno de los operadores:

1. Operador Elimina: elimina una RBF que tiene un mal comportamiento, es decir, una RBF que comete un gran error a la hora de clasificar los patrones que están dentro de su radio de activación.

2. Operador Mutación Aleatoria: modifica el centro y el radio de una

RBF de forma aleatoria.

Dada una RBF a mutar, no se van a modificar todas las coordenadas de su centro. El número de coordenadas a mutar se determina generando aleatoriamente un número entre 1 y un 25 % del número total de coordenadas.

Si la coordenada a mutar es de tipo numérico, se decrementa o incrementa (de forma aleatoria con 50 % de probabilidad para cada caso) en una cantidad aleatoria h comprendida entre un 5 % y un 50 % del valor del radio, ecuación 3.17:

$$c'_{ij} = \begin{cases} c_{ij} - h & \text{si } u = 0 \\ c_{ij} + h & \text{si } u = 1 \end{cases} \tag{3.17}$$

con $0.05 \cdot r_i \leq h \leq 0.5 \cdot r_i$ y $u \in \{0, 1\}$.

Por otro lado, si la coordenada que se va a mutar es de tipo nominal, no se puede sumar o restar una cantidad porque pueden resultar valores que no están en rango del atributo. Si la RBF está representando la clase k, lo que se hace es calcular la distancia HVDM entre cada uno de los posibles valores, $dom(c_{ij})$, que puede tomar la coordenada c_{ij} con el valor que suele aparecer cuando la clase es k. Estas distancias indican la proximidad de cada valor al más probable para esa clase. A partir de estas distancias calculadas se obtienen probabilidades de salto que son inversamente proporcionales a la cercanía al valor representativo de la clase. De esta forma la probabilidad de saltar a un valor más cercano es mayor que la de saltar a uno más alejado. El nuevo valor para la coordenada del centro, c'_{ij} , se toma eligiendo de forma aleatoria uno de los valores posibles dentro del dominio.

La ecuación 3.18 muestra los cambios que provoca la mutación no informada sobre ciertas coordenadas del centro:

$$\begin{gathered} c'_{ij} = v_d \text{ con probabilidad } p_d \\ v_d \in dom(c_{ij}) \text{ y } p_d = 1/d(v_d/k) \end{gathered} \tag{3.18}$$

midiendo $d(v_d/k)$ la proximidad del valor v_d al valor que aparece cuando la clase de salida es la k.

El radio no siempre se va a modificar. La probabilidad de modificarlo es inversamente proporcional al número de características que presenta el problema que se está tratando. Si se decide modificarlo, el nuevo valor para el radio r', se obtiene incrementando o decrementando (de forma aleatoria con 50 % de probabilidad para cada caso) el radio antiguo r, en una cantidad aleatoria h comprendida entre un 5 % y un 50 % de su valor.

La ecuación 3.19 muestra cómo se realiza la mutación aleatoria del radio:

$$r' = \begin{cases} r & \text{si } u >= p_r \\ r - h & \text{si } u < p_r \text{ y } u_1 = 0 \\ r + h & \text{si } u < p_r \text{ y } u_1 = 1 \end{cases} \tag{3.19}$$

con $0.05 \cdot r_i \leq h \leq 0.5 \cdot r_i$, $u, u_1 \in \{0, 1\}$ y $p_r = 1/nAtributos$, la probabilidad de mutar el radio. Como todo tipo de mutaciones aleatorias, este operador intenta provocar pequeños cambios en los valores de la RBF de forma que se exploren zonas cercanas.

3. Operador Mutación Informada: utiliza información del entorno de la

RBF para modificar su centro $\vec{c_i}$ y su radio r_i.

El objetivo que se persigue con la mutación del centro de la RBF es aproximar ésta al centro de los patrones que pertenecen a la misma clase de la RBF y que están dentro del radio de ésta.

Se van a modificar todas las coordenadas j del centro $\forall j = 1 \ldots n$. Si la coordenada a mutar es de tipo numérico se calcula $\vec{dm}$, la media de las distancias entre el centro y los patrones que están dentro del radio de la RBF y que pertenecen a la clase que predice la RBF. La coordenada del centro, c_{ij}, se modifica como se muestra en la ecuación 3.20, es decir, la coordenada sufre un decremento o incremento en una cantidad aleatoria $h(0.05 \cdot r_i \leq h \leq 0.5 \cdot r_i)$:

$$c'_{ij} = \begin{cases} c_{ij} - h & \text{si } c_{ij} > dm_j \\ c_{ij} + h & \text{si } c_{ij} \leq dm_j \end{cases} \tag{3.20}$$

donde c'_{ij} es el nuevo valor para la coordenada j del centro, c_{ij} es el valor antiguo y dm_j es el valor de la coordenada j de la distancia media calculada.

Si la coordenada es de tipo nominal, se calcula la distancia entre cada uno de los posibles valores, $dom(c_{ij})$, que puede tomar la coordenada c_{ij} y el valor de la coordenada j de la distancia media calculada dm_j. Todos aquellos valores v_d cuya distancia, a la coordenada de la media $d(v_d, dm_j)$, sea inferior o igual a la distancia entre el valor de la coordenada del centro actual y la coordenada de la media $d(c_{ij}, dm_j)$, se consideran posibles valores para el cambio. El nuevo valor para la coordenada del centro se asigna eligiendo de forma aleatoria uno de los valores posibles calculados.

La ecuación 3.21 muestra los cambios que provoca la mutación informada sobre todas las coordenadas del centro.

$$\begin{aligned} c'_{ij} = v_d,\ v_d \in v_1, v_2, \ldots, v_p \text{ siendo } v_d \in dom(c_{ij}) \\ \text{y } d(v_d, dm_j) < d(c_{ij}, dm_j) \forall d = 1, 2, \ldots, p \end{aligned} \tag{3.21}$$

El objetivo de modificar el radio es que el máximo número posible de patrones que son de la misma clase que una RBF queden dentro del radio de ésta. De esta forma la variación del radio se hace de la forma indicada en la ecuación 3.22:

$$\begin{cases} r' = r - h & \text{si} \quad u \leq D \\ r' = r + h & \text{en otro caso} \end{cases} \qquad D = \frac{npnci}{npci} \qquad A = \frac{npci2}{npnci2} \tag{3.22}$$

donde h es un número aleatorio ($0.05 \cdot r_i \leq h \leq 0.5 \cdot r_i$), u es un número aleatorio($u \leq D + A$), $npnci$ es el número de patrones que no pertenecen a la misma clase de la RBF y que están dentro de su radio, $npci$ es el número de patrones que pertenecen a la misma clase de la RBF y están dentro de su radio, $npci2$, es el número de patrones que pertenecen a la misma clase de la RBF y están dentro de dos veces su radio y $npnci2$, es el número de patrones que no pertenecen a la misma clase de la RBF y están dentro de dos veces su radio.

Con esta mutación informada, se pretende explorar zonas cercanas del espacio, al igual que con la mutación no informada, pero explotando, en este caso, la información local del entorno de la neurona. De esta forma la información del entorno ayuda a dirigir la dirección del espacio

por la que se va a seguir.

4. Operador Nulo: este operador no altera ningún parámetro de la RBF. Se decide utilizar este operador dado que a todas las RBFs se les ha de aplicar algún operador y no interesa modificar aquellas RBFs que presenten un buen comportamiento. De esta forma no se tiene que hacer una selección previa de RBF a las que aplicar los operadores.

Los operadores de mutación permiten obtener un equilibrio adecuado entre la explotación y la exploración del entorno, lo cual es deseable en cualquier algoritmo evolutivo. La mutación informada utiliza información del entorno de la RBF para conseguir su adaptación óptima. En el otro lado, la mutación aleatoria provoca alteraciones en la RBF que promueven una mayor exploración del entorno para evitar óptimos locales.

3.4.5. Sistema basado en reglas difusas para la determinación de operadores a aplicar en el algoritmo evolutivo

Los operadores se aplican a la población entera de RBFs. La probabilidad de elegir un operador se determina mediante un SBRD tipo Mamdani [Mamdani y Assilian, 1975], que representa conocimiento experto para la aplicación de los operadores de forma que se pueda obtener una red simple y precisa.

Según se vio en el capítulo 1 un sistema basado en reglas difusas transforma variables de entrada en variables de salida, utilizando para ello un fuzzificador, una base de conocimiento, un motor de inferencia y un desfuzzificador.

Las entradas de este sistema son los parámetros a_i, e_i y s_i usados para definir la asignación de crédito de la RBF ϕ_i. Las salidas del sistema, $p_{elimina}$, p_{ma}, p_{mi} y p_{nulo}, representan la probabilidad de aplicación de los operadores Elimina, Mutación aleatoria, Mutación informada y Nulo, respectivamente.

Los componentes del SBRD utilizados se describen a continuación.

Base de conocimiento

En la base de conocimiento se refleja el conocimiento experto que se va a utilizar en la inferencia. La base conocimiento está dividida en la base de datos y la base de reglas difusas:

- Base de datos: en ella se encuentra la información sobre las variables y etiquetas lingüísticas.

  Se consideran tres etiquetas lingüísticas va_i, ve_i y vs_i, para cada una de las variables de entrada y para las variables de salida: $v_{p_{elimina}}$, $v_{p_{ma}}$, $v_{p_{mi}}$ y $v_{p_{nulo}}$. El número de etiquetas lingüísticas se ha determinado en base a información experta y el sistema difuso se ha definido de acuerdo con su significado.

  Para cada una de las variables lingüísticas de entrada se definen tres etiquetas lingüísticas {Baja, Media, Alta} que definen sus correspondientes conjuntos difusos (o partición del espacio de entrada). Se han elegido funciones triangulares como funciones de pertenencia para dichos conjuntos difusos. En la figura 3.6 se muestran particiones de Ruspini utilizadas para las variables lingüísticas de entrada [Ruspini, 1969].

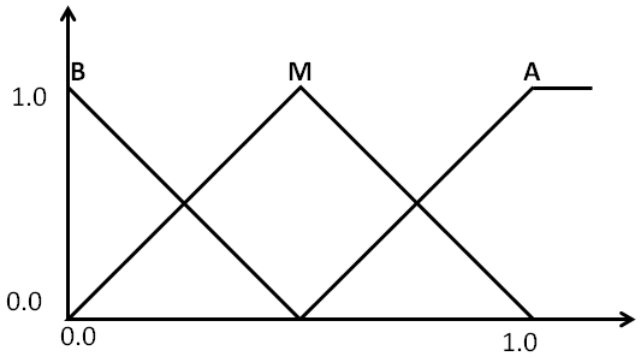


Figura 3.6: *Funciones de pertenencia para las variables de entrada*

En cuanto a las variables lingüísticas de salida, se define un conjunto de cuatro etiquetas lingüísticas {Baja, Media-Baja, Media-Alta, Alta} que representan la división del espacio de salida. En este caso también se han elegido funciones de pertenencia triangulares para los conjuntos difusos que definen estas etiquetas lingüísticas. En la figura 3.7 se muestran las funciones de pertenencia para las variables de salida. En este caso el número de etiquetas es cuatro para que se tenga más precisión a la hora de cuantificar la probabilidad de aplicación de los operadores y además conseguir variaciones progresivas en la red.

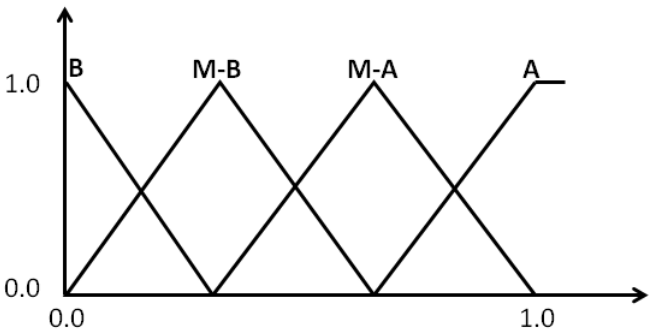


Figura 3.7: *Funciones de pertenencia para las variables de salida*

- Base de reglas difusas. Almacena el conocimiento experto que se posee sobre el diseño de RBFNs y van a almacenar las directrices que guían el proceso de razonamiento.

  Una regla de un sistema difuso representa una relación o implicación entre antecedentes y consecuentes. Las variables de entrada a_i, e_i y s_i una vez fuzzificadas va_i, ve_i y vs_i, constituyen los antecedentes de las reglas, mientras que las variables de salida $v_{p_{elimina}}$, $v_{p_{ma}}$, $v_{p_{mi}}$ y $v_{p_{nulo}}$, forman los consecuentes. La tabla 3.2 muestra la base de reglas que relacionan los antecedentes con los consecuentes descritos.

Tabla 3.2: *Base de reglas difusas que representan conocimiento experto en el diseño de RBFNs*

	Antecedentes			Consecuentes			
	v_a	v_e	v_s	$v_{p_{elimina}}$	$v_{p_{ma}}$	$v_{p_{mi}}$	$v_{p_{nulo}}$
R1	B			M-A	M-A	B	B
R2	M			M-B	M-A	M-B	M-B
R3	A			B	M-A	M-A	M-A
R4		B		B	M-A	M-A	M-A
R5		M		M-B	M-A	M-B	M-B
R6		A		M-A	M-A	B	B
R7			B	B	M-A	M-A	M-A
R8			M	M-B	M-A	M-B	M-B
R9			A	M-A	M-A	B	B

En la tabla 3.2 cada fila representa una regla. Por ejemplo, la interpretación de la primera regla es: Si la contribución de la RBF es Baja, Entonces la probabilidad de aplicar el operador Elimina es Medio-Alta, la probabilidad de aplicar el operador Mutación aleatoria es Medio-Alta, la probabilidad de aplicar el operador Mutación informada es Baja y la de aplicar el operador Nulo es Baja. Como ya se ha mencionado, la base de reglas representa conocimiento experto

en el diseño de RBFNs.

El conocimiento experto recogido en el SBRD determina que:

- Se considera que una RBF es peor cuando más baja es su aportación (a_i), más alto su error (e_i) y más alto su solapamiento (s_i). En el otro lado, una RBF tiene un mejor comportamiento cuando más alta es su aportación, más bajo su error y más bajo su solapamiento.
- La variable $v_{p_{elimina}}$, que representa la probabilidad de eliminar una RBF, presenta mayores valores cuando menor es v_a, lo cual implica que una RBF que aporta poco a la salida de la red (tiene un peso bajo o cubre un número bajo de patrones) va a tener mayor probabilidad de ser eliminada de la red.
- Con respecto a la variable v_e la probabilidad de eliminar es en cierta manera proporcional a ésta, de forma que cuanto mayor es el error cometido por una RBF mayor es la probabilidad de eliminarla.
- En cuanto al solapamiento, representado por la variable v_s, la tendencia implica que a medida que este crece también lo hace la probabilidad de eliminar la neurona.
- Si se observa la variable $v_{p_{nulo}}$, que representa la probabilidad de no aplicar ningún operador a la RBF, se muestra que sigue la misma evolución que la variable v_a, de forma que una RBF con una aportación baja tiene pocas probabilidades de mantenerse igual, mientras que si la aportación es alta su probabilidad de no cambiar incrementa.

- Con respecto a v_e su evolución es inversa, ya que si una RBF tiene un error pequeño es más alta su probabilidad de mantenerse y la probabilidad decrece a medida que el error se hace más grande.

- En cuanto a la variable v_s tiene también un comportamiento similar al error, de forma que a medida que la RBF presenta un mayor solapamiento decrece la probabilidad de que se mantenga sin modificaciones.

- Las variables $v_{p_{ma}}$ y $v_{p_{mi}}$, que representan la probabilidad de aplicar el operador de mutación aleatoria y de mutación informada respectivamente, suelen presentarse con un valor alto, esto es para que los operadores de mutación tengan una alta probabilidad de aplicarse de forma que se promueve una evolución progresiva en la red. Así, se intenta que de una generación a la siguiente no haya cambios bruscos y no se rompa una continuidad en el acercamiento a la solución final. Si la variable v_a es baja, no está aportando mucho a la red, pero se puede pensar que la RBF no tiene bien ajustados sus parámetros y se intenta, antes de eliminarla, ajustarlos mediante la mutación aleatoria.

- El mismo razonamiento se pueden hacer para las variables v_e y v_s. La probabilidad de realizar una mutación aleatoria es, en algunas ocasiones, más alta que la de la mutación informada, para ampliar la capacidad de exploración del algoritmo. Además es menos costosa de realizar, ya que no tiene que estudiar el entorno de la RBF.

El fuzzificador

La función de este componente es transformar los valores nítidos que se dan como entrada al sistema en valores difusos. Una vez transformados, dichos valores, pasan al motor de inferencia en donde se convierten en los antecedentes de las reglas.

En el algoritmo propuesto, el fuzzificador recibe los valores de entrada a, e y s, y para cada uno de ellos fuzzifica su valor y obtiene una cuantificación difusa del mismo según sus etiquetas lingüísticas o conjuntos difusos definidos para él.

Para realizar la fuzzificación se utiliza el método tradicional, que consiste en calcular el grado de pertenencia al conjunto difuso correspondiente.

Motor de inferencia

El motor de inferencia se encarga de extraer consecuencias tras aplicar las reglas a los antecedentes correspondiente. Los valores de los consecuentes dependen del valor que el fuzzificador extrae de las variables de entrada y de la base de reglas difusas.

El el algoritmo propuesto utiliza como motor de inferencia un sistema tipo Mamdani [Mamdani y Assilian, 1975]. El mecanismo de razonamiento se configura estableciendo como t-conorma el máximo, y como t-norma el mínimo.

El defuzzificador

Una vez aplicado el mecanismo de razonamiento se obtienen valores difusos para las variables de salida que tendrán que transformarse en valores nítidos. El defuzzificador es el encargado de realizar dicha transformación. Existen diferentes expresiones para el operador de defuzzificación tal y como se describe en la figura 1.19.

En el algoritmo propuesto se ha optado por la estrategia del *centro del área*. Gráficamente, consiste en obtener el centro del área de la forma de la función que delimitan las etiquetas para una variable lingüística dada, para ello se utiliza la ecuación 1.30.

La salida del defuzzificador es la salida del SBRD, es decir, las probabilidades de aplicar los operadores Elimina, Mutación aleatoria, Mutación informada y Nulo.

3.4.6. Estrategia de reemplazo

Tras aplicar los operadores de mutación, aparecen nuevas RBFs. El algoritmo usa una estrategia de reemplazo para determinar cuáles de las RBFs serán incluidas en la nueva población. Para ello se compara el comportamiento de la nueva RBF con la del padre para determinar con cual de ellas la red tiene un comportamiento global mejor. Con la que se consiga un mejor comportamiento será la que se incluirá en la nueva población.

3.4.7. Introducción de nuevas funciones base

En este paso del algoritmo, las RBFs eliminadas se van a sustituir por otras nuevas, de forma que se mantenga el tamaño de la población.

En el algoritmo se implementan dos formas de elegir el sitio dónde situar la nueva RBF que se va introducir. La selección de una de las dos opciones se realiza generando un número aleatorio y ambas opciones tienen asociada una probabilidad de 0.5.

Una de las opciones es puramente aleatoria y consiste en situar el centro de la nueva RBF en un patrón del conjunto de entrenamiento elegido al azar y que no esté cubierto por ninguna otra RBF. La otra opción estudia el espacio y busca un patrón que no esté dentro del radio de ninguna RBF y que esté siendo mal clasificado por la red. El centro de la nueva RBF se sitúa en dicho patrón.

En cualquiera de las dos opciones el radio de la nueva RBF se inicializa con el valor del radio medio de las RBF que están en la población y su peso es inicializado a 0.

Si llamamos ϕ_j a la nueva RBF que se va a insertar, en la ecuación 3.23 se muestran los valores para sus parámetros:

$$\begin{aligned} \vec{c}_j =& \begin{cases} \vec{p}_s \in E \mid d(\vec{p}_s, \vec{c}_k) > r_k \forall \phi_k & \text{si } u = 0 \\ \vec{p}_s \in E \mid d(\vec{p}_s, \vec{c}_k) > r_k \forall \phi_k \ y \ f(\vec{p}_s) \neq y_s & \text{si } u = 1 \end{cases} \\ r_j =& \frac{\sum_{i=1}^{l} r_i}{l} \qquad u = \{0, 1\} \\ \vec{w}_j =& 0 \qquad u = \{0, 1\} \end{aligned} \tag{3.23}$$

donde E es el conjunto de patrones de entrenamiento, $f(\vec{p}_s)$ es la clase que da como salida la red para el patrón $\vec{p}_s$, y_s es la clase a la que pertenece p_s y l es el número de RBFs que están en la red en el momento de la inserción.

3.5. Resultados experimentales

Para probar la eficacia del modelo diseñado, se ha aplicado éste a resolver el problema de clasificación sobre once bases de datos. Los resultados obtenidos se han comparado con los de otros cinco métodos *soft-computing* diferentes. En el estudio se compara tanto la eficiencia como la complejidad de los métodos.

La colección de bases de datos usada se ha obtenido del repositorio UCI (*Repository of Machine Learning Database*) [Asuncion y Newman, 2007a]. En la tabla 3.3 se muestran las características de las bases de datos utilizadas.

Tabla 3.3: *Características de las bases de datos*

Bases Datos	#Instancias	#Atributos Numéricos	#Atributos Nominales	#Clases
Car	1728	0	6	4
Credit	690	6	9	2
Glass	114	9	0	7
Hepatitis	155	6	13	2
Ionosphere	351	34	0	2
Iris	150	4	0	3
Pima	768	8	0	2
Sonar	208	60	0	2
Wbcd	699	9	0	2
Vehicle	846	18	0	4
Wine	178	13	0	3

Las bases de datos Car, Credit, Hepatitis y Wbcd presentan valores perdidos por lo que se les ha realizado un preprocesamiento. En dicho preprocesamiento, los valores perdidos de atributos de tipo numérico se han sustituido por la media de los valores de dicho atributo en el resto de las instancias que son de su misma clase. En el caso de atributos de tipo nominal se ha utilizado la moda.

El estudio se ha realizado usando 10-validación cruzada, es decir, diez particiones para entrenamiento y test, el 90 % de los datos para entrenamiento y el 10 % para test. Para cada base de datos se obtiene y se muestra la media de las diez particiones.

CO²RBFN se ha comparado con diferentes métodos que cubren un amplio rango dentro del campo del aprendizaje máquina: otros paradigmas alternativos para el diseño de RBFNs, otro modelo de red neural (Perceptrón Multicapa), y un modelo de árbol de decisión. Específicamente:

- GeneticRBFN: algoritmo evolutivo clásico para el diseño de RBFNs basado en un esquema *Pittsburgh* donde cada individuo es la red completa. La implementación se ha realizado específicamente para hacer la comparación. Una descripción más extensa se puede observar en el apéndice B.

- C4.5: algoritmo que genera reglas de clasificación, en forma de árboles de decisión, a partir de las bases de datos mediante particiones realizadas recursivamente [Quilan, 1993].

  El algoritmo consiste en ir creando un árbol, a partir de una raíz, en el que cada nodo representa un atributo, cada posible valor del atributo una rama y las hojas son los conjuntos ya clasificados de ejemplos

y etiquetados con el nombre de una clase. Para ir construyendo el árbol se sigue una estrategia profundidad-primero, se asigna al nodo siguiente el atributo que ofrezca mayor ganancia de información y se crea una nueva rama para cada valor que pueda tomar dicho atributo. A continuación se clasifican los ejemplos del conjunto de entrenamiento de ese nodo entre sus descendientes, si todos los ejemplos del conjunto de entrenamiento quedan clasificados el proceso se para, si no es así el proceso sigue realizándose sobre los descendientes de ese nodo, sin volver a utilizar el atributo ya utilizado. C4.5 permite trabajar tanto con atributos continuos como discretos, utiliza una heurística de ganancia proporcional en caso de que la ganancia no sea conveniente, utiliza un método de poda para evitar sobre-aprendizaje y es capaz de solucionar problemas de atributos con valor desconocido mediante un método probabilístico.

La implementación se ha obtenido de KEEL [Alcalá-Fdez y otros, 2009].

- MLP-Back: algoritmo para el diseño de redes Perceptrón Multicapa que usa el algoritmo *Back-propagation* para el aprendizaje [Rojas y Feldman, 1996]. Esta clase de redes se compone de varias capas de neuronas interconectadas, generalmente, de forma *feed-forward*, cada neurona de una capa dirige sus conexiones a las neuronas de la capa siguiente. Como técnica de aprendizaje usa la propagación. Los valores de salida se comparan con la salida real para así calcular el error. El algoritmo utiliza la información de error para ir ajustando los pesos de las conexiones, de forma que el error se vaya minimizando. Este proceso se repite un número de veces hasta que se consigue un error

pequeño.

Como indica el nombre del algoritmo la propagación de los errores (y por lo tanto el aprendizaje) se propagan hacia atrás desde los nodos de salida a los nodos interiores. Así que técnicamente hablando, *Back-propagation* se utiliza para calcular el gradiente del error de la red con respecto a los pesos modificables de la red. Este gradiente se utiliza en un simple algoritmo estocástico de descenso de gradiente para encontrar los pesos que minimizan el error.

La implementación se ha obtenido de KEEL.

- RBFN-Decr: algoritmo para el diseño de RBFNs basado en un esquema decremental [Broomhead y Lowe, 1988] . El método parte de una red compleja y va eliminando RBFs que considera innecesarias de forma que se llega a una red menos compleja y que proporciona una buena solución. En la sección 2.3.4 del capítulo 2 se describe este método.

  La implementación se ha obtenido de KEEL.

- RBFN-Incr: algoritmo para el diseño de RBFNs basado en un esquema incremental [Plat, 1991]. En este caso el algoritmo comienza con una red pequeña, con pocas o una RBF, y va añadiendo RBFs hasta que logra una buena solución. En la sección 2.3.4 del capítulo 2 se describe este método.

  La implementación se ha obtenido de KEEL.

Tal y como se ha comentado con anterioridad, en CO²RBFN la población al completo codifica una sola red, y el número de individuos dentro de la misma es igual al número de nodos o RBFs. CO²RBFN se ha ejecutado con

un número de RBFs que oscila entre el número de clases de la base de datos y cuatro veces este número de clases, de todas éstas ejecuciones se toma el resultado mejor para medir la eficiencia de la red. En el apéndice A se muestran los resultados de todas las ejecuciones realizadas con CO^{2}RBFN.

Los valores de los parámetros utilizados por CO^{2}RBFN se muestran en la tabla 3.4. Los parámetros de GeneticRBFN se encuentran en el apéndice B donde el algoritmo se describe. El resto de algoritmos con los que comparamos, C4.5, MLP-Back, RBFN-Decr y RBFN-Incr, utilizan como valores de los parámetros los aconsejados por los autores de los mismos y se muestran en el apéndice C. El número de repeticiones para los algoritmos no determinísticos se ha fijado a 5.

Tabla 3.4: *Parámetros de CO^{2}RBFN*

Parámetro	*Valor*
Generaciones del ciclo principal	200
Número de RBF's	Mínimo = número de clases Máximo = 4 · número de clases

De la tabla 3.5 a la tabla 3.15 se muestra el porcentaje de acierto en test de la clasificación, así como la correspondiente complejidad (número de nodos o número de reglas) de los métodos.

3.6. Análisis de resultados

Un primer análisis de los datos muestra que CO^{2}RBFN obtiene RBFNs cuyos resultados en cuanto al acierto en test son comparables a los obtenidos por los otros métodos (incluso mejores en seis de las bases de datos) y las

Tabla 3.5: *Resultados con la base de datos Car*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	119.4	91.550 ± 0.949
CO²RBFN	**5.0**	81.007 ± 4.800
GeneticRBFN	15.9	80.783 ± 6.259
MLP-Back	30.0	49.245 ± 7.607
RBFN-Decr	16.0	73.847 ± 5.999
RBFN-Incr	1340.6	**93.171 ± 1.390**

Tabla 3.6: *Resultados con la base de datos Credit*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	22.0	**87.101 ± 2.629**
CO²RBFN	**2.0**	84.232 ± 6.500
GeneticRBFN	7.0	85.507 ± 6.183
MLP-Back	30.0	82.609 ± 3.834
RBFN-Decr	7.3	61.449 ± 3.855
RBFN-Incr	599.9	66.087 ± 4.260

Tabla 3.7: *Resultados con la base de datos Glass*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	26.3	67.443 ± 11.648
CO²RBFN	22.0	**67.710 ± 11.100**
GeneticRBFN	27.0	65.089 ± 12.545
MLP-Back	30.0	37.609 ± 4.246
RBFN-Decr	**15.6**	24.438 ± 17.604
RBFN-Incr	124.0	54.072 ± 10.434

redes obtenidas tienen una complejidad baja. Además se puede observar que suele tener desviaciones típicas bajas en los resultados, lo que implica que

Tabla 3.8: *Resultados con la base de datos Hepatitis*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	**7.1**	**89.647 ± 6.704**
CO^{2}RBFN	8.0	87.399 ± 7.400
GeneticRBFN	7.5	86.711 ± 8.591
MLP-Back	30.0	71.817 ± 11.219
RBFN-Decr	12.6	76.711 ± 7.287
RBFN-Incr	120.1	76.637 ± 7.077

Tabla 3.9: *Resultados con la base de datos Ionosphere*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	13.2	90.632 ± 3.804
CO^{2}RBFN	8.0	91.411 ± 3.900
GeneticRBFN	**7.5**	**92.885 ± 4.358**
MLP-Back	30.0	72.948 ± 8.473
RBFN-Decr	13.0	82.348 ± 11.756
RBFN-Incr	180.5	92.592 ± 5.003

Tabla 3.10: *Resultados con la base de datos Iris*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	**4.8**	94.000 ± 4.667
CO^{2}RBFN	6.0	**96.267 ± 4.200**
GeneticRBFN	10.4	95.067 ± 5.131
MLP-Back	30.0	64.667 ± 16.344
RBFN-Decr	9.6	94.000 ± 6.289
RBFN-Incr	29.8	94.667 ± 5.819

el método desarrollado es robusto.

Tal y como se indica en [García y otros, 2009b], es necesario realizar

Tabla 3.11: *Resultados con la base de datos Pima*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	18.3	73.972 ± 3.769
CO²RBFN	**4.0**	**75.950 ± 4.900**
GeneticRBFN	7.6	75.615 ± 5.370
MLP-Back	30.0	70.819 ± 6.888
RBFN-Decr	7.6	72.014 ± 2.451
RBFN-Incr	671.4	67.728 ± 5.010

Tabla 3.12: *Resultados con la base de datos Sonar*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	14.3	71.071 ± 12.320
CO²RBFN	8.0	**75.086 ± 9.800**
GeneticRBFN	**7.7**	73.305 ± 9.353
MLP-Back	30.0	67.762 ± 10.382
RBFN-Decr	13.8	59.143 ± 8.540
RBFN-Incr	159.8	74.976 ± 12.095

Tabla 3.13: *Resultados con la base de datos Vehicle*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	71.8	**71.034 ± 3.0165**
CO²RBFN	16.0	69.192 ± 4.700
GeneticRBFN	16.0	67.043 ± 3.891
MLP-Back	30.0	38.066 ± 6.882
RBFN-Decr	**10.8**	45.046 ± 8.684
RBFN-Incr	750.7	56.259 ± 5.280

un análisis estadístico para detectar si existen o no diferencias significativas entre los resultados de los distintos métodos. En esta memoria, el análisis se

Tabla 3.14: *Resultados con la base de datos Wbcd*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	12.4	94.995 ± 1.845
CO^2RBFN	**5.0**	$\mathbf{97.083 \pm 1.800}$
GeneticRBFN	6.2	96.713 ± 1.933
MLP-Back	30.0	87.722 ± 27.854
RBFN-Decr	10.4	92.119 ± 6.357
RBFN-Incr	319.9	94.303 ± 3.089

Tabla 3.15: *Resultados con la base de datos Wine*

Algoritmo	#nodos/ #reglas	Precisión (%)
C4.5	**5.1**	94.902 ± 5.875
CO^2RBFN	7.0	$\mathbf{96.739 \pm 4.600}$
GeneticRBFN	10.4	95.275 ± 6.330
MLP-Back	30.0	93.301 ± 9.555
RBFN-Decr	8.3	68.562 ± 9.634
RBFN-Incr	125.3	74.739 ± 5.588

va a hacer por un lado en cuanto a la precisión alcanzada en la clasificación por parte de los algoritmos, y por otro lado en cuanto a la complejidad de los modelos obtenidos.

Demšar en [Demšar, 2006], ilustra la necesidad de utilizar estadísticos no paramétricos cuando tratamos con algoritmos evolutivos dado que no se satisfacen las características necesarias para poder utilizar test paramétricos. En base a esto, en este estudio se van a aplicar los siguientes tests no paramétricos [Demšar, 2006; García y Herrera, 2008; García y otros, 2009a]:

- El test de *Iman-Davenport* [Sheskin, 2006] que detecta si existen

diferencias estadísticas entre los resultados de los diferentes algoritmos.

- El test de Holm [Holm, 1979] que permite hacer comparaciones entre grupos de algoritmos.
- El test de ranking de signos de *Wilcoxon* [Wilcoxon, 1945] que permite establecer comparaciones entre cada dos algoritmos.

Una descripción más amplia de estos tests se encuentra en el apéndice D.

En las siguientes dos subsecciones se van a aplicar los tests a los resultados obtenidos por los métodos. En la subsección 3.6.1 se va a estudiar el comportamiento de los algoritmos desde el punto de vista de su precisión en la clasificación, mientras que su comportamiento desde el punto de vista de complejidad se estudiará en la subsección 3.6.2.

Los tests se aplican con un nivel de confianza α=0.05.

3.6.1. Análisis de la precisión en la clasificación

En esta sección se realiza un estudio de la precisión alcanzada en clasificación por los métodos estudiados. Primero se aplica el test de *Friedman* y se obtiene un ranking de los métodos. Dicho ranking es utilizado por el test de Iman-Davenport para determinar si existen o no diferencias significativas entre los métodos.

El ranking se utiliza para poder mostrar cómo de bueno es un método con respecto a los otros y se obtiene asignando una posición a cada algoritmo dependiendo de su comportamiento para cada base de datos. El algoritmo que alcanza los mejores resultados en una base de datos específica va a ser el

Algoritmo	Ranking
C4.5	2.591
CO^{2}RBFN	**1.727**
GeneticRBFN	2.455
MLP-Back	5.364
RBFN-Decr	5.136
RBFN-Incr	3.727

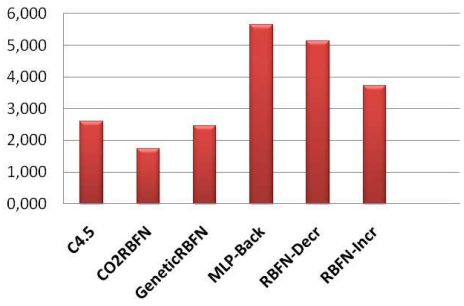


Figura 3.8: *Ranking obtenido en cuanto a la precisión de los modelos (el menor valor es el mejor)*

primero en el ranking (valor 1); después, al algoritmo con el segundo mejor resultado se le asigna un ranking 2 y así sucesivamente. Esta tarea se realiza para todas las bases de datos y finalmente se calcula un valor medio de ranking para todos los valores.

En la figura 3.8 se muestra el ranking obtenido por los algoritmos. El valor más bajo del ranking representa el algoritmo con mejor comportamiento. Se puede observar que CO^{2}RBFN es el algoritmo que tiene el mejor comportamiento en cuando a la precisión en la clasificación obtenida.

El estadístico obtenido por el test de Iman-Davenport (tabla 3.16) es 18.065. El valor crítico de la distribución F_F con 5 y 50 grados de libertad es menor que el valor del estadístico lo cual evidencia que se rechaza la hipótesis de igualdad de medias, es decir, existen diferencias significativas entre los métodos.

Una vez visto que sí existen diferencias significativas entre los resultados obtenidos por los diferentes algoritmos, se va a aplicar el test de Holm para comparar el mejor método del ranking, en este caso CO^{2}RBFN, con el resto de métodos. De esta forma se intenta ver con cuáles de ellos tiene

Tabla 3.16: *Resultados del test de Iman-Davenport en cuanto a la precisión en clasificación*

Test	Estadístico	Valor F_F	Hipótesis nula (α=0.05)
Iman-Davenport	18.065	2.400	Rechazada

diferencias significativas. Los resultados se muestran en la tabla 3.17, en la cual los algoritmos están ordenados con respecto al valor z obtenido. El valor p_i obtenido se compara con el valor $\alpha \setminus i$ situado en la misma fila de la tabla, en los casos tres primeros casos ocurre que el valor p_i es menor que el correspondiente $\alpha \setminus i$, lo que implica que rechaza la hipótesis de igualdad de medias, es decir, existen diferencias significativas entre el algoritmo de control, CO²RBFN, y los algoritmos MLP-Back, RBFN-Decr y RBFN-Incr. En el caso de C4.5 y GeneticRBFN se acepta la hipótesis de igualdad de medias, el test no distingue diferencias de comportamiento entre CO²RBFN y los algoritmos C4.5 y GeneticRBFN.

Tabla 3.17: *Resultados del test de Holm en cuanto a la precisión en la clasificación. CO²RBFN es el método de control*

i	Algoritmo	z	p	$\alpha \setminus i$	Hipótesis nula (α=0.05)
5	MLP-Back	4.558	5.154E-6	0.01	Rechazada
4	RBFN-Decr	4.274	1.924E-5	0.0125	Rechazada
3	RBFN-Incr	2.507	0.012	0.017	Rechazada
2	C4.5	1.083	0.279	0.025	Aceptada
1	GeneticRBFN	0.912	0.362	0.05	Aceptada

A continuación se aplica el test de Wilcoxon (tabla 3.18) para detectar si existen diferencias significativas entre cada par de métodos. En este caso comparamos CO²RBFN con cada uno de los restantes métodos. Como se

puede observar el p-valor obtenido es bajo cuando se compara CO^{2}RBFN con los algoritmos MLP-Back, RBFN-Decr y RBFN-Incr, lo cual indica que existen diferencias entre CO^{2}RBFN y cada uno ellos. La hipótesis de igualdad de medias se acepta cuando se compara CO^{2}RBFN con C4.5 y GeneticRBFN, aunque el p-valor obtenido al comparar con GeneticRBFN no es demasiado alto, indicando que la hipótesis se podría rechazar con una probabilidad aproximada del 90 %.

Tabla 3.18: *Resultados del test de Wilcoxon en cuanto a la precisión en la clasificación*

R^+ CO^{2}RBFN	R^-		p-valor	Hipótesis nula ($\alpha = 0.05$)
35.0	C4.5	31.0	0.859	Aceptada
52.0	GeneticRBFN	14.0	0.091	Aceptada
66.0	MLP-Back	0.0	0.003	Rechazada
66.0	RBFN-Decr	0.0	0.003	Rechazada
57.0	RBFN-Incr	9.0	0.033	Rechazada

3.6.2. Análisis de la complejidad

La complejidad de los modelos se determina en términos del número de nodos, en el caso de los algoritmos de diseño de redes, o en términos de número de reglas generadas, en el caso del algoritmo basado en árboles de decisión.

El ranking de los algoritmos atendiendo a su complejidad se muestra en la figura 3.9. Como se puede observar de nuevo CO^{2}RBFN es el mejor método, ahora en cuanto a la complejidad de los modelos obtenidos. La tabla 3.19 muestra los resultados del test de Iman-Davenport.

El estadístico de Iman-Davenport es 25.499, el valor crítico de la

Algoritmo	Ranking
C4.5	3.273
CO2RBFN	**1.773**
GeneticRBFN	2.5
MLP-Back	4.909
RBFN-Decr	2.636
RBFN-Incr	5.909

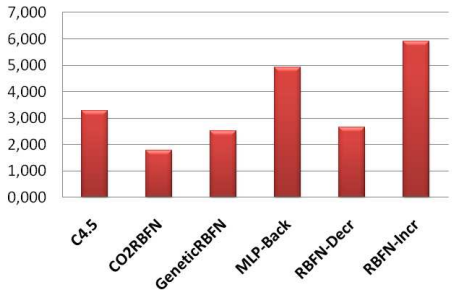


Figura 3.9: *Ranking obtenido en cuanto a la complejidad de los modelos (el menor valor es el mejor)*

distribución F_F con 5 y 50 grados de libertad es menor que el estadístico lo que implica que se rechaza la hipótesis de igualdad de medias, existen diferencias significativas entre los métodos.

Tabla 3.19: *Resultados del test de Iman-Davenport en cuanto a la complejidad de los modelos*

Test	Estadístico	Valor F_F	Hipótesis nula (α=0.05)
Iman-Davenport	25.499	2.400	Rechazada

Los resultados de aplicar el test de Holm, para estudiar la complejidad de los modelos, se muestran en la tabla 3.20. Se puede observar que el algoritmo de control, CO^2RBFN, presenta diferencias significativas respecto a RBFN-Incr y MLP-Back. Las diferencias con respecto a C4.5, RBFN-Decr y GeneticRBFN no son significativas cuando se hacen las comparaciones múltiples.

En la tabla 3.21 se muestran los resultados de aplicar el test de Wilcoxon a los algoritmos estudiando la complejidad de los modelos obtenidos. Con el test se compara CO^2RBFN con cada uno de los métodos restantes. Se

Tabla 3.20: *Resultados del test de Holm en cuanto a la complejidad de los modelos. CO²RBFN es el método de control*

i	Algoritmo	z	p	$\alpha \setminus i$	Hipótesis nula (α=0.05)
5	RBFN-Incr	5.185	2.158E-7	0.01	Rechazada
4	MLP-Back	3.932	8.437E-5	0.0125	Rechazada
3	C4.5	1.880	0.060	0.017	Aceptada
2	RBFN-Decr	1.083	0.279	0.025	Aceptada
1	GeneticRBFN	0.912	0.362	0.05	Aceptada

puede observar que existen diferencias significativas entre CO²RBFN y C4.5, GeneticRBFN, MLP-Back y RBFN-Incr, dado que la hipótesis de igualdad de medias se rechaza con un p-valor pequeño. En el caso de la comparación entre los resultados de CO²RBFN y RBFN-Decr se acepta la hipótesis de igualdad de medias con un p-valor de 0.13.

Tabla 3.21: *Resultados del test de Wilcoxon en cuanto a la complejidad de los modelos*

R^+ CO²RBFN	R^-		p-valor	Hipótesis nula ($\alpha = 0.05$)
60.0	C4.5	6.0	0.016	Rechazada
56.05	GeneticRBFN	9.5	0.028	Rechazada
66.0	MLP-Back	0.0	0.003	Rechazada
50.0	RBFN-Decr	16.0	0.130	Aceptada
66.0	RBFN-Incr	0.0	0.003	Rechazada

3.6.3. Resumen del análisis de resultados

Los ranking calculados muestran que CO²RBFN es el mejor algoritmo tanto desde el punto de vista de la precisión como de la complejidad de los modelos que consigue, tal y como se ve en las figuras 3.8 y 3.9 donde el valor

mínimo es el alcanzado por el mejor algoritmo.

El test de Iman-Daverport demuestra que existen diferencias significativas entre los métodos utilizados (tablas 3.16 y 3.19). Para poder determinar si existen diferencias significativas entre dos métodos comparados se utiliza el test de Holm para establecer comparaciones entre todos los métodos y el test de signos de Wilcoxon que compara CO²RBFN con cada uno de los otros algoritmos. El análisis estadístico realizado muestra que existen diferencias significativas entre CO²RBFN y MLP-Back, RBFN-Incr y RBFN-Decr, cuando se mide la precisión de los algoritmos, y no existen diferencias significativas cuando se compara con C4.5 y GeneticRBFN, aunque en este último caso en el test de Wilcoxon se acepta la hipótesis de igualdad de medias con un p-valor que no es muy alto. Sin embargo, si analizamos los resultados en cuando a la complejidad del modelo que consiguen, CO²RBFN sí que muestra diferencias significativas al compararlo con C4.5 y GeneticRBFN, también con MLP-Back y RBFN-Incr.

Resumiendo lo anterior, CO²RBFN supera, con diferencias significativas, desde el punto de vista de la precisión, a todos los algoritmos con los que se compara menos a C4.5 y GeneticRBFN, pero consigue mejores resultados que éstos en cuanto a la complejidad del modelo que obtiene. Cuando se analiza la complejidad, CO²RBFN obtiene los mejores resultados frente a todos los algoritmos con los que se compara excepto con RBFN-Decr, pero a éste lo supera en precisión.

Con estos resultados se puede afirmar que CO²RBFN es el algoritmo que consigue un mejor equilibrio entre la precisión y la complejidad de las RBFNs que diseña u obtiene.

3.7. Conclusiones

El algoritmo, CO^2RBFN, es un algoritmo evolutivo, con un enfoque cooperativo-competitivo, para el diseño de RBFNs aplicado a la resolución de problemas de clasificación. En él se hibridan diferentes técnicas *soft-computing*, tales como redes neuronales, algoritmos evolutivos y sistemas basados en reglas difusas.

El objetivo del método es obtener redes simples y precisas. Teniendo en mente dicho objetivo se diseña tanto el esquema del algoritmo evolutivo, de forma que se elige un enfoque cooperativo-competitivo, como el resto de componentes del algoritmo.

Para medir la asignación de crédito de las RBFs se consideran tres factores: la aportación de la RBF a la salida de la red (este factor promueve la generalización de la RBF), el error que comete la RBF en su radio de activación (éste refuerza la calidad individual) y el grado de solapamiento entre la RBF y el resto (con él se promueve una buena colocación de las RBFs).

Para conducir el proceso cooperativo-competitivo, de forma que se consiga un adecuado equilibrio entre explotación y exploración del entorno, se usan cuatro operadores: eliminar una RBF con un mal comportamiento, mantener los parámetros de una RBF con un buen comportamiento, y dos operadores que producen mutaciones sobre una RBF (cambios aleatorios y cambios guiados por la información del entorno de la RBF).

Para decidir qué operador aplicar a una RBF en un determinado momento, el algoritmo utiliza un SBRD que representa conocimiento experto

en el diseño de RBFNs. Las entradas de dicho sistema son los tres parámetros utilizados para medir la asignación de crédito: a_i, e_i, y s_i y su salida es la probabilidad de aplicación de cada uno de los operadores: $p_{elimina}$, p_{ma}, p_{mi} y p_{nulo}.

El algoritmo utiliza como distancia la medida HVDM, la cual permite manejar, con una pérdida mínima de información, tanto las diferencias entre atributos numéricos como nominales.

CO²RBFN se ha aplicado a resolver problemas de clasificación, tarea para la que se ha diseñado. Los resultados con él obtenidos sobre once bases de datos, se han comparado con los obtenidos mediante otros cinco métodos. Estos algoritmos con los que se compara cubren un rango amplio dentro del campo de aprendizaje máquina, incluyendo paradigmas alternativos en el diseño de RBFNs, otro modelo de redes neuronales y un modelo basado en árboles de decisión.

Se ha realizado un análisis estadístico de los resultados obtenidos mediante todos los algoritmos, en cuanto a la precisión en la clasificación y en cuanto a la complejidad del modelo obtenido. El análisis muestra que CO²RBFN obtiene RBFNs con adecuado equilibrio entre precisión y complejidad, superando a los otros métodos con los que se compara.

Capítulo 4

CO^2RBFN aplicado a clasificación de datos no balanceados

En este capítulo, se analiza el comportamiento del algoritmo CO^2RBFN en problemas de clasificación no balanceada, es decir, problemas en los que las diferentes clases no están igualmente representadas en los ejemplos del conjunto de entrenamiento.

Para solucionar el problema del desbalanceo de clases se han utilizado en la bibliografía especializada diferentes soluciones agrupadas en dos líneas:

- soluciones a nivel de datos, a través de diferentes formas de remuestreo y
- soluciones a nivel de algoritmo, ajustando los costes de las distintas clases del problema a nivel de clasificación.

En [Chawla y otros, 2008] se muestra la utilidad del pre-procesamiento

de los datos en clasificación no balanceada sin necesidad de modificar los algoritmos de minería de datos, y destaca entre ellos, el algoritmo SMOTE.

En este capítulo se estudia el problema de clasificación con datos de clases no balanceadas, los métodos desarrollados para trabajar con este tipo de problemas y los utilizados con RBFNs, con el objetivo de analizar el comportamiento de CO^2RBFN en problemas de clasificación no balanceada. Se estudiarán los resultados sin pre-procesamiento de datos y los resultados obtenidos con datos sobre los que se ha aplicado SMOTE.

4.1. El problema de clasificación en bases de datos no balanceadas

El problema de desbalanceo en las bases de datos se produce cuando la distribución de ejemplos de las clases es muy diferente. Si nos centramos en bases de datos con solo dos clases, el problema del desbalanceo ocurre cuando una clase está representada por un conjunto amplio de ejemplos (clase mayoritaria) mientras que el número de ejemplos que representan a la otra (clase minoritaria) es bajo, lo cual afecta negativamente a los algoritmos de clasificación [Chawla y otros, 2004]. En [Weiss y Provost, 2003] se presenta un análisis detallado sobre el efecto que la distribución desigual de las clases produce sobre los clasificadores.

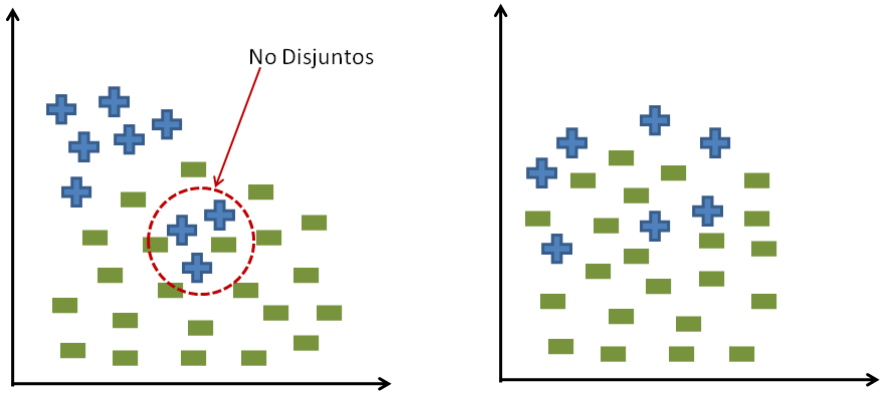
En el mundo real son muchos los conjuntos de datos en los que las distribuciones de las diferentes clases no están balanceadas, por lo que este problema es muy significativo [Yang y Wu, 2006]. Este problema se presenta en aplicaciones tales como clasificación de imágenes de satélite

[Suresh y otros, 2008], análisis de riesgos [Huang y otros, 2006], datos sobre proteínas [Provost y Fawcett, 2001], clasificación de datos de teleobservación [Bruzzone y Serpico, 1997] y especialmente en aplicaciones médicas [Kilic y otros, 2007; Mazurowski y otros, 2008; Peng y King, 2008]. Es importante resaltar que la clase minoritaria representa la mayoría de las veces el concepto de interés y por tanto es donde reside el conocimiento más novedoso.

El problema en la clasificación de bases de datos no balanceadas es que los algoritmos de clasificación tienen tendencia a describir la clase mayoritaria. Esto ocurre puesto que el clasificador intenta reducir el error global, y éste no tiene en cuenta la distribución de los datos. Supongamos que tenemos datos médicos de características de pacientes en los que las clases son padecer o no una determinada enfermedad, de forma que el 95 % de los pacientes están sanos y el 5 % padece dicha enfermedad. Si tenemos un clasificador y medimos su eficiencia con las fórmulas de las ecuaciones 3.2 o 3.3, el modelo puede obtener buenos resultados en cuanto a precisión (las clases mayoritarias las clasifica bien) pero puede resultar inútil al no clasificar bien las clases minoritarias. De lo anterior se deduce que dicha medida no es la apropiada en un escenario donde las clases no están balanceadas. Necesitamos que los clasificadores sean también precisos con la clase minoritaria.

Otro problema añadido (figura 4.1), que aparece en estos escenarios de datos no balanceados, es la existencia de algunos ejemplos de la clase minoritaria en el entorno de la clase mayoritaria, lo cual hace que las clases no sean disjuntas (aunque en poco grado) o que exista solapamiento (mayor grado) entre clases que es lo que más dificulta el aprendizaje del algoritmo

de clasificación [Batista y otros, 2004; García y otros, 2008; Jo y Japkowicz, 2004; Weiss y Provost, 2003]. También en [Japkowicz y Stephen, 2002] se estudia el efecto de las clases no balanceadas en otro tipo de modelos distintos a los árboles de decisión, como son las redes neuronales y máquinas de soporte vectorial.



(a) *No Disjuntas* (b) *Clases Solapadas*

Figura 4.1: *Problemas en las clases no balanceadas*

En los problemas no balanceados es mejor tratar la precisión del clasificador en cada una de las clases de modo independiente. De esta forma a partir de la matriz de confusión de la tabla 3.1, se pueden extraer las siguientes medidas:

- Tasa de verdaderos positivos $VP_{tasa} = \frac{VP}{VP+FN}$, definida como la fracción de ejemplos positivos predichos correctamente por el modelo.
- Tasa de verdaderos negativos: definida como la fracción de ejemplos negativos predichos correctamente por el modelo,

  $VN_{tasa} = \frac{VN}{VN+FP}$.

- Tasa de falsos positivos $FP_{tasa} = \frac{FP}{VN+FP}$, definida como el porcentaje de casos negativos mal clasificados.

- Tasa de falsos negativos $FN_{tasa} = \frac{FN}{VP+FN}$, definida como el porcentaje de casos positivos mal clasificados.

En la literatura [Tan y otros, 2006] se considera que la clase minoritaria es la clase P y la mayoritaria la N. Para los problemas de datos no balanceados, se suelen utilizar como medidas la tasa de aciertos positivos a la que también se le conoce como sensibilidad (*sensivity*) y la tasa de aciertos negativos denominada especificidad (*specificity*). De esta forma el objetivo del aprendizaje es minimizar la tasa de falsos positivos y falsos negativos o de forma análoga, maximizar la tasa de verdaderos positivos y verdaderos negativos.

En [Barandela y otros, 2003] se utiliza como métrica la *Media Geométrica* (MG) (ecuación 4.1) de las tasas individuales de acierto. Esta medida tiene en cuenta la correcta clasificación de ambas clases y no solo de la minoritaria.

$$MG = \sqrt{\frac{TP}{TP+FN} \cdot \frac{TN}{FP+TN}} \tag{4.1}$$

Existen otras medidas también derivadas de la matriz de confusión tales como las basadas en la curva ROC (*Receiver Operating Characteristic*), que representa un compromiso entre las proporciones de verdaderos positivos y falsos positivos. El área bajo la curva (AUC) se acepta como una medida del comportamiento del clasificador [Bradley, 1997]. En otros casos se intenta asociar un coste a los errores cometidos y se intenta minimizar dicho coste, en [Domingos, 1999] se utiliza una matriz de coste y en [Drummond y Holte,

2000] una curva de coste.

En la literatura especializada, hay autores que manejan todos los conjuntos no balanceados como un todo [Barandela y otros, 2003; Batista y otros, 2004; Chen y otros, 2008]. En otros casos, los conjuntos se organizan de acuerdo a su *índice de desbalanceo* (IR) [Orriols-Puig y Bernadó-Mansilla, 2009], que se define como la proporción del número de instancias entre la clase mayoritaria y la minoritaria.

4.2. Métodos de pre-procesamiento de datos no balanceados

Existen distintas aproximaciones que intentan manejar los problemas asociados con las bases de datos no balanceadas. Dichos enfoques se pueden clasificar en dos grupos según su relación con el clasificador:

- aproximaciones internas que crean algoritmos de clasificación nuevos o modifican los existentes de forma que sean éstos los que tengan en cuenta el desbalanceo de clases [Barandela y otros, 2003; Weiss y Provost, 2003; Wu y Chang, 2005; Xu y otros, 2007]. Además, otros algoritmos incorporan en su aprendizaje que las soluciones tengan asociado un coste (*cost-sentitive*), de tal manera que asocian un mayor coste a la mala clasificación de la clase minoritaria y centran su objetivo en minimizar el coste [Domingos, 1999; Sun y otros, 2007; Zhou y Liu, 2006], y
- aproximaciones externas al clasificador que pre-procesan los datos para poder disminuir el efecto causado por las clases desbalanceadas [Cha-

wla y otros, 2002; Batista y otros, 2004; Estabrooks y otros, 2004].

La ventaja que presentan las aproximaciones basadas en el pre-procesamiento de los datos es que son más versátiles y su uso es independiente del clasificador que se utilice [Chawla y otros, 2008].

Nos vamos a centrar en los métodos de pre-procesamiento que intentan ajustar las distribuciones de las clases en los datos de entrenamiento. En [Batista y otros, 2004], los métodos se clasifican en:

- **Métodos de bajo-muestreo** (*under-sampling*): crean un subconjunto de los datos originales de forma que se eliminan ejemplos de la clase mayoritaria. En este conjunto de métodos se encuentran:
  - *Random under-sampling*: es un método que elimina de forma aleatoria ejemplos de la clase mayoritaria.
  - *Tomek links* [Tomek, 1976]: este método calcula las distancias entre cada dos ejemplos de diferentes clases. Si dados dos ejemplos de diferentes clases la distancia entre ellos es menor que la que se consigue con otro ejemplo, se dice que los dos tienen un enlace si alguno es ruido o ambos caen en los bordes. Este método usado como método de *under-sampling* elimina el ejemplo de la clase mayoritaria. También se puede usar como método de limpieza en general, y en esta caso eliminaría ambos ejemplos.
  - *Condensed Nearest Neighbor Rule* (CNN) [Hart, 1968]: este algoritmo se utiliza para buscar un subconjunto de ejemplos que sea consistente con el original. Para ello utiliza el método descrito en [Kubat y Matwin, 1997].

- *One-sided selection* (OSS) [Kubat y Matwin, 1997]: este método resulta de la aplicación de *Tomek links* seguido de la aplicación de CNN. Con el primero elimina ejemplos de la clase mayoritaria que están en los bordes y con la aplicación del segundo elimina ejemplos que están distantes de los bordes de decisión.
- *CNN + Tomek links* [Batista y otros, 2004]: es similar al anterior pero aplica los métodos en orden inverso.
- *Neighborhood Cleaning Rule* (NCL) [Laurikkala, 2001]: usa el método ENN [Wilson, 1972] para eliminar sólo ejemplos de la clase mayoritaria.

▪ **Métodos de sobre-muestreo**(*over-sampling*): crean un nuevo conjunto a partir del original incluyendo nuevos ejemplos de la clase minoritaria. Los nuevos ejemplos puede ser duplicados de otros existentes o de nueva creación.

- *Random over-sampling:* consigue el balanceo replicando de forma aleatoria ejemplos de la clase minoritaria.
- *Synthetic Minority Over-sampling Technique* (SMOTE) [Chawla y otros, 2002], es un método que consiste en formar nuevos ejemplos de la clase minoritaria mediante interpolación de ejemplos de dicha clase que están juntos. De esta forma se evita el problema de sobre-aprendizaje y que ejemplos de la clase minoritaria se introduzcan en el espacio de la clase mayoritaria.

▪ **Métodos híbridos:** combinan las dos aproximaciones anteriores de forma que eliminan algunos ejemplos de clase mayoritaria y añaden nuevos ejemplos de la clase minoritaria.

- *SMOTE + Tomek links* [Batista y otros, 2004]. Aunque tras aplicar el método de over-sampling las clases quedan balanceadas, puede haber problemas de invasión del espacio de la clase minoritaria con respecto a la mayoritaria. La idea es aplicar al conjunto nuevo creado con SMOTE el método *Tomek links* para eliminar ejemplos de ambas clases de forma que se evite solapamiento entre ellas.

- *SMOTE + ENN* [Batista y otros, 2004]. La idea de este método es similar a la del anterior. ENN tiende a eliminar más ejemplos que *Tomek links*. La diferencia con NCL es que este último se utiliza sólo para eliminar ejemplos de la clase minoritaria y ENN elimina ejemplos de ambas.

Distintos autores están de acuerdo en que los métodos aleatorios tanto de sobre-muestreo como de bajo-muestreo (*random under-sampling y random over-sampling*) presentan algunos inconvenientes. Los métodos aleatorios de sobre-muestreo pueden conducir al sobre-aprendizaje al hacer copias exactas de ejemplos de la clase minoritaria, mientras que los aleatorios de bajo-muestreo pueden eliminar ejemplos significativos. Para evitar dichos problemas el resto de métodos usan ciertas heurísticas.

En [Batista y otros, 2004] se estudia el comportamiento de los distintos métodos de pre-procesamiento para un algoritmo de inducción de reglas (C4.5). En [Fernández y otros, 2008] dicho estudio se realiza para un sistema de clasificación basado en reglas difusas. En dichos trabajos se concluye que los métodos de sobre-muestreo funcionan bien, y en particular los de la familia de SMOTE.

En base a los estudios anteriores, en este capítulo se va a utilizar el método SMOTE como método de sobre-muestreo para conseguir un adecuado balance entre las clases. En dicho método, se aumentan los ejemplos de la clase minoritaria introduciendo nuevos ejemplos sintéticos a partir de ejemplos que están próximos. Los nuevos ejemplos se obtienen a lo largo de los segmentos de líneas que unen un ejemplo con otros cercanos pertenecientes a la clase minoritaria. Se generarán tantos ejemplos sintéticos como vecinos se consideren. Dado un ejemplo se seleccionan k vecinos suyos aleatoriamente cuando lo que se quiere obtener son k nuevos ejemplos sintéticos. Este proceso se ilustra en la figura 4.2, donde x_i es el ejemplo seleccionado, x_{i1} to x_{i4} son los vecinos más cercanos elegidos y r_1 to r_4 los nuevos ejemplos sintéticos creados mediante interpolación aleatoria.

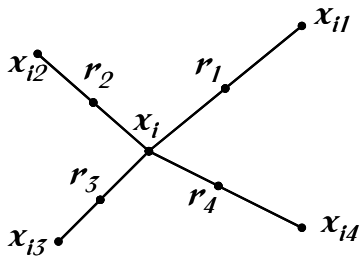


Figura 4.2: *Creación de ejemplos sintéticos mediante SMOTE*

Los ejemplos sintéticos se generan de la siguiente forma: se calcula la diferencia entre el ejemplo (vector de características) seleccionado y su vecino más cercano. Se multiplica dicha diferencia por un número aleatorio entre 0 y 1, y se suma esta cantidad al ejemplo considerado. Esto hace que se seleccione un punto aleatorio en el segmento de la linea que une los ejemplos considerados. Esta aproximación fuerza a que la región de decisión de la clase minoritaria sea más general.

La idea principal de formar nuevos ejemplos de la clase minoritaria interpolando ejemplos de la misma que están juntos, evita el problema del sobre-aprendizaje y que ejemplos de la clase minoritaria se introduzcan en el espacio de la clase mayoritaria.

4.3. Redes de funciones de base radial con datos no balanceados

Dentro del campo de las RNFNs no está muy estudiado el problema de clasificación con bases de datos no balanceadas. A continuación se describen los trabajos existentes.

En [Alejo y otros, 2007] se describe un método que trata internamente el problema de desbalanceo de clases. Para ello usa una función de costo en el proceso de entrenamiento, que intenta compensar la clase minoritaria, y estrategias para reducir el impacto de la función de costo sobre la distribución de probabilidad de los datos. Las RBFNs que utiliza en el estudio se entrenan mediante un algoritmo de *Back-propagation*.

En [Al-Haddad y otros, 2000] se realiza un estudio para evaluar el efecto del tamaño del conjunto de datos de entrenamiento y del índice de desbalanceo de los datos, sobre la precisión en un problema de clasificación de microalgas. El desbalanceo afecta más a medida que el número de especies a identificar es más grande.

Hay también aproximaciones externas para resolver el problema. En [Murhphey y Guo, 2004] se usan los métodos de pre-procesamiento de sobre-muestreo y *Snowball Training* y los resultados se evalúan con tres arqui-

tecturas diferentes: *MLP Back-Propagation*, RBFN y *Fuzzy* ARTMAP. En [Padmaja y otros, 2007] se usa SMOTE como método de pre-procesamiento y se consiguen buenos resultados con RBFNs aplicadas a problemas de detección de fraudes. En [Li y otros, 2006] se utiliza un método de bajo-muestreo para eliminar patrones de la clase mayoritaria, antes de aplicar la RBFN. Padmaja en [Padmaja y otros, 2008] propone un filtro para seleccionar ejemplos de la clase mayoritaria, aquellos que caen fuera de los *clusters* realizados con la clase minoritaria; para realizar la experimentación utiliza distintos clasificadores entre ellos una RBFN.

En [Wu y Chow, 2004; Rui y Minghu, 2008] se encuentran ejemplos del problema de desbalanceo de clases tratado con diferentes modelos de redes neuronales, entre los que se incluye una RBFN, en el primer caso los métodos se aplican sobre datos de detección de fallos en máquinas y en el segundo caso a clasificación de textos chinos, aquí se utiliza un método de selección de características. En [Zhao, 2009] se propone un nuevo modelo de red neuronal para resolver problemas de multi-clasificación sobre conjuntos de datos no balanceados, los resultados se comparan, con un modelo de RBFN entre otros.

El análisis de la bibliografía existente denota que el problema de la clasificación no balanceada con RBFNs se ha abordado sólo desde el punto de vista de la aplicación a problemas reales concretos, no desde el punto de vista del análisis del algoritmo de diseño de RBFNs y determinación de su robustez frente a este tipo de problemas o tipo de método de pre-procesamiento más adecuado. El método que se ha utilizado con mayor frecuencia es SMOTE, algoritmo considerado en este capítulo.

4.4. Resultados experimentales

Para el estudio experimental, se han seleccionado cuarenta y cuatro bases de datos del UCI [Asuncion y Newman, 2007b], con diferentes grados de desbalanceo (IR) y en las que existe sólo una clase positiva y otra negativa.

En la tabla 4.1 se muestra el número de ejemplos (#Ej.), número de atributos (#Atbs.), el nombre de cada una de las clases (minoritaria y mayoritaria), la distribución del atributo que representa la clase y el IR. La tabla está ordenada de forma que se presentan las bases de datos de forma creciente respecto a su IR, desde un bajo a un alto desbalanceo.

Para realizar el estudio comparativo se utiliza validación cruzada de orden 5, de forma que hay cinco particiones para entrenamiento y cinco para test. En cada partición el 80 % de los datos son para entrenamiento y el 20 % para test.

En este estudio experimental, los algoritmos de clasificación se van a aplicar sobre los datos originales y se analizará el comportamiento del algoritmo CO^{2}RBFN (en comparación con otros) frente a datos pertenecientes a clases no blanceadas.

Posteriormente, para intentar reducir el efecto provocado por el desbalanceo, se utiliza el método de pre-procesamiento SMOTE [Chawla y otros, 2002], considerando sólo 1 vecino más cercano para generar ejemplos sintéticos y balancear ambas clases de forma que se consiga una distribución del 50 % para cada clase.

Los resultados obtenidos por CO^{2}RBFN se van a comparar con los resultados obtenidos mediante otros algoritmos alternativos dentro del

Tabla 4.1: *Descripción de las bases de datos no balanceadas*

Base datos	#Ex.	#Atbs.	Clase(min., may.)	%Clase(min., may.)	IR
Glass1	214	9	(build-win-non-foat-proc, remainder)	(35.51, 64.49)	1.82
Ecoli0vs1	220	7	(im, cp)	(35.00, 65.00)	1.86
Wisconsin	683	9	(malignant, benign)	(35.00, 65.00)	1.86
Pima	768	8	(tested-positive, tested-negative)	(34.84, 66.16)	1.90
Iris0	150	4	(Iris-Setosa, remainder)	(33.33, 66.67)	2.00
Glass0	214	9	(build-win-float-proc, remainder)	(32.71, 67.29)	2.06
Yeast1	1484	8	(nuc, remainder)	(28.91, 71.09)	2.46
Vehicle1	846	18	(Saab, remainder)	(28.37, 71.63)	2.52
Vehicle2	846	18	(Bus, remainder)	(28.37, 71.63)	2.52
Vehicle3	846	18	(Opel, remainder)	(28.37, 71.63)	2.52
Haberman	306	3	(Die, Survive)	(27.42, 73.58)	2.68
Glass0123vs456	214	9	(non-window glass, remainder)	(23.83, 76.17)	3.19
Vehicle0	846	18	(Van, remainder)	(23.64, 76.36)	3.23
Ecoli1	336	7	(im, remainder)	(22.92, 77.08)	3.36
New-thyroid2	215	5	(hypo, remainder)	(16.89, 83.11)	4.92
New-thyroid1	215	5	(hyper, remainder)	(16.28, 83.72)	5.14
Ecoli2	336	7	(pp, remainder)	(15.48, 84.52)	5.46
Segment0	2308	19	(brickface, remainder)	(14.26, 85.74)	6.01
Glass6	214	9	(headlamps, remainder)	(13.55, 86.45)	6.38
Yeast3	1484	8	(me3, remainder)	(10.98, 89.02)	8.11
Ecoli3	336	7	(imU, remainder)	(10.88, 89.12)	8.19
Page-blocks0	5472	10	(remainder, text)	(10.23, 89.77)	8.77
Yeast2vs4	514	8	(cyt, me2)	(9.92, 90.08)	9.08
Yeast05679vs4	528	8	(me2, mit, me3, exc, vac, erl)	(9.66, 90.34)	9.35
Vowel0	988	13	(hid, remainder)	(9.01, 90.99)	10.10
Glass016vs2	192	9	(ve-win-float-proc, build-win-float-proc, build-win-non-float-proc, headlamps)	(8.89, 91.11)	10.29
Glass2	214	9	(Ve-win-float-proc, remainder)	(8.78, 91.22)	10.39
Ecoli4	336	7	(om, remainder)	(6.74, 93.26)	13.84
Yeast1vs7	459	8	(nuc, vac)	(6.72, 93.28)	13.87
Shuttle0vs4	1829	9	(Rad Flow, Bypass)	(6.72, 93.28)	13.87
Glass4	214	9	(containers, remainder)	(6.07, 93.93)	15.47
Page-blocks13vs2	472	10	(graphic, horiz.line, picture)	(5.93, 94.07)	15.85
Abalone9vs18	731	8	(18, 9)	(5.65, 94.25)	16.68
Glass016vs5	184	9	(tableware, build-win-float-proc, build-win-non-float-proc, headlamps)	(4.89, 95.11)	19.44
Shuttle2vs4	129	9	(Fpv Open, Bypass)	(4.65, 95.35)	20.5
Yeast1458vs7	693	8	(vac, nuc, me2, me3, pox)	(4.33, 95.67)	22.10
Glass5	214	9	(tableware, remainder)	(4.20, 95.80)	22.81
Yeast2vs8	482	8	(pox, cyt)	(4.15, 95.85)	23.10
Yeast4	1484	8	(me2, remainder)	(3.43, 96.57)	28.41
Yeast1289vs7	947	8	(vac, nuc, cyt, pox, erl)	(3.17, 96.83)	30.56
Yeast5	1484	8	(me1, remainder)	(2.96, 97.04)	32.78
Ecoli0137vs26	281	7	(pp, imL, cp, im, imU, imS)	(2.49, 97.51)	39.15
Yeast6	1484	8	(exc, remainder)	(2.49, 97.51)	39.15
Abalone19	4174	8	(19, remainder)	(0.77, 99.23)	128.87

paradigma de diseño de RBFNs y de otros modelos de redes neuronales tales como Perceptrón Multicapa y LVQ. Específicamente, se consideran los cinco algoritmos siguientes:

- LVQ. Construye una red de tipo LVQ (*Learning Vector Quantization Network*) formada por un conjunto de neuronas, que representan el prototipo más significativo de cada clase después del entrenamiento. De esta forma, la clase de cada instancia se predice como la clase de la neurona más cercana, siguiendo el modelo KNN [Bezdek y Kuncheva, 2001].

  El algoritmo empieza seleccionando un conjunto aleatorio de n_p prototipos, de forma que cada clase esté representada por al menos un prototipo.

  A continuación el algoritmo itera modificando las neuronas. En cada iteración se barajan los ejemplos del conjunto de entrenamiento y se van tomando uno a uno. Si la neurona más próxima a dicho ejemplo tiene la misma clase que dicho ejemplo, entonces la mueve hacia el vector de entrada y si la clase es diferente entonces la aleja de él.

  La actualización se realiza a través de un factor ponderado α, que se va haciendo menor a lo largo de la ejecución de algoritmo. El algoritmo acaba cuando no se produce ningún cambio sobre las neuronas o se alcanza una cantidad T de iteraciones.

- MLP-Back: algoritmo para el diseño de redes Perceptrón Multicapa que usa el algoritmo *Backpropagation* para el aprendizaje [Rojas y Feldman, 1996] (descrito en la sección 3.5 del capítulo 3).

- MLP-Grad: algoritmo para el diseño de redes Perceptrón Multicapa que usa el algoritmo *del Gradiente conjugado* [Widrow y Lehr, 1990; Moller, 1993] para el entrenamiento de la red. En general, las técnicas basadas en el gradiente son métodos [Lipmann, 1987] clásicos en redes

neuronales para la determinación de los pesos. Su funcionamiento consiste en ir presentando a la red las distintas muestras del conjunto de entrenamiento, para cada muestra se obtiene la diferencia entre la salida de la red y la salida real. Esta diferencia es la información de gradiente que se obtiene y se utilizará para determinar el cambio en los pesos de forma que la salida de la red se aproxime a la real. Comparada con la técnica de gradiente descendiente, la del gradiente conjugado toma un camino más directo hacia el conjunto de pesos óptimo.

- RBFN-Decr: algoritmo para el diseño de RBFNs basado en un esquema decremental [Broomhead y Lowe, 1988] (descrito en la sección 2.3.4 del capítulo 2).

- RBFN-Incr: algoritmo para el diseño de RBFNs basado en un esquema incremental [Plat, 1991] (descrito en la sección 2.3.4 del capítulo 2).

Las ejecuciones de todos los métodos anteriores se han realizado con la herramienta KEEL [Alcalá-Fdez y otros, 2009] y los valores utilizados para los parámetros de los algoritmos son los recomendados por los autores de los mismos en la bibliografía. En la tabla 4.2 se muestran los valores de los parámetros utilizados por CO^2RBFN. Los valores de los parámetros utilizados por el resto de algoritmos usados en la experimentación se muestran en el apéndice C.

La medida para calcular la media de aciertos es la métrica MG. El número de repeticiones para los algoritmos se ha fijado a 5.

Tabla 4.2: *Parámetros de CO^2RBFN*

Parámetro	*Valor*
Generaciones del ciclo principal	200
Número de RBFs	5

4.5. Análisis de resultados

En esta sección se va a comparar y analizar los resultados obtenidos mediante los diferentes algoritmos. Primero se analizan los resultados obtenidos cuando los algoritmos se aplican sobre los datos originales, sin pre-procesamiento. Posteriormente se analizarán los resultados conseguidos cuando los algoritmos se aplican a los datos pre-procesados mediante SMOTE.

Para el análisis de los resultados se aplican técnicas de test de contraste de hipótesis [García y otros, 2009a; Sheskin, 2006]. Específicamente, se van a utilizar tests no paramétricos debido a que las condiciones iniciales que se deben garantizar para poder aplicar tests paramétricos puede que no se satisfagan [Demšar, 2006].

Se utiliza el test de ranking de signos de Wilcoxon [Wilcoxon, 1945] como test no paramétrico para establecer comparaciones entre cada dos algoritmos. Para establecer comparaciones entre múltiples algoritmos se emplea el test de Iman-Davenport [Sheskin, 2006] que detecta diferencias estadísticas entre un grupo de resultados, y el test de Holm [Holm, 1979] para detectar entre qué algoritmos existen tales diferencias.

El test de Holm permite conocer si se rechaza o no una hipótesis de igualdad de medias con un nivel de confianza α. Es importante calcular el

p-valor asociado con cada comparación, que representa el nivel más bajo de confianza para que se rechace la hipótesis. De esta forma, se determina si entre dos algoritmos existen o no diferencias significativas y cuantificarlas si las hay. Se considera un nivel de confianza de α=0.05.

4.5.1. Análisis de los resultados sin pre-procesamiento de los datos

En primer lugar, se compararan los resultados obtenidos mediante los diferentes métodos aplicados sobre las bases de datos originales, sin ningún tipo de pre-procesamiento.

En la tabla 4.3 podemos observar los resultados obtenidos. Cada fila se corresponde con los resultados para una base de datos y cada columna muestra el acierto en test y la desviación típica, conseguido por cada uno de los métodos. En la última fila se muestra la media de acierto en test y su desviación típica. Como puede observarse CO^2RFBN obtiene los mejores resultados en media de todas las bases de datos así como una desviación baja, inferior al resto de métodos, lo cual indica que es un algoritmo robusto.

A continuación se aplican los test estadísticos sobre los resultados obtenidos.

En la figura 4.3 se muestra el ranking medio calculado para cada algoritmo de acuerdo con la métrica MG. Se puede observar que CO^2RBFN es el algoritmo que tiene mejor valor en el ranking (obtiene el valor más bajo), mientras que LVQ y RBFN-Decr obtienen las peores posiciones en la ordenación. MLP-Back, MLP-Grad y RBFN-Incr tienen comportamientos similares entre sí.

Tabla 4.3: *Resultados experimentales sin pre-procesamiento*

Base datos	CO^{2}RBFN	LVQ	MLP-Back	MLP-Grad	RBFN-Decr	RBFN-Incr
glass1	69.30 ± 6.16	62.27 ± 11.19	56.32 ± 10.53	71.28 ± 4.47	67.99 ± 10.07	73.23 ± 8.10
ecoli0vs1	97.03 ± 2.80	93.68 ± 3.87	97.59 ± 1.99	0.00 ± 0.00	94.71 ± 4.23	92.93 ± 8.40
wisconsin	97.29 ± 0.77	92.65 ± 5.78	96.23 ± 1.15	93.89 ± 2.23	95.31 ± 1.88	95.55 ± 1.89
pima	71.73 ± 4.30	56.95 ± 10.07	71.10 ± 4.80	67.72 ± 4.14	65.00 ± 6.63	62.03 ± 5.48
iris0	99.79 ± 1.01	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	99.70 ± 0.82	99.80 ± 0.69
glass0	75.69 ± 7.12	69.58 ± 7.76	64.68 ± 8.60	80.08 ± 6.44	68.01 ± 15.37	70.76 ± 12.06
yeast1	70.77 ± 3.58	53.62 ± 8.61	65.86 ± 4.92	63.36 ± 2.91	39.71 ± 18.55	46.94 ± 15.37
vehicle1	65.57 ± 2.98	52.18 ± 13.27	60.85 ± 13.43	77.10 ± 4.55	52.45 ± 18.91	59.36 ± 6.46
vehicle2	83.37 ± 3.83	69.53 ± 8.85	63.86 ± 11.07	97.65 ± 1.71	72.50 ± 8.73	91.23 ± 3.46
vehicle3	66.97 ± 3.46	52.75 ± 8.07	64.15 ± 4.84	74.68 ± 5.7.0	56.27 ± 10.93	54.42 ± 6.37
haberman	61.21 ± 7.26	42.08 ± 12.97	61.10 ± 8.59	45.74 ± 6.85	47.62 ± 12.55	47.99 ± 9.51
glass0123vs456	92.27 ± 3.27	85.88 ± 8.21	91.55 ± 3.44	86.15 ± 4.52	87.74 ± 7.08	93.63 ± 4.39
vehicle0	89.12 ± 4.55	67.90 ± 12.22	68.54 ± 11.90	95.70 ± 2.43	79.21 ± 14.2	94.09 ± 2.70
ecoli1	88.65 ± 4.42	77.00 ± 10.51	85.05 ± 3.83	66.92 ± 34.41	79.20 ± 13.93	82.17 ± 8.92
newthyroid2	98.40 ± 3.72	78.73 ± 25.19	84.16 ± 18.26	95.93 ± 4.32	90.89 ± 6.38	98.86 ± 2.32
newthyroid1	98.02 ± 3.05	84.86 ± 12.48	82.81 ± 19.05	96.49 ± 3.78	92.98 ± 6.21	97.99 ± 3.73
ecoli2	92.02 ± 3.40	85.89 ± 11.71	81.11 ± 8.22	68.50 ± 34.87	77.11 ± 15.43	78.67 ± 14.51
segment0	96.05 ± 2.20	82.61 ± 9.12	1.71 ± 8.36	0.00 ± 0.00	59.55 ± 23.33	98.22 ± 1.46
glass6	87.07 ± 7.38	85.75 ± 8.04	90.40 ± 6.42	88.60 ± 11.01	88.12 ± 8.95	91.23 ± 6.60
yeast3	89.51 ± 2.58	67.78 ± 18.86	74.57 ± 5.51	81.82 ± 6.51	32.65 ± 24.84	67.40 ± 14.61
ecoli3	87.02 ± 7.65	74.24 ± 19.72	59.14 ± 33.33	54.32 ± 29.03	66.64 ± 23.66	68.43 ± 29.52
pageblocks0	86.07 ± 2.40	55.39 ± 15.72	73.72 ± 8.05	85.01 ± 1.85	65.00 ± 15.38	86.58 ± 2.43
yeast2vs4	86.87 ± 4.90	73.54 ± 11.47	72.81 ± 6.48	65.44 ± 34.06	67.13 ± 25.15	70.71 ± 19.93
yeast05679vs4	77.06 ± 8.20	64.12 ± 12.75	66.31 ± 10.08	58.25 ± 11.16	52.06 ± 18.50	40.14 ± 20.82
vowel0	87.03 ± 5.86	51.52 ± 14.27	68.74 ± 6.33	98.97 ± 1.79	83.81 ± 12.29	99.43 ± 1.39
glass016vs2	47.27 ± 19.50	32.94 ± 31.01	23.99 ± 27.91	46.9 ± 29.85	21.98 ± 26.97	29.92 ± 31.83
glass2	57.23 ± 15.11	27.58 ± 26.77	18.89 ± 21.63	28.6 ± 30.78	23.86 ± 27.24	26.67 ± 28.65
ecoli4	90.96 ± 6.23	78.49 ± 15.70	62.30 ± 32.00	69.09 ± 34.55	82.12 ± 15.88	80.65 ± 15.93
shuttlec0vsc4	69.69 ± 12.50	96.13 ± 10.16	82.64 ± 24.84	99.60 ± 0.81	98.53 ± 5.42	99.75 ± 0.55
yeast1vs7	99.67 ± 0.80	36.75 ± 28.45	53.8 ± 16.04	38.51 ± 23.65	35.90 ± 23.42	6.53 ± 14.97
glass4	81.84 ± 14.07	65.47 ± 24.22	79.66 ± 18.82	71.70 ± 25.45	72.06 ± 34.5	94.89 ± 8.68
pageblocks13vs4	90.15 ± 7.70	74.97 ± 11.49	88.21 ± 9.29	98.66 ± 4.04	45.81 ± 27.89	83.41 ± 11.33
abalone918	75.70 ± 9.52	27.54 ± 22.45	55.56 ± 18.66	62.98 ± 11.28	32.60 ± 15.58	20.62 ± 16.85
glass016vs5	62.40 ± 40.00	35.92 ± 41.56	87.52 ± 4.55	79.55 ± 35.97	71.78 ± 29.02	80.06 ± 21.55
shuttlec2vsc4	93.59 ± 11.70	70.14 ± 40.99	80.28 ± 18.41	94.66 ± 10.51	55.25 ± 49.00	81.59 ± 36.05
yeast1458vs7	55.02 ± 14.70	21.76 ± 21.34	45.10 ± 17.16	8.75 ± 17.76	3.89 ± 13.45	0.00 ± 0.00
glass5	57.30 ± 43.95	21.89 ± 35.73	85.28 ± 5.84	88.51 ± 21.88	66.02 ± 42.34	67.24 ± 39.76
yeast2vs8	71.88 ± 14.13	60.66 ± 29.22	64.18 ± 15.59	70.72 ± 19.06	71.76 ± 13.96	72.79 ± 13.42
yeast4	77.33 ± 10.86	45.02 ± 20.08	60.29 ± 14.76	44.57 ± 8.61	30.71 ± 26.18	12.94 ± 17.64
yeast1289vs7	55.19 ± 22.50	26.13 ± 22.28	38.00 ± 23.10	42.08 ± 23.47	18.45 ± 23.43	0.00 ± 0.00
yeast5	94.12 ± 4.40	76.93 ± 23.64	59.79 ± 26.63	63.98 ± 14.45	35.99 ± 29.21	44.80 ± 23.77
ecoli0137vs26	70.50 ± 29.50	52.16 ± 46.90	61.92 ± 35.45	58.83 ± 48.37	68.70 ± 40.14	69.50 ± 40.62
yeast6	83.27 ± 10.45	64.85 ± 28.28	60.33 ± 16.06	53.23 ± 20.1	15.33 ± 25.68	28.34 ± 30.40
abalone19	50.12 ± 21.81	11.42 ± 20.60	49.88 ± 13.87	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
Media	**79.48 ± 9.46**	61.53 ± 17.76	67.27 ± 13.18	66.69 ± 13.62	59.82 ± 18.03	65.03 ± 12.8

Una vez vistos los rankings obtenidos, se analiza si existen diferencias significativas entre los resultados de los algoritmos usados en el estudio

Algoritmo	Ranking
CO^2RBFN	**1.932**
LVQ	4.455
MLP-Back	3.477
MLP-Grad	3.273
RBFN-Decr	4.50
RBFN-Incr	3.364

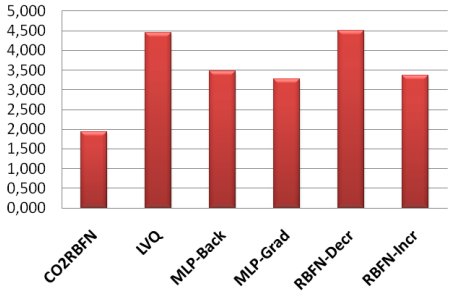


Figura 4.3: *Ranking de acuerdo con la MG, resultados sin pre-procesamiento. El algoritmo mejor es el que consigue el valor más bajo*

experimental, para ello utiliza el test de Iman-Davenport. El estadístico obtenido por el test de Iman-Davenport es 14.624. El valor crítico de la distribución F_F con 5 y 215 grados de libertad es de 2.256. Al ser menor el valor crítico que el estadístico obtenido se deduce que se rechaza la hipótesis de igualdad de medias, es decir, existen diferencias significativas entre los métodos.

La tabla 4.4 muestra los resultados del test de Iman-Davenport.

Tabla 4.4: *Resultados del test de Iman-Davenport en cuanto a la precisión en clasificación sin pre-procesamiento*

Test	Estadístico	Valor F_F	Hipótesis nula (α=0.05)
Iman-Davenport	14.624	2.256	Rechazada

Una vez visto que sí existen diferencias significativas entre los resultados obtenidos por los diferentes algoritmos, se va a aplicar el test de Holm para comparar el mejor método del ranking, en este caso CO^2RBFN, con el resto de métodos, de esta forma se intenta ver con cuáles de ellos tiene diferencias significativas. Los resultados se muestran en la tabla 4.5. En

todos los casos ocurre que el valor p_i es menor que el correspondiente $\alpha \setminus i$, lo que implica que se rechaza la hipótesis de igualdad de medias, es decir, existen diferencias significativas entre el algoritmo de control, CO^{2}RBFN, y el resto de algoritmos.

Tabla 4.5: *Resultados del test de Holm aplicado a las bases de datos sin pre-procesamiento. CO^{2}RBFN es el método de control*

i	Algoritmo	z	p	$\alpha \setminus i$	Hipótesis nula (α=0.05)
5	RBFN-Decr	6.439	1.204E-10	0.01	Rechazada
4	LVQ	6.325	2.535E-10	0.0125	Rechazada
3	RBFN-Back	3.875	1.068E-4	0.017	Rechazada
2	MLP-Incr	3.590	3.310E-4	0.025	Rechazada
1	MLP-Grad	3.362	7.743E-4	0.05	Rechazada

Como se observa en la tabla 4.5 se rechaza en todos los casos la hipótesis nula de igualdad, con lo que se puede concluir que la eficiencia de CO^{2}RBFN supera significativamente a la de los otros métodos, en este caso con los resultados obtenidos al procesar las bases de datos originales, sin pre-procesamiento.

4.5.2. Análisis de los resultados con SMOTE como algoritmo de pre-procesamiento

En esta sección se va a realizar un pre-procesamiento con SMOTE de las bases de datos antes de aplicar los métodos de clasificación. Los resultados de la experimentación sobre las bases de datos pre-procesadas se muestran en la tabla 4.7.

En un primer estudio sobre los resultados obtenidos se ha comprobado que los resultados obtenidos por cada método individualmente sin la

aplicación de SMOTE (tabla 4.3) son peores que los obtenidos tras la aplicación de SMOTE (tabla 4.7). Para realizar un análisis más formal de los resultados, se aplica el test de ranking de signos de Wilcoxon para comparar los resultados de cada método obtenidos con los datos sin pre-procesamiento, frente a los resultados obtenidos por el mismo método aplicado sobre los datos a los que se les ha aplicado SMOTE.

En la tabla 4.6 se muestran los resultados obtenidos, de los que se puede deducir que el pre-proceso de datos es necesario cuando se tratan bases de datos no balanceadas, así el resultado de todos los métodos cuando se ha aplicado SMOTE mejora con respecto a los resultados de ellos mismos sobre los datos originales.

Tabla 4.6: *Test de Wilcoxon para comparar el uso de SMOTE frente al no pre-procesamiento de los datos.* R^+ *se corresponde con los valores con SMOTE y* R^- *los resultados con los datos originales*

Comparación	R^+	R^-	p-valor	Hipótesis nula ($\alpha = 0.05$)
CO^2RBFN+SMOTE vs. CO^2RBFN	856.0	134.0	0.000	Rechazada para CO^2RBFN+SMOTE
LVQ+SMOTE vs LVQ	969.5	20.5	0.000	Rechazada para LVQ+SMOTE
MLP-Back+SMOTE vs MLP-Back	853.5	136.5	0.000	Rechazada para MLP-Back+SMOTE
MLPGrad+SMOTE vs MLPGrad	873.0	117.0	0.000	Rechazada para MLPGrad+SMOTE
RBFNDecr+SMOTE vs RBFNDecr	816.0	174.0	0.000	Rechazada para RBFNDecr+SMOTE
RBFNIncr+SMOTE vs RBFNIncr	811.0	179.0	0.000	Rechazada para RBFNIncr+SMOTE

Como se ve en la tabla 4.6, el comportamiento de todos los métodos mejora cuando se aplican sobre los datos pre-procesados.

A continuación se analiza la eficiencia de los métodos aplicados a las bases de datos pre-procesadas.

En la figura 4.4 se muestra el ranking medio calculado para cada algoritmo de acuerdo con la métrica MG. Se puede observar que CO^2RBFN

Tabla 4.7: *Resultados experimentales con SMOTE*

Base datos	CO^2RBFN	LVQ	MLP-Back	MLP-Grad	RBFN-Decr	RBFN-Incr
glass1	69.86 ± 6.44	65.33 ± 7.63	57.53 ± 7.14	71.93 ± 5.98	67.95 ± 14.41	73.78 ± 8.91
ecoli0vs1	96.18 ± 2.96	93.05 ± 3.63	97.22 ± 1.70	0.00 ± 0.00	94.69 ± 4.71	95.13 ± 5.26
wisconsin	97.26 ± 0.85	94.02 ± 5.18	90.52 ± 17.72	95.31 ± 1.57	94.41 ± 4.95	94.16 ± 5.07
pima	72.56 ± 3.71	59.63 ± 5.77	69.89 ± 6.63	69.62 ± 4.85	63.97 ± 7.71	61.95 ± 4.45
iris0	99.90 ± 0.50	100.00 ± 0.00	100.00 ± 0.00	100.00 ± 0.00	99.80 ± 0.69	99.69 ± 1.10
glass0	75.64 ± 6.99	67.79 ± 5.03	68.62 ± 6.21	79.53 ± 5.75	71.89 ± 9.37	76.36 ± 6.62
yeast1	70.08 ± 3.47	57.77 ± 6.70	63.16 ± 13.54	72.34 ± 3.11	34.15 ± 20.61	62.45 ± 9.80
vehicle1	69.08 ± 4.37	59.21 ± 6.01	61.19 ± 5.93	81.94 ± 3.31	61.78 ± 9.60	60.7 ± 5.03
vehicle2	87.24 ± 3.98	72.34 ± 5.75	70.18 ± 8.18	97.50 ± 1.58	73.19 ± 10.18	91.55 ± 2.98
vehicle3	69.55 ± 3.98	60.27 ± 5.96	64.82 ± 3.06	79.27 ± 3.49	61.28 ± 6.82	58.37 ± 6.79
haberman	60.21 ± 6.25	50.92 ± 8.28	56.24 ± 10.64	57.47 ± 7.29	58.04 ± 8.51	53.38 ± 5.21
glass0123vs456	93.78 ± 3.28	88.46 ± 5.58	84.03 ± 7.56	87.67 ± 5.65	88.94 ± 9.22	92.17 ± 4.33
vehicle0	92.15 ± 2.48	77.65 ± 5.05	81.4 ± 5.02	95.07 ± 2.70	73.22 ± 19.41	90.21 ± 5.11
ecoli1	87.84 ± 4.10	85.03 ± 5.72	86.85 ± 3.88	69.58 ± 35.02	81.50 ± 18.02	87.57 ± 6.31
newthyroid2	98.46 ± 2.22	91.34 ± 7.51	98.48 ± 1.12	98.82 ± 3.00	93.52 ± 5.17	99.04 ± 1.37
newthyroid1	97.54 ± 4.03	94.95 ± 4.83	97.35 ± 2.63	99.44 ± 0.69	88.98 ± 7.35	98.63 ± 2.43
ecoli2	93.14 ± 4.50	83.62 ± 7.41	87.54 ± 6.36	72.93 ± 36.58	72.68 ± 21.91	81.85 ± 15.8
segment0	97.97 ± 0.81	82.45 ± 6.08	1.99 ± 9.77	0.00 ± 0.00	62.58 ± 14.89	97.96 ± 1.15
glass6	85.93 ± 8.39	85.86 ± 7.19	88.05 ± 7.17	85.19 ± 9.20	88.16 ± 8.22	85.52 ± 9.05
yeast3	91.11 ± 2.34	82.92 ± 4.88	82.96 ± 17.18	91.65 ± 2.47	56.92 ± 29.97	87.77 ± 4.56
ecoli3	85.72 ± 7.90	82.27 ± 6.06	86.65 ± 6.09	68.36 ± 34.94	78.96 ± 20.01	86.26 ± 9.65
pageblocks0	88.60 ± 2.01	76.39 ± 4.95	78.65 ± 7.02	93.72 ± 1.05	68.33 ± 16.66	57.03 ± 13.20
yeast2vs4	87.29 ± 3.60	82.79 ± 5.09	85.61 ± 6.06	67.96 ± 34.57	69.93 ± 23.54	81.72 ± 10.33
yeast05679vs4	78.22 ± 5.10	69.77 ± 9.12	76.90 ± 5.61	75.32 ± 6.46	53.02 ± 27.34	72.19 ± 12.11
vowel0	93.77 ± 3.52	78.12 ± 6.18	84.55 ± 6.61	99.19 ± 1.73	93.14 ± 10.87	99.36 ± 1.63
glass016vs2	56.44 ± 20.80	58.93 ± 12.85	45.07 ± 26.29	62.97 ± 18.08	66.17 ± 18.50	64.96 ± 14.63
glass2	58.15 ± 25.25	58.76 ± 13.33	63.98 ± 10.68	67.51 ± 17.36	62.63 ± 17.59	66.42 ± 11.93
ecoli4	89.65 ± 6.17	92.37 ± 5.29	91.35 ± 6.65	68.62 ± 34.32	88.05 ± 18.95	93.72 ± 5.94
shuttlec0vsc4	99.58 ± 0.80	99.60 ± 0.81	99.83 ± 0.55	99.75 ± 0.66	98.27 ± 5.48	99.91 ± 0.12
yeast1vs7	99.49 ± 0.90	59.38 ± 15.49	69.75 ± 8.32	62.94 ± 12.31	42.46 ± 25.31	64.85 ± 11.41
glass4	85.45 ± 12.62	80.68 ± 20.55	87.93 ± 10.08	80.94 ± 26.7	76.64 ± 25.73	93.62 ± 8.31
pageblock13vs4	96.65 ± 3.60	93.29 ± 4.61	79.88 ± 12.93	98.50 ± 2.76	88.33 ± 18.54	86.55 ± 13.99
abalone918	77.41 ± 10.60	52.11 ± 11.77	75.41 ± 16.73	75.42 ± 11.58	53.82 ± 9.78	70.30 ± 8.91
glass016vs5	84.71 ± 31.80	85.09 ± 12.39	91.21 ± 6.84	85.16 ± 27.25	85.27 ± 12.36	83.58 ± 22.37
shuttlec2vsc4	99.51 ± 1.00	97.92 ± 4.48	97.93 ± 5.06	99.26 ± 1.28	77.14 ± 38.8	75.25 ± 38.66
yeast1458vs7	60.8 ± 11.20	52.49 ± 12.66	60.79 ± 7.76	55.12 ± 20.49	34.52 ± 22.68	52.03 ± 17.95
glass5	74.91 ± 38.45	88.73 ± 7.60	80.49 ± 30.59	77.83 ± 31.32	69.87 ± 36.50	69.82 ± 37.32
yeast2vs8	77.31 ± 12.23	69.26 ± 10.47	71.19 ± 11.67	70.06 ± 21.25	53.58 ± 25.15	74.19 ± 12.05
yeast4	78.95 ± 4.23	76.50 ± 6.58	80.92 ± 4.56	77.96 ± 8.51	59.46 ± 21.78	78.62 ± 7.27
yeast1289vs7	70.14 ± 7.50	55.15 ± 8.35	69.44 ± 5.48	60.87 ± 9.10	23.28 ± 24.96	63.19 ± 10.36
yeast5	94.69 ± 3.60	94.86 ± 2.07	91.67 ± 6.02	93.17 ± 5.09	40.41 ± 43.18	94.54 ± 3.96
ecoli0137vs26	70.09 ± 36.30	66.41 ± 34.27	54.87 ± 45.05	57.69 ± 47.13	62.74 ± 38.02	61.16 ± 43.29
yeast6	86.57 ± 8.40	76.41 ± 10.18	85.78 ± 5.61	84.59 ± 7.71	34.09 ± 36.30	83.47 ± 9.04
abalone19	70.18 ± 11.77	52.86 ± 15.69	60.46 ± 14.83	51.87 ± 11.45	54.47 ± 14.09	63.32 ± 12.10
Media	**83.36 ± 7.84**	76.20 ± 8.07	76.78 ± 9.29	75.91 ± 11.94	68.69 ± 17.81	79.19 ± 10.18

es el algoritmo que tiene mejor ranking, mientras que LVQ y RBFN-Decr obtienen los peores rankings. MLP-Back, MLP-Grad y RBFN-Incr tienen

Algoritmo	Ranking
CO^2RBFN	**2.250**
LVQ	4.364
MLP-Back	3.295
MLP-Grad	3.160
RBFN-Decr	4.591
RBFN-Incr	3.341

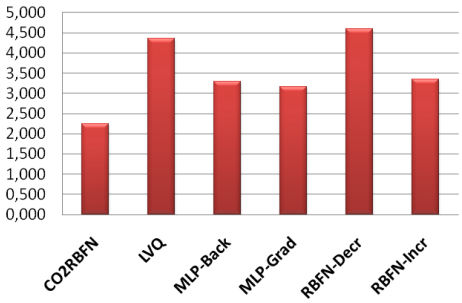


Figura 4.4: *Ranking de acuerdo con la MG utilizando SMOTE como pre-procesamiento. El algoritmo mejor es el que consigue el valor más bajo*

comportamientos similares entre sí.

Una vez vistos los rankings obtenidos, se aplica el test de Iman-Davenport. El estadístico obtenido por el test es de 11.457. El valor crítico de la distribución F_F con 5 y 215 grados de libertad es de 2.256. Al ser menor el valor crítico que el estadístico obtenido se deduce que existen diferencias significativas entre los métodos.

La tabla 4.8 muestra los resultados del test de Iman-Davenport.

Tabla 4.8: *Resultados del test de Iman-Davenport en cuanto a la precisión en clasificación usando SMOTE*

Test	Estadístico	Valor F_F	Hipótesis nula (α=0.05)
Iman-Davenport	11.457	2.256	Rechazada

A continuación se aplica el test de Holm para comparar el método que tiene el mejor ranking (CO^2RBFN) con el resto de métodos. Los resultados se muestran en la tabla 4.9, en la cual los algoritmos están ordenados con respecto al valor z obtenido. El valor p_i obtenido se compara con el valor

$\alpha \setminus i$ situado en la misma fila de la tabla, en todos los casos ocurre que el valor p_i es menor que el correspondiente $\alpha \setminus i$, lo que implica que existen diferencias significativas entre el algoritmo de control, CO^{2}RBFN, y el resto de algoritmos.

Tabla 4.9: *Test de Holm aplicado a todas las bases de datos pre-procesadas con SMOTE. CO^{2}RBFN es el método de control*

i	Algoritmo	z	p	$\alpha \setminus i$	Hipótesis nula (α=0.05)
5	RBFN-Decr	5.869	4.385E-9	0.01	Rechazada
4	LVQ	5.299	1.163E-7	0.0125	Rechazada
3	RBFN-Incr	2.735	0.006	0.017	Rechazada
2	MLP-Back	2.621	0.009	0.025	Rechazada
1	MLP-Grad	2.280	0.023	0.05	Rechazada

Como se observa en la tabla 4.9 se rechaza en todos los casos la hipótesis nula de igualdad, con lo que se puede concluir que la eficiencia de CO^{2}RBFN supera a la de los otros métodos.

4.6. Conclusiones

En este capítulo se ha aplicado CO^{2}RBFN, a la clasificación de bases de datos no balanceadas y sus resultados se han comparado con los obtenidos mediante otros algoritmos dentro del paradigma de las redes neuronales.

Comparando los resultados de los algoritmos aplicados a las bases de datos sin pre-procesar, obtenemos que el mejor comportamiento es el de CO^{2}RBFN y el análisis estadístico de los datos muestra que las diferencias con el resto de métodos son significativas. CO^{2}RBFN obtiene también los resultados con la desviación típica más baja indicando que es un método

robusto.

En una segunda fase de comparaciones, se aplican los algoritmos a las bases de datos pre-procesadas mediante SMOTE. En primer lugar se demuestra que existen diferencias significativas comparando cada método aplicado a los datos sin pre-procesar con él mismo aplicado a los datos pre-procesados mediante SMOTE, lo cual pone de manifiesto la necesidad del pre-procesamiento en este tipo de bases de datos desbalanceadas.

Por último se han comparado los resultados obtenidos por los algoritmos, aplicados sobre las bases de datos pre-procesadas mediante SMOTE, y CO^{2}RBFN vuelve a obtener los mejores resultados en la clasificación. De nuevo los test estadísticos aplicados muestran que existen diferencias significativas entre él y el resto de algoritmos.

Capítulo 5

CO^2RBFN aplicado a la predicción de series temporales

Muchas de las bases de datos están formadas por series con observaciones de carácter cronológico que normalmente se realizan de forma repetida y con la misma frecuencia. A este tipo de series se les denomina series temporales. Estas bases de datos se utilizan en muchas aplicaciones reales por lo que existe un gran interés histórico en su estudio dentro del área de la estadística. Este interés se ha acrecentado con el desarrollo de la minería de datos, dando lugar a lo que se conoce como minería de datos temporales.

La predicción de series temporales es un área de investigación activa en la que los paradigmas típicos utilizados han sido los modelos estadísticos [Frances y Dijk, 2000], tales como ARIMA (*Auto-Regressive Integrated Moving Average*), y en la que cada vez más se están aplicando nuevos métodos pertenecientes a la minería de datos. Entre las técnicas de

minería de datos aplicadas a predicción de series temporales cabe destacar principalmente las redes neuronales [Hobbs y otros, 1998; Ture y Kurt, 2006; Pino y otros, 2008; Co y Boosarawongse, 2007] y los sistemas basados en reglas difusas [Azadeh y otros, 2008; Jang, 1993; Khashei y otros, 2008; Liu y otros, 2008; Yu y Wilkinson, 2008].

En este capítulo el objetivo es aplicar CO^2RBFN para extraer conocimiento de problemas de predicción de series temporales. Para ello se analizan previamente las características y el problema de predicción de series temporales, así como un estudio bibliográfico de los métodos utilizados en este tipo de problemas. A continuación, se describen la modificaciones realizadas en CO^2RBFN para poder aplicarlo a este tipo de problema y se aplicará el método a series temporales descritas en la bibliografía y a un problema de predicción del precio del aceite de oliva virgen extra.

5.1. El problema de predicción de series temporales

El objetivo del análisis de series temporales es realizar inferencias sobre el proceso estocástico desconocido a partir de la serie temporal observada. El problema es que prácticamente todas las variables aleatorias, cuando se ordenan según un parámetro temporal, pueden ser consideradas un proceso estocástico y que, por otra parte, únicamente disponemos de una realización muestral del proceso (una única serie temporal). Por este motivo debemos imponer una serie de condiciones o requisitos que permitan realizar las inferencias de interés.

El tipo de análisis, así como los modelos en los que se base el estudio, dependerán en gran medida del tipo de cuestiones que se quieran responder. Cuando el estudio se corresponde con una única variable, el análisis de series temporales usualmente intenta construir un modelo que explique la estructura (descripción) y anticipe la evolución (predicción) de la variable de interés. De esta forma, en el modo descriptivo, se usan los datos para encontrar patrones de su comportamiento, reglas que describen las asociaciones entre las ocurrencias que permiten descubrir generalidades y anomalías en el comportamiento de los datos. En el modo predictivo, los datos se analizan para poder descubrir un modelo de su comportamiento futuro y así poder estimar posibles valores y tendencias. Es decir, existen dos grandes objetivos que han impulsado el estudio de las series temporales: identificar la naturaleza del sistema que genera la secuencia de los datos, y predecir los valores futuros que tomará la serie temporal.

El objetivo de cualquier método de predicción básico es deducir un resultado a partir de un conjunto de valores pasados. Hay diversas razones para el estudio de las series temporales:

- la necesidad de predecir el comportamiento de una variable en el futuro,
- la necesidad de controlar un proceso dado,
- mejorar los beneficios de una empresa, anticipando los incrementos o decrementos de precio en el mercado,
- la simulación de fenómenos que no se pueden implementar físicamente,
- la generación de nuevas teorías físicas o biologías, etc.

En general, el objetivo último es incrementar el conocimiento sobre un determinado fenómeno o aspecto de nuestro entorno con los datos que se tienen del pasado y del presente. Por lo tanto, el principal objetivo es extraer las regularidades que se han observado en el pasado sobre el comportamiento de la variable, es decir, obtener el mecanismo que la genera, para así entender mejor su comportamiento a lo largo del tiempo. Además, bajo el supuesto de que las condiciones estructurales que conforman la serie objeto de estudio permanecen constantes, también se trata de predecir el comportamiento futuro.

El análisis de una serie temporal se puede ver cómo la construcción de un modelo que se ajuste a la evolución de los datos de serie. El modelo puede tener enfoques diferentes según la orientación de su estudio.

5.2. Procesos estocásticos y series temporales

En esta sección se describen las características de las series temporales necesarias para poder realizar un análisis de las mismas.

Una serie temporal es un conjunto de observaciones regulares ordenadas en el tiempo, sobre una determinada variable, tomadas en periodos de tiempo sucesivos y en la mayoría de los casos equidistantes. Las series pueden tener una periodicidad anual, semestral, trimestral, mensual, etc., según los periodos de tiempo en los que están recogidos los datos que la componen.

Ejemplos de series temporales se pueden encontrar en muchos ámbitos de conocimiento: economía (tasa de desempleo, evolución de precios de un determinado producto, etc), demografía (crecimiento de la población,

envejecimiento de la misma, etc), meteorología (nivel de temperatura anual, tasa de lluvias, etc), medio ambiente (emisiones anuales de CO_2, tasa de deforestación, etc).

Una serie temporal se suele representar mediante un gráfico temporal en el que el tiempo se mide en el eje de abscisas y el valor de la variable observada en el eje de ordenadas.

Una serie temporal se dice que es *estacionaria* (o estable) si los valores de ésta oscilan alrededor de un nivel constante, es decir la media y la variabilidad se mantienen constantes a lo largo del tiempo. Por el contrario, una serie que no mantiene una media constante se dice que es *no estacionaria*. Cuando las series, independientemente de tener un nivel fijo o variable en el tiempo, tienen además un comportamiento superpuesto que se repite a lo largo del tiempo, se dice que la serie es *estacional*. La estacionalidad hace que la media de las observaciones no sea constante, pero evoluciona de forma previsible de acuerdo con un patrón cíclico.

En la práctica la clasificación de una serie como estacionaria o no estacionaria depende del periodo durante el cual se lleve a cabo la observación, ya que por ejemplo una serie puede ser estacionaria en un periodo corto y no estacionaria si se considera un periodo más largo.

Una serie temporal se considera una realización muestral o trayectoria (conjunto de observaciones ordenadas en el tiempo) de lo que se conoce como *proceso estocástico*. Un proceso estocástico se define como un conjunto de variables aleatorias, $\{x_t\} \quad t = 1, 2, \ldots$, ordenadas según un parámetro temporal t. Una serie temporal sería una muestra de tal proceso, es decir, $x_1, x_2, \ldots, x_t$, es una serie de t datos. Por ejemplo, podemos considerar

como proceso estocástico la temperatura media diaria en una ciudad y las observaciones se realizan todos los días del año. Una serie temporal serían las observaciones correspondientes a un año.

Un proceso estocástico queda caracterizado al definir la distribución de probabilidades conjunta de las variables aleatorias $x_1, x_2, \ldots, x_t$, para cualquier valor de t. Estas distribuciones (conocidas como *finito-dimensionales*) se pueden calcular cuando se puede disponer de varias series (realizaciones) del proceso estocástico, sin embargo en ciertas situaciones sólo podemos observar una realización del proceso. Para poder estimar las características *transversales* del proceso (medias, varianzas, etc.) a partir de su evolución *longitudinal*, es necesario suponer que dichas características son estables a lo largo del tiempo. Es decir, para poder efectuar inferencias a partir de una sola realización se han de imponer al proceso estocástico las restricciones de *estacionaridad* y *ergodicidad*.

Un proceso estocástico (o serie temporal) es estacionario en *sentido estricto* o *fuertemente estacionario*, si se cumple que la distribución conjunta de cualquier conjunto de variables no se modifica al realizar un desplazamiento de las variables en el tiempo, es decir, $F(x_j, x_k, \ldots, x_l) = F(x_{j+i}, x_{k+i}, \ldots, x_{l+i})$.

La estacionaridad en sentido estricto es una restricción muy fuerte, ya que para probarla se debe disponer de todas las distribuciones conjuntas para cualquier subconjunto de variables del proceso. Para relajar esta condición, de forma que se pueda contrastar en la práctica, se define un proceso estacionario en *sentido amplio* o *débilmente estacionario*, como aquel que cumple las condiciones siguientes:

1. la media es constante en el tiempo, $E[x_t] = \mu \quad \forall t$

2. la varianza también es constante en el tiempo, $Var(x_t) = \sigma^2 \quad \forall t$

3. la covarianza entre dos observaciones sólo depende de los retardos entre ellas (k) y no del tiempo, $E[(x_t - \mu)(x_{t-k} - \mu)] = \gamma_k \quad \forall t$

En un proceso estacionario las autocovarianzas y autocorrelaciones sólo dependen del retardo entre las observaciones y la relación entre x_t y x_{t-k} es siempre igual a la relación entre x y x_{t+k}. De esta forma, en los procesos estacionarios $Cov(x_t, x_{t+k}) = Cov(x_{t+j}, x_{t+j+k}) = \gamma_k \quad j = 0, \pm 1, \pm 2, \ldots$ y las autocorrelaciones quedan definidas como $\rho_k = \frac{\gamma_k}{\gamma_0}, \quad k \geq 0$. Al ser el proceso estacionario se cumple, en resumen, $\gamma_0 = \sigma^2$, $\gamma_k = \gamma_{-k}$ y por tanto $\rho_k = \rho_{-k}$.

Se dice que un proceso es ergódico cuando los valores de la serie temporal alejados en el tiempo están poco correlados. Una condición necesaria, aunque no suficiente, de la ergodicidad es que $\lim_{k\to\infty} \rho_k = 0$.

Se puede estimar la media (μ), varianza (γ_0) y covarianzas ($\gamma_1, \ldots$) (denominadas autocovarianzas, por cuanto se refieren a covarianzas de la misma variable), de un proceso haciendo uso de las las ecuaciones 5.1. Para que estas estimaciones se puedan hacer a partir de una única realización se necesita que el proceso sea estacionario y para que dichas inferencias sean consistentes se necesita que el proceso sea ergódico.

$$\begin{aligned}\hat{\mu} =& \frac{1}{T}\sum_{t=1}^{T} x_t \\ \hat{\gamma_0} =& \frac{1}{T}\sum_{t=1}^{T} (x_t - \hat{\mu})^2 \\ \hat{\gamma_k} =& \frac{1}{T}\sum_{t=1}^{T-k} (x_{t+k} - \hat{\mu})(x_t - \hat{\mu})\end{aligned} \tag{5.1}$$

La mayoría de las series no son estacionarias, pero se puede eliminar la tendencia y estabilizar la varianza para transformarlas en estacionarias, de forma que se pueda aplicar el proceso de inferencia visto para los procesos estacionarios.

La representación gráfica de los coeficientes de autocorrelación del proceso en función del retardo se denomina *función de autocorrelación simple* (FAS). Se define el coeficiente de autocorrelación parcial de orden k, ρ_k^p, como el coeficiente de correlación entre observaciones separadas k periodos, cuando se ha eliminado de la relación entre las dos variables la dependencia debida a los valores intermedios. Se llama *función de autocorrelación parcial* (FAP) a la representación de los coeficientes de correlación parcial en función del retardo. Estas dos funciones constituyen un instrumento de análisis de series temporales de gran interés práctico.

Para entender mejor el comportamiento de una serie temporal, ésta se supone compuesta de cuatro elementos o movimientos principales, tendencia (T_t), factor cíclico (C_t), movimiento estacional (E_t) y movimiento irregular (I_t) [Han y Kamber, 2000]:

- Tendencia o movimientos a largo plazo: indican el comportamiento general de la serie en un período de tiempo largo. Ayudan a identificar

cuál es la tendencia que sigue o ha seguido la serie.

- Variaciones cíclicas: representan ciclos que tienen las series. Dichas variaciones pueden o no ser periódicas, es decir, los ciclos pueden no ser completamente iguales después de períodos de tiempo idénticos.

- Movimientos estacionales: estos movimientos se deben a eventos que ocurren con una frecuencia establecida y constante, aunque la amplitud puede ser variable.

- Movimientos aleatorios o irregulares: representan el comportamiento de la serie debido a eventos aleatorios o semi-Alearorios. Es el componente que no está sujeto a ninguna periodicidad en el tiempo. Dentro de los movimientos irregulares la parte que no es predecible se considera aleatoria.

Dentro del contexto del análisis de la serie por componentes, una serie temporal X_t se puede ver según el *esquema aditivo* como la suma de los cuatro componentes (ecuación 5.2) o como el producto de los cuatro según el *esquema multiplicativo* (ecuación 5.3).

$$Y_t = T_t + C_t + E_t + I_t \tag{5.2}$$

$$Y_t = T_t \cdot C_t \cdot E_t \cdot I_t \tag{5.3}$$

5.3. Modelos para el análisis de series temporales

Hay distintos modelos para realizar el análisis de series temporales atendiendo a si dichos modelos se centran sólo en una serie o en las posibles relaciones de varias series entre sí:

- modelos univariantes: representan la evolución de una serie temporal y pueden generar predicciones de su comportamiento futuro, basándose únicamente en la evolución histórica de la propia serie. Las predicciones obtenidas por dichos modelos se basan en la hipótesis de que las condiciones futuras serán análogas a las pasadas.

- modelos de regresión dinámica o de función de transferencia: estos modelos pretenden encontrar relaciones de dependencia entre la serie estudiada y ciertas variables. Las previsiones univariantes se pueden mejorar incorporando información de la evolución de otras variables y construyendo modelos que tengan en cuenta dichas dependencias.

- modelos multivariantes: son modelos que representan conjuntamente las relaciones dinámicas entre un grupo de series y obtienen predicciones simultáneas de sus valores futuros.

El interés se va a centrar en los métodos utilizados para analizar una serie temporal sin tener en cuenta otras variables que puedan influir en la misma, es decir, en los métodos univariantes. En particular interesa la aproximación estocástica (frente a la determinista) es decir aquélla que supone que la serie temporal tiene un carácter probabilístico, por cuanto ha sido generada por alguna variable aleatoria con una distribución de probabilidad determinada

aunque habitualmente desconocida.

En el análisis univariante se pueden considerar tres grandes grupos: métodos de descomposición, métodos de alisado exponencial y modelos ARIMA univariantes.

5.3.1. Modelos lineales de series temporales

En los procesos que tienen distribuciones conjuntamente normal (conocidos como procesos *lineales* o *gaussinanos*), la estacionaridad débil coincide con la estricta. Los procesos lineales son una clases especial de procesos estacionarios y ergódicos, caracterizados porque se pueden representar como una combinación lineal de variables aleatorias. Dentro de este tipo se incluyen:

- Procesos puramente aleatorios. Se caracterizan porque su media es 0, su varianza constante a lo largo del tiempo y no existe relación entre los valores observados en diferentes momentos de tiempo. A estos procesos puramente aleatorios se les suele denominar procesos de *ruido blanco*.
- Autorregresivos (AR). Son aquellos en los que una observación se obtiene mediante regresión de valores anteriores. El proceso autoregresivo de orden p, AR(p), viene expresado por (ecuación 5.4)

$$x_t = \phi_1 x_{t-1} + \phi_2 x_{t-2} + \ldots + \phi_p x_{t-p} + \epsilon_t \tag{5.4}$$

  donde ϵ_t es una variable aleatoria de ruido blanco.
- De medias móviles (MA). Se obtiene como promedio de variables de ruido blanco. El nombre de móviles viene dado al variar dichas

variables a lo largo del tiempo. Un proceso MA(q) se expresa como indica la ecuación 5.5

$$x_t = \epsilon_t - \theta_1\epsilon_{t-1} - \theta_2\epsilon_{t-2} - \ldots - \theta_q\epsilon_{t-q} \tag{5.5}$$

donde θ_i son coeficientes de ponderación. Tanto estos procesos como los autorregresivos fueron introducidos por Yule [Yule, 1921, 1926, 1927].

- Modelo mixto (ARMA). Es un proceso obtenido como combinación de los dos anteriores. En un proceso ARMA(p, q), p indica el retardo máximo de la parte autorregresiva y q indica el orden correspondiente a las medias móviles. La expresión del proceso es la indicada en la ecuación 5.6

$$x_t = \phi_1 x_{t-1} + \ldots + \phi_p x_{t-p} + \epsilon_t - \theta_1\epsilon_{t-1} - \ldots - \theta_q\epsilon_{t-q} \tag{5.6}$$

Estos procesos fueron estudiados por Wold [Wold, 1954] y popularizados por Box y Jenkins [Box y Jenkins, 1976].

El estudio de estos tipos es importante pues Wold demostró que cualquier proceso estacionario X_t se puede representar unívocamente como la suma de dos procesos mutuamente incorrelados $X_t = D_t + Y_t$, donde D_t es linealmente determinista e Y_t es un proceso MA(∞).

5.3.2. Identificación de modelos estacionarios

Como instrumentos básicos para la identificación de los modelos estacionarios, se utiliza la función de autocorrelación simple (FAS) y la función de autocorrelación parcial (FAP).

Existe una dualidad entre procesos AR y MA, de manera que la FAP de un MA(q) tiene la estructura de la FAS de un AR(q), y la FAS de un MA(q) tiene la estructura de la FAP de un AR(q). Un proceso AR(p) puede escribirse como un proceso MA(∞), es decir, como suma infinita de innovaciones. Un proceso MA(q) puede expresarse como un AR(∞), es decir, como suma infinita de valores anteriores de la serie. Todo proceso estacionario en sentido débil puede expresarse como suma de infinitas variables aleatorias incorreladas (ruido), o como suma de infinitas variables anteriores.

En un proceso AR la FAS empieza a decrecer a partir de un determinado retardo pero nunca se hace 0, esto ocurre con la FAP en un MA. En un AR(p) los coeficientes serán distintos de cero para retardos menores que p y cero para retardos mayores (la FAP presenta p *palos* que indican el orden del proceso autorregresivo). En un proceso AM(q) los coeficientes de autocorrelación parcial no se igualarán nunca a cero, aunque a partir de un retardo q decaerán de forma rápida. Así pues, la FAP de un proceso MA se comporta de manera análoga a como lo hace la FAS en un AR y recíprocamente.

En un proceso mixto ARMA(p, q) tanto la FAS como la FAP tienen infinitos elementos distintos de 0, ya que dan lugar, bien a un MA(∞) o bien a un AR(∞), por lo que se presentan mayores dificultades para su identificación.

En la tabla 5.1 se muestra un resumen del comportamiento de la FAS y FAP en los modelos AR(p), MA(q) y ARMA(p, q).

Tabla 5.1: *Comportamiento de la FAS y FAP según modelos*

	FAS	FAP
AR(p)	Muchos coeficientes no nulos con decrecimiento rápido	Se anula para retardos superiores a p
MA(q)	Se anula para retardos superiores a q	Muchos coeficientes no nulos con decrecimiento rápido
ARMA(p,q)	Muchos coeficientes no nulos	Muchos coeficientes no nulos

5.3.3. Procesos no estacionarios

La mayoría de las series que se manejan son de carácter no estacionario. Para poder aplicar el análisis visto se necesita que los procesos sean estacionarios, por lo que se tendrán que realizar una serie de transformaciones en ellos, de forma que tras éstas se conviertan en estacionarios.

Hay modelos que se pueden transformar en estacionarios mediante su *diferenciación*.

Diferenciar una serie x_t con T términos, consisten en obtener otra serie con $T-1$ términos obtenidos de la forma $y_t = x_t - x_{t-1}$, para $t = 2, \ldots T$. La diferenciación se puede realizar más de una vez y al número de veces que se realiza se le conoce como *orden de diferenciación*.

La diferenciación de un determinado orden, n, es suficiente en muchos casos para conseguir series estacionarias en media y varianza. En esta situación, se dice que dichas series son *procesos integrados de orden n*.

No obstante hay series que afectadas por una fuerte tendencia que además necesitarán de otra transformación del tipo Box-Cox para alcanzar la estacionaridad en varianza.

En otras ocasiones la no estacionaridad del proceso aparece debida a la estacionalidad del mismo. Dicha estacionalidad puede ser determinista (se modela como un proceso constante), puede ir cambiando pero su evolución es estacionaria, o puede ir cambiando de forma no determinística (va cambiando en el tiempo sin ningún valor medio fijo). En procesos tanto con estacionalidad determinista como estocástica, la diferenciación estacional convierte un proceso estacional en estacionario.

En resumen podemos decir que un proceso no estacionario se puede convertir en estacionario aplicando diferencias regulares, entre periodos consecutivos, y además si el proceso presenta estacionalidad, ésta se puede eliminar mediante diferencias estacionales.

Con objeto de aplicar la metodología de los modelos ARIMA es preciso transformar la serie a estudiar para que sea estacionaria.

5.4. ARIMA

Los modelos autorregresivos integrados de medias móviles, ARIMA (*Autorregresive Integrated Moving Average*) representan la familia de modelos más importantes para el análisis univariante de series temporales. Dichos métodos, también conocidos como modelos de Box-Jenkins [Box y Jenkins, 1976], predicen valores presentes de una variable a partir de valores pasados de la misma. Estos autores propusieron un mecanismo cíclico de tratamiento de este tipo de modelos con objeto de conocer, de modo relativamente sencillo, cuál de entre los posibles elementos de aquella familia de modelos ejerce una mejor representación de la serie estudiada.

Los procesos ARIMA son una clase particular de procesos estocásticos con los que se puede describir el comportamiento de la mayoría de las series temporales. Se considera que una serie temporal dada es una realización de un proceso estocástico específico ARIMA(p, d, q).

El desarrollo de la metodología ARIMA consiste en encontrar un modelo ARIMA(p, d, q), que sea capaz de generar la serie temporal objeto de estudio. En dicho modelo, p es el valor para el parámetro auto-regresivo, d es el orden de diferenciación (el número de veces que se han de tomar diferencias para que el proceso sea estacionario) y q es el parámetro de la media móvil.

La modelización ARIMA implica las siguientes etapas:

- Identificación del modelo o parámetros iniciales p, d, y q.
- Estimación de los parámetros p, y q.
- Validación del modelo.
- Utilización del modelo para la predicción.

El propósito de la fase de identificación es seleccionar el mejor modelo, entre los posibles candidatos, que haya podido generar la serie temporal objeto de estudio. Para ello es necesario alcanzar la estacionaridad en la serie, de forma que ni la media, ni la varianza ni la autocorrelación dependan del tiempo. Por tanto en la primera etapa de identificación se aplican las transformaciones necesarias a la serie para transformarla en estacionaria (siendo d el orden diferenciación necesario para estabilizar la serie en media y a veces son también necesarias transformaciones para estabilizarla en varianza), posteriormente se calculan p, y q para un modelo inicial. En una

segunda etapa, se procede a determinar el orden de la parte autorregresiva (p) y el orden las medias móviles (q) del proceso que haya podido generar la serie ya estacionaria y así se van ajustando los parámetros del modelo. Una de propuestas más populares para realizar dicha estimación es MLE (*Maximum Likelihood Estimation*) [Edwards, 1972].

En la fase de validación se determina si existe o no una adecuación entre la serie y el modelo obtenido en las fases anteriores. Para validar el modelo se realiza un análisis de los residuos (diferencia entre el valor dado por el modelo y el observado) para ver si el modelo se ajusta a los datos. Los residuos deben ser aleatorios y estar distribuidos según una normal, si no ocurre esto, se tendrá que volver de nuevo a la etapa de identificación y repetir el proceso de obtención de otro modelo. Cuando el modelo ya es el adecuado se podrá utilizar para la predicción de valores futuros de la serie a partir de los observados.

Más información sobre métodos para el análisis de series temporales se puede consultar en [Peña, 2005; Uriel y Peiró, 2000].

5.5. Adaptaciones del algoritmo CO^2RBFN

El algoritmo CO^2RBFN se va a adaptar en esta sección para poder aplicarse a la tarea de predicción de series temporales.

En clasificación, las RBFNs diseñadas tenían tantas salidas como posibles clases en el conjunto de datos y se activaba con un mayor valor la salida que representaba la clase que la RBFN decidía para el ejemplo presentado. Ahora, en predicción, los valores de salida pertenecen a un conjunto de

valores continuos y las RBFNs a diseñar van a tener una única salida por lo que la arquitectura empleada será del tipo descrito en la figura 5.1.

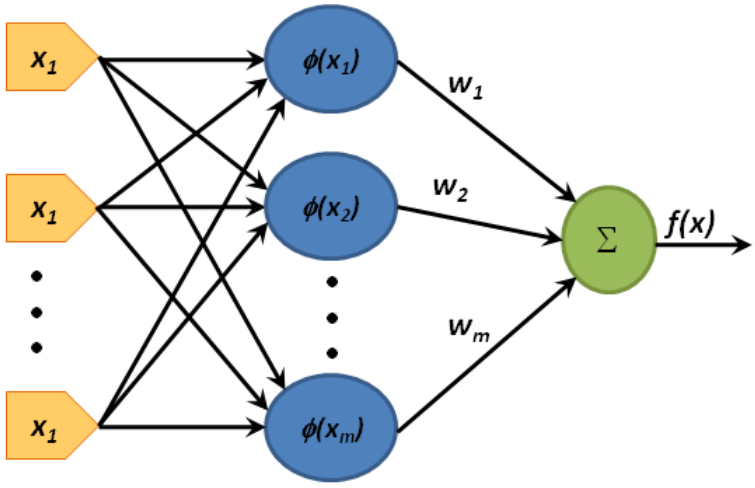


Figura 5.1: *Arquitectura típica de una RBFN*

Como ya se vio anteriormente, el algoritmo CO^2RBFN mide la asignación de crédito de una RBF mediante tres factores: aportación, error y solapamiento. Para adaptar el modelo a la predicción de series temporales se cambia la medida de error, de forma que ahora se calcula para cada RBF, ϕ_i, su error, e_i, como el Error Porcentual Absoluto Medio ($MAPE$) cometido representado en la ecuación 5.7:

$$e_i = \frac{\sum_{\forall pi_i} \left| \frac{f_i - y_i}{f_i} \right|}{pi_i} \tag{5.7}$$

donde f_i es la salida predicha por el modelo, y_i es la salida deseada y pi_i el número de patrones dentro del radio de la RBF ϕ_i.

Otro cambio aparece en el operador de mutación informada, ya que

en clasificación se intentaba situar la RBF en el centro de los patrones de su clase y ahora esto no tiene sentido. El operador de mutación en predicción puede modificar el radio y las coordenadas del centro de la RBF, usando información de su entorno. Las modificaciones del centro y radio siguen las recomendaciones dadas en [Ghost y otros, 1992] similares a las utilizadas por el algoritmo LMS en el cálculo de los pesos. El proceso consiste en calcular el error cometido en los ejemplos que están dentro del radio de la RBF, ϕ_i. Para cada coordenada del centro y para el radio se calcula una variación Δc_{ij} y Δr_i respectivamente (ecuaciones 5.8 y 5.9). Las nuevas coordenadas del centro y el nuevo radio se obtienen cambiando (incrementando o decrementando) sus valores antiguos en una cantidad aleatoria (entre un 5 % y un 50 % del valor del radio actual de la RBF). El decremento o incremento se realiza en función del signo de variación que se calcula.

$$\Delta r_i = \sum_k e(\vec{p}_k) \cdot w_i \tag{5.8}$$

donde $e(\vec{p}_k)$ es el error cometido con el ejemplo $\vec{p}_k$.

$$\Delta c_{ij} = signo(c_{ij} - p_{kj}) \cdot e(\vec{p}_k) \cdot w_i \tag{5.9}$$

La última diferencia es la inserción informada de nuevas RBFs. Como se vio en el capítulo anterior, al insertar una nueva RBF se podría hacer de forma aleatoria sobre un patrón fuera del radio de cualquier RBF o bien insertarla, utilizando cierta información, en un patrón que estuviese mal clasificado. Ahora la inserción informada se va a hacer en el centro de una

zona del espacio donde se esté cometiendo el máximo error en la predicción.

5.6. Resultados en predicción de series temporales

En esta sección y como primera aproximación se van a comparar los resultados obtenidos por CO^2RBFN y los obtenidos mediante otros modelos de minería de datos, tales como Fuzzy-GAP, MLP-Grad, NU-SVR, RBFN-LMS. Los algoritmos se han aplicado sobre tres series disponibles en http://www.alianzaeditorial.es/3491099/ejercicios.zip.

Las figuras 5.2, 5.3 y 5.4 muestran gráficamente las series utilizadas:

- Accidentes en jornada de trabajo en España: formada por los datos recogidos mensualmente desde enero de 1979 hasta diciembre de 1998. Fuente: INE.
- Índice general de la Bolsa de Madrid: recoge los datos mensuales desde enero de 1988 hasta Mayo de 2003. Fuente: Banco de España e INE.
- Tipo de interés interbancario a un año: recoge los datos mensuales desde enero de 1988 hasta marzo de 2002. Fuente: Ministerio de Economía.

Como conjunto de test se han tomado los últimos 25 datos de la serie y como conjunto de entrenamiento el resto de los datos. Para formar las muestras de los conjuntos de datos se ha elegido un diseño clásico de patrones (n-3, n-2, n-1, n, n+1), donde n+1 es el dato a predecir y el resto los datos de entrada al modelo. A continuación se describen los métodos de minería de datos con los que se compara CO^2RBFN:

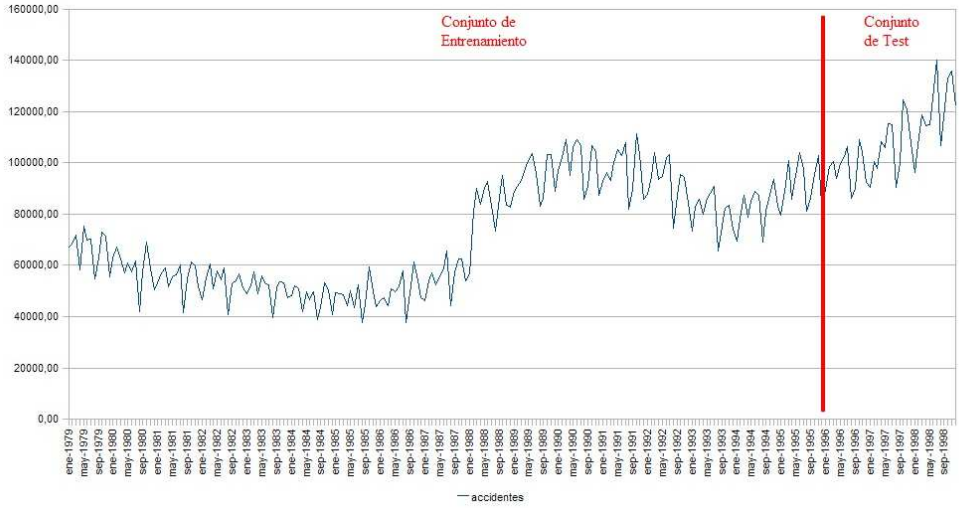


Figura 5.2: *Accidentes en jornada de trabajo en España*

Figura 5.3: *Índice general de la bolsa de Madrid*

- Fuzzy-GAP: Un método GA-P [Howard y D'Angelo, 1995] usa un algoritmo de computación evolutivo, un híbrido entre algoritmos

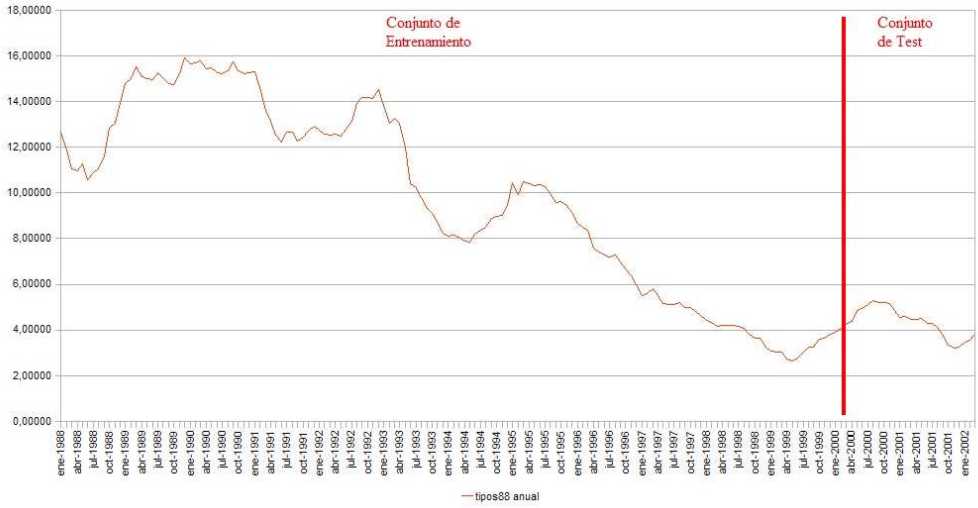


Figura 5.4: *Tipo de interés interbancario a un año*

genéticos y programación genética, y está optimizado para realizar regresiones simbólicas. Cada elemento está compuesto por una cadena de parámetros y un árbol que describe una función que depende de dichos parámetros. Utiliza operadores de cruce y mutación para generar los miembros de la nueva población. Ambos operadores se aplican independientemente sobre el árbol y la cadena de parámetros.

En el algoritmo Fuzzy-GAP propuesto en [Sánchez y Couso, 2000], los conjuntos difusos del modelo se codifican en los nodos terminales del árbol y los operadores aritméticos difusos se usan para evaluar el árbol. El objetivo es encontrar una función de salida g tal que las diferencias entre la salida real y la del modelo, $y-g(\vec{x})$, sean las menores para cada valor de $\vec{x}$. De este modo, se define $\alpha - corte$ como un intervalo con β grado de confianza, donde las funciones g^+ y g^- se definen de forma que $g^+(\vec{x})$ es el valor máximo del intervalo y $g^-(\vec{x})$ el valor mínimo. La salida del modelo difuso se define mediante la unión de las salidas

que produce cada una de sus $\alpha - corte$ aplicando la misma entrada a todas ellas.

- MLP-Grad: algoritmo para el diseño de Redes Perceptrón Multicapa que utiliza el algoritmo de Gradiente Conjugado para el aprendizaje. Descrito anteriormente en la sección 4.4 del capítulo 4.

- NU-SVR: algoritmo para el diseño de una máquina de vectores de soporte (SVM) [Fan y otros, 2005]. Los kernels utilizados son del tipo RBF.

  La tarea principal de una SVM es resolver un problema de optimización de segundo grado, encuentra un hiperplano de separación lineal con el margen máximo en este espacio de dimensiones superiores.

  Los datos se transforman por medio de una función del núcleo, que aumenta la dimensionalidad de los datos. Este aumento provoca que los datos pueden ser separados por un hiperplano con una probabilidad mucho mayor, y establecer un mínimo en la medida de la probabilidad de error de predicción.

- RBFN-LMS: Construye una RBFN con un número prefijado de RBFs. Por medio del algoritmo de *clustering* de las K-medias, elige un número de puntos del conjunto de entrenamiento para que sean el centro de las neuronas. Finalmente, establece un radio de la misma media para todas las neuronas como la mitad de la distancia media entre el conjunto de centros. Una vez que se han fijado los centros y los radios para las neuronas, se calcula el conjunto de pesos mediante el algoritmo LMS [Widrow y Lehr, 1990].

Se han utilizado estos métodos de forma que queden representados

los modelos más utilizados en minería de datos, tales como las redes neuronales, sistemas difusos y máquinas de vectores de soporte. Dentro de éstos paradigmas se han escogido los algoritmos que suelen dar mejores resultados en predicción de series temporales.

En la tabla 5.2 se muestran los valores de los parámetros utilizados por CO^2RBFN. La implementación del resto de algoritmos se ha obtenido de KEEL [Alcalá-Fdez y otros, 2009] y los valores de sus parámetros se han fijado según las indicaciones dadas por los autores de dichos algoritmos (apéndice C). Todos los métodos de minería de datos no determinísticos se han ejecutado 10 veces.

Tabla 5.2: *Parámetros de CO^2RBFN en la predicción de series temporales*

Parámetro	*Valor*
Generaciones del ciclo principal	200
Número de RBF's	20

La medida de error considerada, para evaluar la precisión de las predicciones por parte de los métodos, es el error MAPE. Las series temporales se han diferenciado para evitar problemas derivados de la no estacionalidad de las mismas. Las predicciones se han realizado sobre los datos diferenciados, pero los errores se han calculado después de recomponer la serie original. En las tablas (5.3, 5.4 y 5.5) se muestra la media del error MAPE cometido y su desviación típica sobre el conjunto de test. Las figuras 5.5, 5.3 y 5.4 muestran la mejor predicción conseguida por los métodos para el conjunto de test.

El análisis de los resultados muestra:

Método	*Error MAPE*
CO^2RBFN	0.08310 ± 0.00351
Fuzzy-GAP	0.08276 ± 0.00684
MLP-Grad	0.08311 ± 0.00495
NU-SVR	0.09632 ± 0.00000
RBFN-LMS	**0.08019 ± 0.00496**

Tabla 5.3: *Resultados con la serie de Accidentes*

Método	Error MAPE
CO^2RBFN	**0.05257 ± 0.00119**
Fuzzy-GAP	0.06094 ± 0.00655
MLP-Grad	0.05563 ± 0.00224
NU-SVR	0.05260 ± 0.00000
RBFN-LMS	0.05962 ± 0.00238

Tabla 5.4: *Resultados con la serie de la Bolsa*

Método	Error MAPE
CO^2RBFN	0.03743 ± 0.00199
Fuzzy-GAP	0.04016 ± 0.00280
MLP-Grad	**0.03522 ± 0.00125**
NU-SVR	0.04049 ± 0.00000
RBFN-LMS	0.03863 ± 0.00326

Tabla 5.5: *Resultados con la serie de Interés Interbancario*

- Un comportamiento correcto y similar de los métodos utilizados para la predicción de las series temporales. La serie más complicada de predecir es la que contienen los datos de los accidentes en jornada de trabajo en España. Tal y como se puede observar esta serie presenta oscilaciones muy bruscas y un comportamiento en el conjunto de test que no se asemeja al del conjunto de entrenamiento.

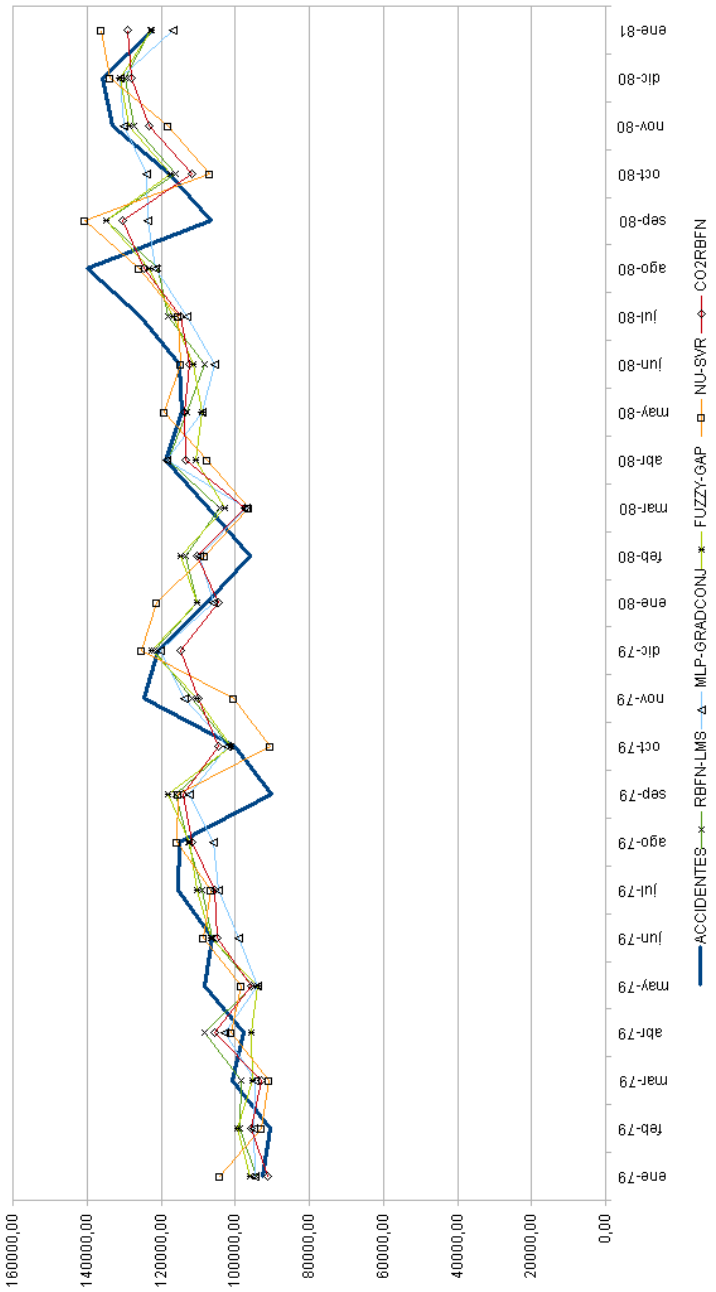


Figura 5.5: *Resultados de la predicción para la serie de accidentes*

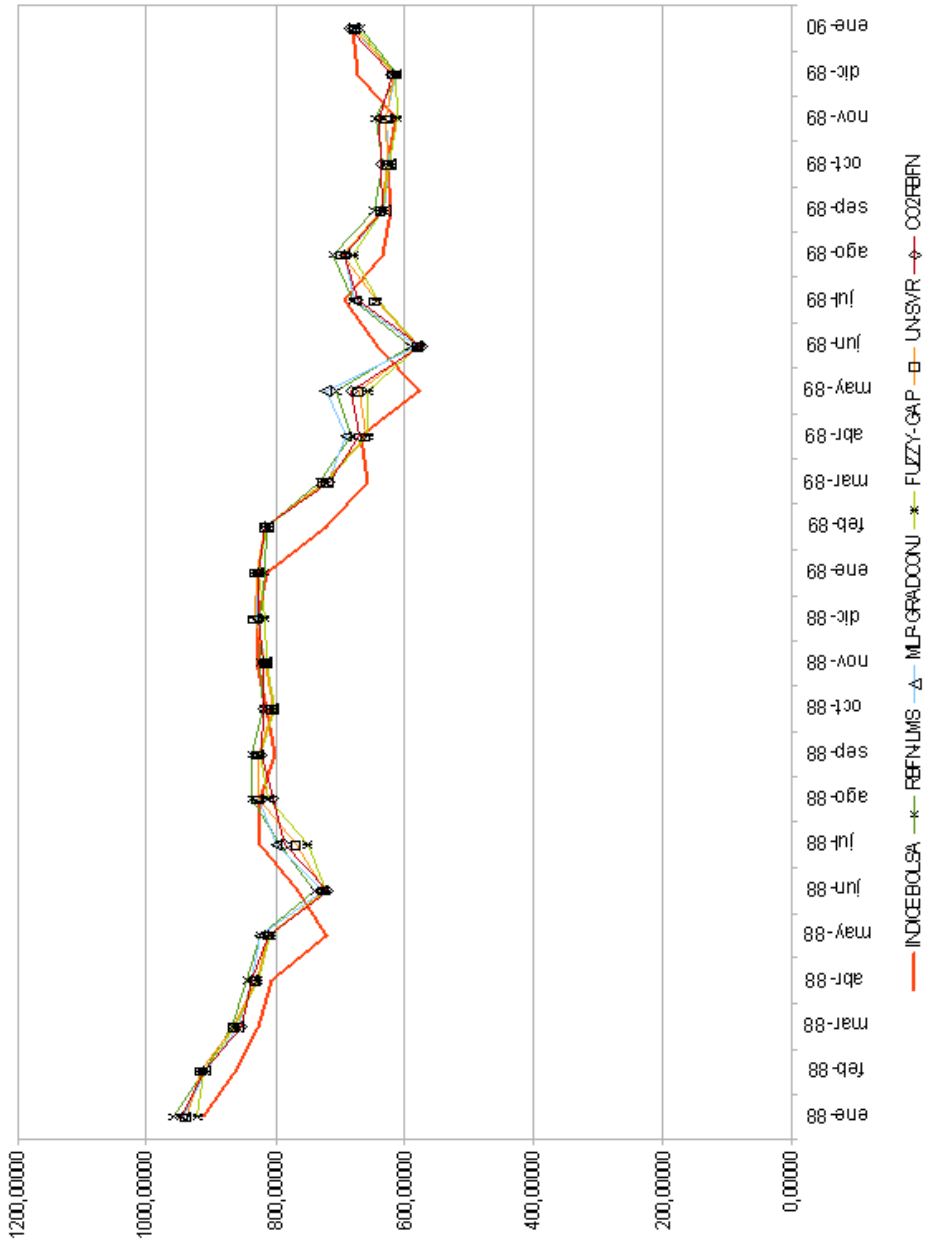


Figura 5.6: *Resultados de la predicción para la serie de la bolsa*

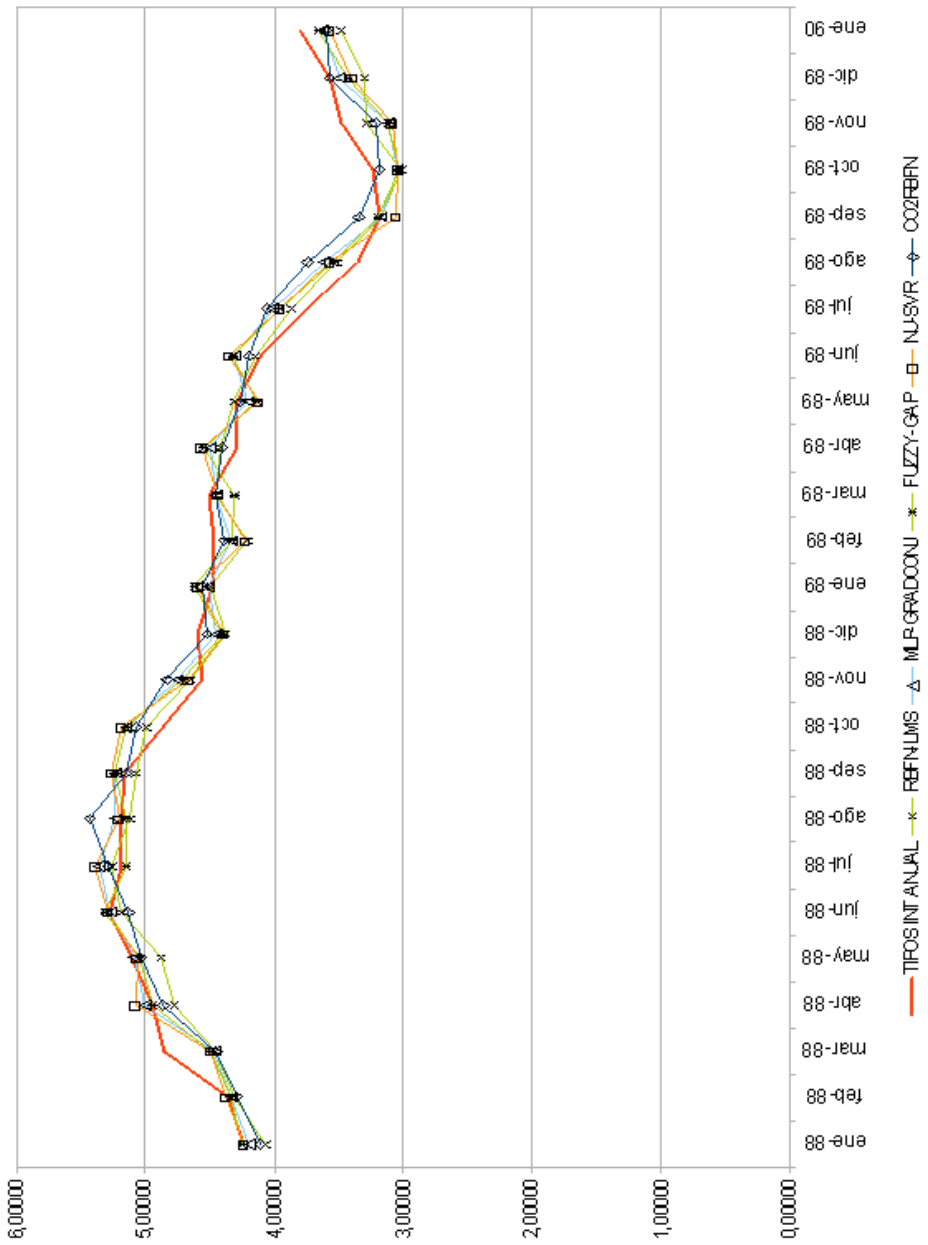


Figura 5.7: *Resultados de la predicción para el interés interbancario a un año*

- Los métodos han modelizado mejor la serie que contiene los datos del tipo de interés interbancario a un año. Sus variaciones son más suaves y el comportamiento de los datos en el conjunto de test es similar al de los datos en el conjunto de entrenamiento.

- CO^2RBFN obtiene redes que predicen las series con resultados comparables a los obtenidos con los mejores métodos. La desviación típica del error obtenido suele ser más baja que la del resto de métodos salvo que la de NU-SVR, dado que este último es un método determinístico. Esto implica que CO^2RBFN muestra una robustez superior a la del resto de algoritmos.

5.7. Predicción del precio del aceite de oliva virgen extra

El Consejo internacional del aceite y la regulación en la UE definen el aceite de oliva virgen extra como un producto 100 % zumo de aceituna sin aditivos, colorantes, saborizantes, o cualquier otra sustancia añadida permitida. El aceite de oliva virgen extra se obtiene a partir de la presión en frío de las aceitunas y no contienen más de un 0.8 % de acidez.

El aceite de oliva constituye un sector empresarial en continua expansión. España es el primer país productor con una media de producción anual de entre 700,000 y 800,000 toneladas, y también el primer país exportador con una media anual de exportación, en los últimos 10 años, de más de 300,000 toneladas. La provincia de España más productiva es Jaén, con una cosecha media anual entre 400,000 y 500,000 toneladas.

Los agentes implicados en este sector muestran interés por el uso de métodos para la predicción del precio del aceite de oliva. Dicha predicción es especialmente importante en el *Mercado de Futuros* [1], una sociedad cuyo objetivo es negociar un precio adecuado para el aceite de oliva para cuando se produzca su venta en el futuro. En este contexto, una predicción adecuada del precio del aceite en el futuro puede incrementar los beneficios globales.

Dada la importancia que presenta el poder predecir el precio del aceite de oliva virgen extra, se va a aplicar el algoritmo propuesto para resolver dicha tarea.

Los datos con los que vamos a trabajar para la predicción son datos del precio semanal de oliva virgen extra, obtenidos de *Poolred* [2], una iniciativa de la Fundación para la Promoción y Desarrollo del Aceite de Oliva , localizado en Jaén. La serie temporal que forman los datos contiene el precio semanal por kilogramo de aceite de oliva virgen extra. El estudio se va a realizar en dos partes, por un lado se van a utilizar los datos que van desde la semana 32 del año 2000 a la semana 52 del año 2005 y por otro lado el conjunto de datos que va desde la semana 1 del año 2007 hasta la 53 del 2008.

5.7.1. Predicción en el periodo 2000-2005

La tarea que se va a realizar es predecir el precio del aceite para la siguiente semana. En este estudio los datos usados están comprendidos desde la semana 32 (agosto) del año 2000 a la semana 52 (diciembre) del año 2005. El conjunto de datos se divide en dos subconjuntos: uno para entrenamiento y otro para test. Para el entrenamiento se va a utilizar el conjunto de datos

[1] http://www.mfao.es
[2] http://www.oliva.net/poolred/

que va desde la semana 32 del año 2000 hasta la semana 32 del 2005. Como conjunto de test para estimar la predicción de los métodos que se van a aplicar, se usa el conjunto de datos que comprende desde la semana 33 a la 52 del año 2005. El tamaño de los conjuntos de datos se ha seleccionado teniendo en cuenta el uso de los modelos ARIMA. La figura 5.8 muestra la serie temporal que forma los datos (conjuntos de entrenamiento y test).

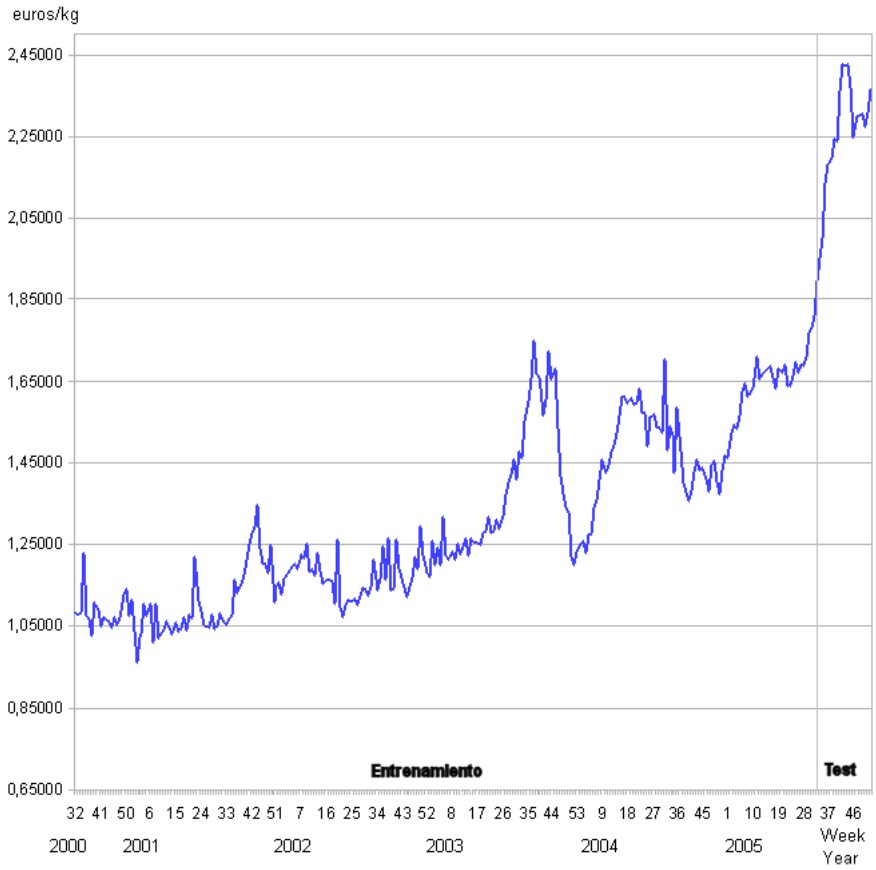


Figura 5.8: *Precio semanal del aceite de oliva virgen extra*

Un patrón de los conjuntos de entrenamiento y test tiene la forma $(n-4, n-3, n-2, n-1, n, n+1)$, donde $n+1$ es el precio a predecir y los valores

que van desde $n-4$ hasta n son los precios pasados a partir de los cuales se va a realizar la predicción. Esta decisión está justificada por el análisis de la serie realizada por los modelos ARIMA que concluyen que la predicción de un valor $n+1$, se puede obtener a partir de los cinco valores anteriores.

Un análisis preliminar del precio semanal de aceite de oliva extra, representado en la figura 5.8, muestra que se trata de una serie no estacionaria, ya que el precio muestra una tendencia creciente en el tiempo. Dicha no estacionaridad inherente se confirma también en el gráfico de la figura 5.9, donde la FAS decrece lentamente para la serie dada.

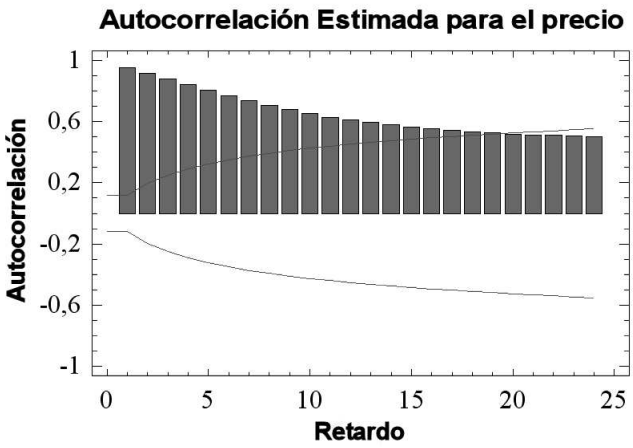


Figura 5.9: *FAS para la serie del precio del aceite de oliva*

La no estacionaridad de la serie se elimina cuando dicha serie se diferencia en el tiempo, tal y como se ve en la muestra la FAS en la figura 5.10. Un valor grande en los retardos indica la necesidad de un modelo MA en la serie. La FAS muestra que en el retardo 5 se cruza el 95 % del límite de confianza, y este comportamiento se puede deber al azar o puede indicar la necesidad de introducir otro parámetro en el modelo. La FAP de la serie diferenciada,

figura 5.11, muestra una atenuación rápida en los primeros retardos y esto se debe a la necesidad de un componente MA en el modelo.

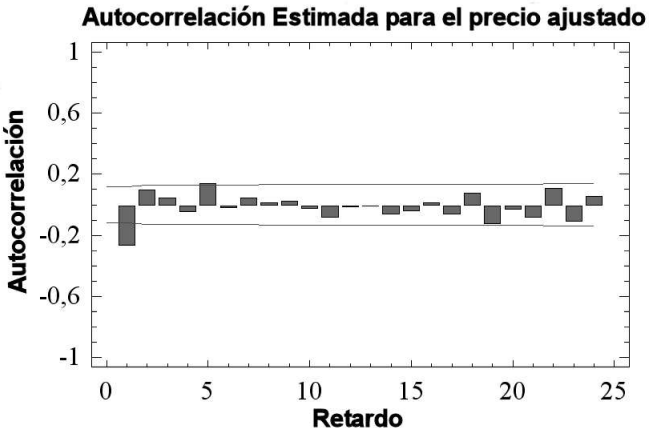


Figura 5.10: *FAS para la serie diferenciada del precio del aceite de oliva*

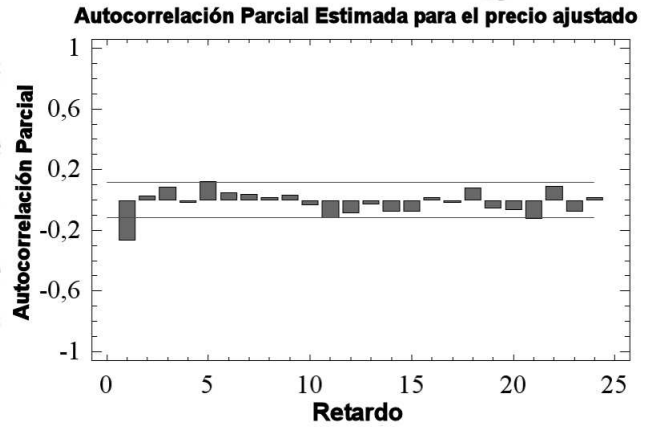


Figura 5.11: *FAP para la serie diferenciada del precio del aceite de oliva*

Basándose en esta información identificada, la serie del precio del aceite de oliva virgen extra se puede modelar mediante un modelo ARIMA(0,1,1)

o ARIMA(0,1,5). Si el precio semanal del aceite de oliva virgen se modela mediante un ARIMA(1,1,1), es decir, con componente AR, el valor para el término AR(1) es 0.964692, mayor o igual que 0.05, por lo que no es estadísticamente significativo. Por tanto, se va a eliminar el término constante del modelo.

Una vez que se han calculado los parámetros del modelo, usando el *Maximum Likelihood Estimation* (MLE) [Edwards, 1972], se aplica el Criterio de Información (AIC) de Akaike [Akaike, 1974], modelo más simple que ofrece una buena aproximación a los datos. Para el modelo ARIMA (0,1,5) AIC obtiene un valor de 47,508 y para el modelo ARIMA (0,1,1) el valor AIC es 1948,930. Se puede deducir desde estos valores AIC que el modelo ARIMA (0,1,5) es mejor que el modelo ARIMA (0,1,1). Los resultados del test de residuos demuestran que se satisfacen las suposiciones de independencia, normalidad y varianza constante.

Tal y como se demostró previamente, la serie de los datos tiene una tendencia positiva. Por tanto, es conveniente diferenciar los datos también para poder aplicar los diferentes métodos de minería de datos, de forma que se consiga una serie estacionaria en la que los valores del conjunto de datos de entrenamiento estén en el mismo rango que los valores en el conjunto de test.

Los métodos de minería de datos con los que se compara, así como sus parámetros están especificados en la sección anterior tabla 5.2. En este caso cambia el número de neuronas de CO^2RBFN que se sitúa en 4.

Los resultados en cuanto al error MAPE obtenido por los diferentes métodos, se muestran en la tabla 5.6.

Método	Error MAPE en Test
ARIMA (0,1,1)	0.03256 ± 0
ARIMA (0,1,5)	0.02659 ± 0
CO^2RBFN	$\mathbf{0.02257 \pm 0.00153}$
Fuzzy-GAP	0.02692 ± 0.00441
MLP-Grad	0.02803 ± 0.00183
NU-SVR	0.02540 ± 0
RBFN-LMS	0.02943 ± 0.00530

Tabla 5.6: *Error MAPE en test para la predicción del aceite de oliva en el periodo 2000-2005*

El comportamiento gráfico de los diferentes modelos sobre los datos de test se puede observar en la figura 5.12 (se muestra la mejor repetición).

Analizando los resultados se pueden obtener las siguientes conclusiones:

- Mirando la gráfica de la serie del precio del aceite, figura 5.8, se observa que la serie no es trivial de predecir, dado que el comportamiento de los datos al final de la misma es diferente de su comportamiento inicial.
- Todos los métodos alcanzan buenos resultados tanto en el conjunto de entrenamiento como en el de test, siendo CO^2RBFN el que mejores resultados en test obtiene.
- Observando la gráfica de las predicciones realizadas, figura 5.12, se puede observar que todos los métodos de la minería de datos consiguen seguir forma de la serie que se predice, mientras que ARIMA tiene un comportamiento con tendencia a seguir una forma media y por tanto no es capaz de predecir bien valores que se salen de dicha media. Tal y como se muestra en la gráfica no sigue bien la forma de la serie cuando presenta la concavidad intermedia.

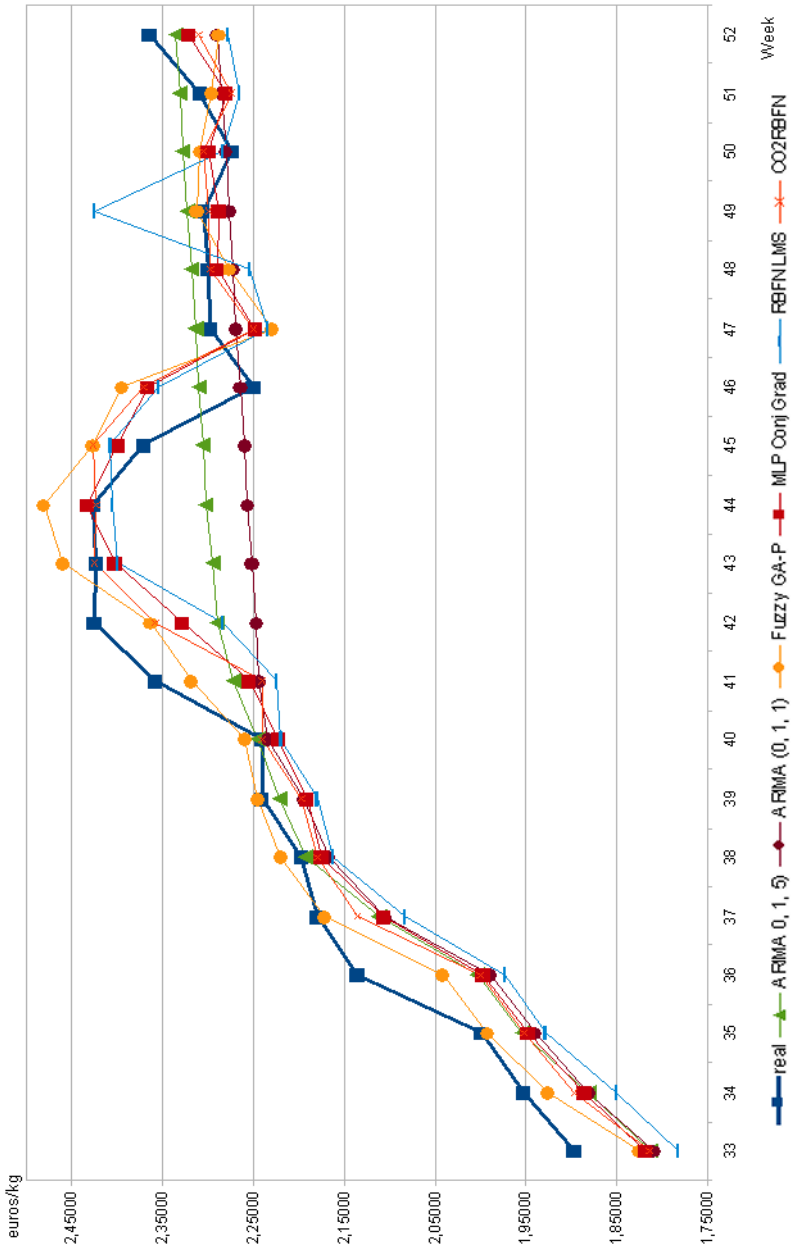


Figura 5.12: *Predicción de valores para el aceite de oliva virgen extra*

5.7.2. Predicción en el periodo 2007-2008

En este estudio se han usado los precios en euros por semana, del aceite de oliva virgen extra en España, que van desde la semana 1 del año 2007 hasta la 53 de 2008, la serie está representada en la figura 5.13.

Figura 5.13: *Serie temporal del precio del aceite de oliva virgen extra en toneladas/euros*

Como conjunto de entrenamiento se han utilizado los precios del aceite desde la semana 1 de 2007 hasta la semana 33 de 2008. Como conjunto de test se han utilizado los datos desde la semana 34 de 2008 hasta la semana 53 de 2008. Para el modelo ARIMA se ha estimado un modelo ARIMA(1,0,0). Los parámetros utilizados en el resto de los métodos de minería de datos con los que comparamos son los recomendados en la literatura. Para CO^2RBFN el número de RBFs o individuos de la población se ha fijado a 10. La medida de error considerada, para evaluar la precisión de las predicciones por parte de los métodos, es el error MAPE.

La serie temporal se ha diferenciado para evitar problemas derivados de la no estacionalidad de la misma. Las predicciones se han realizado sobre los datos diferenciados, pero los errores se han calculado después de recomponer la serie original. Se han hecho experimentos para la predicción con un horizonte de una semana y con un horizonte de cuatro semanas. Para la predicción a una semana y para formar las muestras de los conjuntos de datos se ha elegido un diseño clásico de patrones $(n-2, n-1, n, n+1)$, donde $n+1$ es el dato o diferencia con el valor actual a predecir y el resto los datos de entrada al modelo. Para la predicción a 4 semanas se han diseñado los patrones $(n-3, n-2, n-1, n, n+4)$ donde $n+4$ es la diferencia acumulada a 4 semanas.

El modo de trabajo tradicional de ARIMA consiste predecir el primer valor, del conjunto de test, a partir de los valores necesarios del conjunto de entrenamiento. Los siguientes valores del conjunto de test, dependiendo del modelo que haya generado los irá ya generando a partir de sus propias predicciones por lo que puede acumular mucho error si el número de datos del conjunto de test es superior a seis u ocho muestras. El valor de predicción obtenido por ARIMA tradicional es 0.13036.

Para que ARIMA trabaje en similares circunstancias a los métodos de minería de datos, se le "actualizan"los datos a partir del conjunto de test. Evidentemente y para la predicción a cuatro semanas siempre debe de utilizar en parte sus propias predicciones. Para obtener los resultados se han ejecutado los algoritmos 10 veces y en la tabla 5.7 se muestra la media del error MAPE cometido y su desviación típica. Las figuras muestran la mejor predicción conseguida por los métodos para el conjunto de test.

En las figuras 5.14 y 5.15 se muestran las predicciones obtenidas por los

Método	*Error MAPE en la predicción* *A una semana*	*A cuatro semanas*
ARIMA actualizado	0.02823 ± 0	0.06827 ± 0
CO^2RBFN	$\mathbf{0.01914 \pm 0.00057}$	0.03230 ± 0.00160
Fuzzy-GAP	0.02170 ± 0.00226	0.03536 ± 0.00461
MLP-Grad	0.02052 ± 0.00041	$\mathbf{0.02970 \pm 0.00196}$
NU-SVR	0.01936 ± 0	0.03003 ± 0
RBFN-LMS	0.02111 ± 0.00234	0.04706 ± 0.00901

Tabla 5.7: *Resultados en la predicción del precio del aceite de oliva en el periodo 2007-2008*

diferentes métodos a una semana y a cuatro semanas respectivamente, los resultados mostrados se corresponden con la mejor repetición alcanzada por los distintos métodos.

Si analizamos los resultados podemos sacar las siguientes conclusiones:

- Evidentemente los métodos cometen menos error en la predicción a una semana.
- Los métodos de minería de datos superan con claridad a los modelos ARIMA, los cuales se utilizaban tradicionalmente en econometría para la predicción de esta tipo de valores.
- Esta superioridad de los métodos de minería de datos sobre los métodos ARIMA es aún más clara si se utiliza ARIMA con su metodología tradicional y no se le actualizan los datos.
- El predominio de los métodos de minería de datos frente a los métodos ARIMA (que trabajan con métodos actualizados) se incrementa también al aumentar el horizonte de predicción.

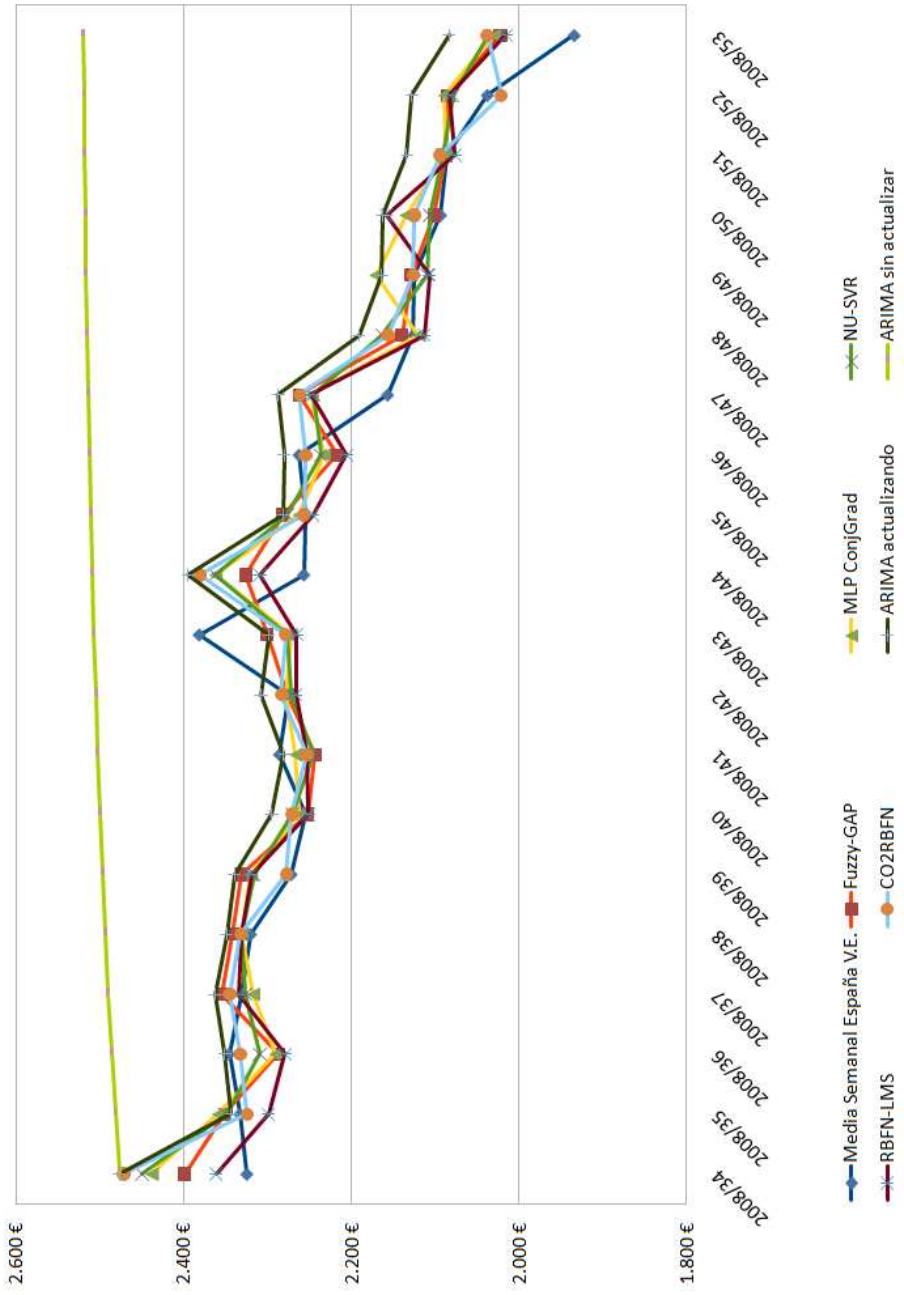


Figura 5.14: *Resultados de la predicción del precio del aceite a una semana*

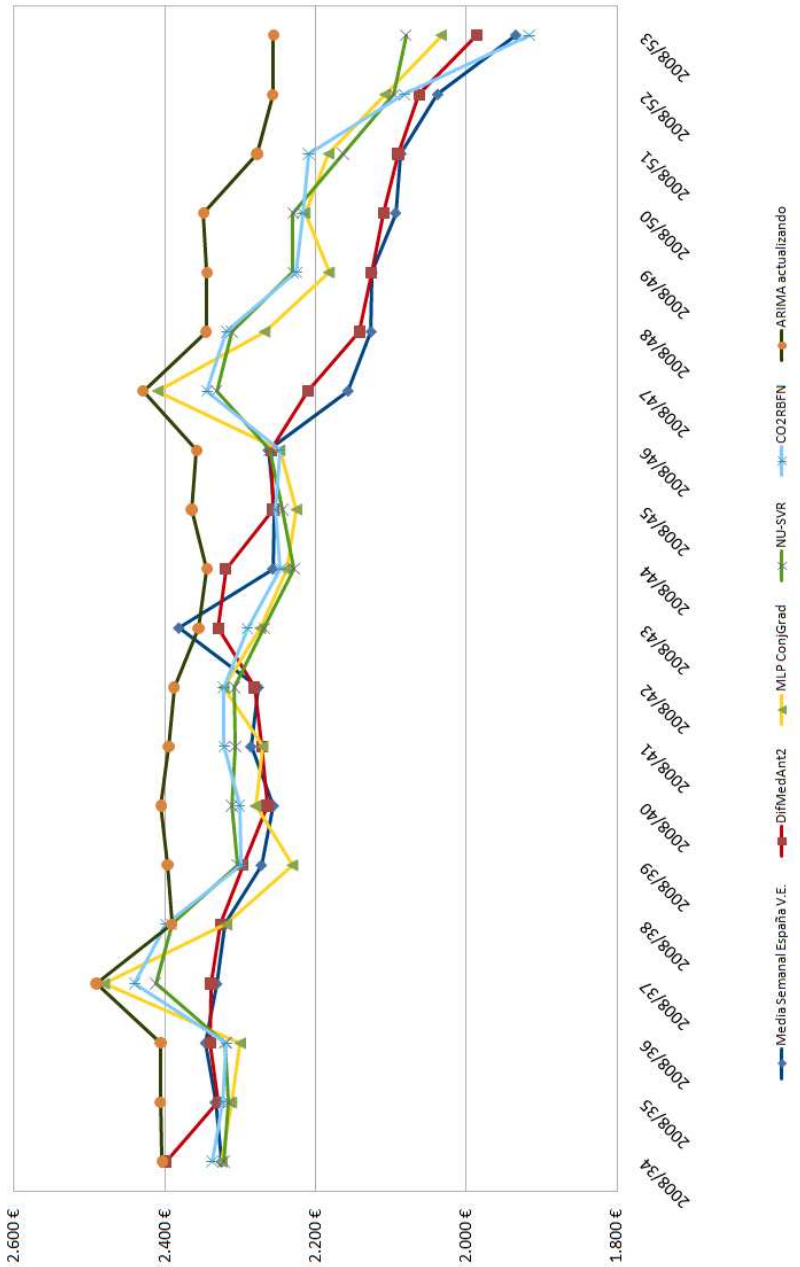


Figura 5.15: *Resultados de la predicción del precio del aceite a cuatro semanas*

- CO^{2}RBFN, es el que obtiene una mejor predicción a una semana y obtiene buenas predicciones, cercanas a los primeros puestos, en la predicción a cuatro semanas.

- La desviación típica de CO^{2}RBFN es prácticamente la más baja de todos los métodos no deterministas, lo que evidencia la robustez del método.

5.8. Conclusiones

Como se ha mencionado con anterioridad surge, en colaboración con otros departamentos de la Universidad, la necesidad de predecir precios en el aceite de oliva virgen extra. En este capítulo se ha abordado la tarea de predicción de series temporales por parte de CO^{2}RBFN, y sus resultados se han comparado con métodos clásicos de análisis de series temporales, métodos ARIMA, y con otros métodos de minería de datos.

Para poder aplicar el modelo a esta nueva tarea de predicción, se han tenido que realizar una serie de adaptaciones mínimas en el algoritmo:

- En clasificación la red presentaba un número de nodos en la capa de salida igual al número de clases. Ahora en predicción el número de nodos en la capa de salida ha de ser 1.

- La medida de error usada en predicción es el error MAPE, por lo que se cambia la forma de calcular el error cometido por las RBFs. Dicho error es uno de los parámetros utilizados para medir la asignación de crédito de las RBFs.

- Se cambia el operador mutación informada. En clasificación éste intenta situar la RBF en el centro de los patrones de su clase. Ahora para predicción el operador utiliza una técnica de gradiente para mover la neurona de forma que se consiga minimizar el error.

- Por último, se cambia el operador de inserción de nuevas RBFs. En clasificación se insertaba el centro de una neurona sobre un patrón mal clasificado, ahora en predicción se inserta en el centro de una zona donde se está cometiendo el máximo error.

En una primera aproximación a este problema de predicción se ha aplicado a tres series temporales existentes en la bibliografía.

Una vez realizado ese estudio preliminar, se aplican los modelos a la predicción de series temporales de precios del aceite de oliva virgen extra. Este estudio se ha realizado en dos fases: predicción en el periodo 2000-2005 y predicción en el periodo 2007-2008.

El comportamiento de los métodos de minería de datos es superior al de ARIMA, método clásico utilizado para la predicción de series econométricas. CO^2RBFN obtiene buenos resultados comparables, y a veces superiores, a los del resto de métodos. Las desviaciones de los resultados de CO^2RBFN son bajas por lo que se muestra que su comportamiento es robusto.

Conclusiones

Para finalizar el contenido de esta memoria, se detallan las conclusiones extraídas del trabajo realizado, se describen las líneas de trabajo futuro y se enumeran la publicaciones asociadas.

El objetivo de la tesis es el desarrollo de métodos híbridos, evolutivos, con enfoque cooperativo-competitivo, para el diseño de Redes de Funciones de Base Radial.

Para lograr este objetivo se ha diseñado una arquitectura donde se hibridan diferentes técnicas *soft-computing*, tales como las redes neuronales, computación evolutiva y lógica difusa. La red neuronal resultante es la encargada del procesamiento de los datos. Para el diseño de la red se utilizan técnicas evolutivas que adaptan los parámetros de la misma. La lógica difusa se utiliza para configurar los mecanismos de aplicación de los operadores de la estrategia evolutiva, representando conocimiento experto.

Se pretende que el modelo propuesto, CO^2RBFN obtenga RBFNs simples, precisas y que generalicen bien. Es decir, redes que con pocas RBFs sean capaces de cubrir el espacio del problema, con un mínimo de solapamiento entre ellas y que dichas redes sean capaces de proporcionar

una respuesta adecuada ante cada nueva entrada que se les presente. Con este objetivo, se diseña su esquema de funcionamiento y componentes:

- La propuesta sigue una estrategia cooperativa-competitiva en la que cada individuo de la población representa una RBF (en este caso es una función gaussiana) y la población entera es la responsable de la solución final. Este paradigma ofrece un marco en el que un individuo representa sólo una parte de la solución, de forma que los individuos cooperan para alcanzar una buena solución (una RBFN que generalice bien para nuevos ejemplos), pero también compiten por su supervivencia, dado que los individuos con peor comportamiento serán eliminados de la población. Gracias a este escenario (de cooperación-competición), se refuerza la explotación por zonas (RBFs con respuesta local), que la mayoría de los ejemplos estén representados (mediante alguna RBF) y se minimiza el solapamiento entre RBFs. Esta guía de diseño en el algoritmo propuesto mejora la interpretabilidad de la RBFN obtenida.

- La asignación de crédito considera tres factores: aportación de la RBF a la salida global de la red, error local cometido por la RBF y solapamiento de la RBF con otras. Con esto se pretende obtener un conjunto de RBFs con una aportación adecuada a la salida de la red, que cometan poco error y con el mínimo solapamiento entre ellas.

- Diseño de operadores evolutivos específicos que analizan el entorno de las RBFs. Se definen cuatro operadores evolutivos que van a poder ser aplicados a una RBF: un operador que elimina una RBF, dos operadores de mutación, y finalmente un operador que mantiene los parámetros de la RBF. Con dichos operadores se intenta lograr un

adecuado equilibrio entre explotación y exploración del espacio de búsqueda.

- Uso de la medida de distancia HVDM que permite, con la mínima pérdida de información, trabajar tanto con valores numéricos como nominales.

- Utilización de un sistema basado en reglas difusas (SBRD) para decidir qué operador aplicar a una RBF durante el diseño. Los factores propuestos para la asignación de crédito se usan como parámetros de entrada para el SBRD y las salidas determinan la probabilidad de aplicación de cada uno de los operadores.

En el capítulo 3, CO^{2}RBFN se ha aplicado a resolver problemas de clasificación. Los resultados con él obtenidos sobre once bases de datos, se han comparado con los obtenidos mediante otros cinco métodos. Los algoritmos utilizados en la comparativa cubren un rango amplio dentro del campo de aprendizaje máquina, incluyendo paradigmas alternativos en el diseño de RBFNs, un modelo diferente de redes neuronales y un modelo basado en árboles de decisión.

Se ha realizado un análisis estadístico de los resultados obtenidos por todos los algoritmos, en cuanto a la precisión en la clasificación y en cuanto a la complejidad del modelo obtenido. El análisis muestra que CO^{2}RBFN obtiene RBFNs con un adecuado equilibrio entre precisión y complejidad, superando a los otros métodos con los que se compara.

CO^{2}RBFN se ha aplicado en el capítulo 4 a la clasificación de bases de datos no balanceadas y sus resultados se han comparado con los obtenidos mediante otros algoritmos dentro del paradigma de las redes neuronales.

Comparando los resultados de los algoritmos aplicados a las bases de datos sin pre-procesar, obtenemos que el mejor comportamiento es el de CO^{2}RBFN y el análisis estadístico de los datos muestra que las diferencias con el resto de métodos son significativas. CO^{2}RBFN obtiene también los resultados con la desviación típica más baja indicando que es un método robusto en situaciones de desbalanceo de clases.

Se ha analizado la influencia del pre-procesamiento en los algoritmos de minería de datos considerados, utilizando SMOTE como método de pre-proceso. Los resultados muestran la utilidad de esta etapa previa en situaciones de desbalanceo, para todos los algoritmos de minería de datos incluidos en el estudo.

Por último se han comparado los resultados obtenidos por los algoritmos, aplicados sobre las bases de datos pre-procesadas mediante SMOTE, y CO^{2}RBFN vuelve a obtener los mejores resultados en la clasificación. De nuevo los test estadísticos aplicados muestran que existen diferencias significativas entre él y el resto de algoritmos.

En el capítulo 5 se adapta el algoritmo para aplicarlo a predicción de series temporales. Para poder aplicar el método a esta nueva tarea de predicción, se han tenido que realizar una serie de modificaciones mínimas en el algoritmo:

- En clasificación la red presentaba un número de nodos en la capa de salida igual al número de clases. Ahora en predicción el número de nodos en la capa de salida ha de ser 1.
- La medida de error usada en predicción es el error MAPE, por lo que se cambia la forma de calcular el error cometido por las RBFs. Dicho

error es uno de los parámetros utilizados para medir la asignación de crédito de las RBFs.

- Se cambia el operador mutación informada. En clasificación éste intenta situar la RBF en el centro de los patrones de su clase. Ahora para predicción el operador utiliza una técnica de gradiente para mover la neurona de forma que se consiga minimizar el error.

- Por último, se cambia el operador de insercción de nuevas RBFs. En clasificación se insertaban en un patrón mal clasificado, ahora en predicción se insertan en el centro de una zona donde se está cometiendo el máximo error.

Se realiza un estudio preliminar del método aplicándolo a series temporales existentes en la bibliografía. Finalmente, se aplica a la predicción de series temporales de precios del aceite de oliva virgen extra. Este estudio se ha realizado en dos fases: predicción en el periodo 2000-2005 y predicción en el periodo 2007-2008.

Los resultados de CO^2RBFN en predicción de series temporales, se han comparado con métodos clásicos de análisis de series temporales, métodos ARIMA, y con otros métodos de minería de datos.

Se puede observar que el comportamiento de los métodos de minería de datos es superior al de ARIMA, método clásico utilizado para la predicción de series econométricas. CO^2RBFN obtiene buenos resultados comparables, y a veces superiores, a los del resto de métodos. Las desviaciones de los resultados de CO^2RBFN son bajas por lo que se muestra que su comportamiento es robusto.

En resumen se puede concluir que el método desarrollado, CO^{2}RBFN, se ha aplicado a clasificación, incluyendo la clasificación en bases de datos no balanceadas, y a predicción de series temporales. Los resultados obtenidos son buenos y el método es robusto.

Como líneas de trabajo futuro se pueden destacar las siguientes:

- Estudio de nuevos operadores evolutivos que exploten información local del entorno de las RBFs.

- Estudio y desarrollo de modelos que aborden el problema de clasificación en entornos no balanceados a nivel de algoritmo.

- Desarrollo de propuestas para el problema de regresión numérica, dado que es la otra tarea a la que se suelen aplicar las RBFNs, junto con la clasificación y predicción de series temporales.

- El enfoque cooperativo-competitivo del modelo, en el que existen RBFs que cooperan por alcanzar una solución global y compiten por su supervivencia, junto con el tipo de red empleada formado por RBFs con respuesta localizada en el espacio, proporciona un entorno adecuado para el desarrollo de una propuesta paralela del método.

- Desarrollo de un nuevo modelo híbrido multiobjetivo que determine de forma conjunta la arquitectura de la RBFN.

- Desarrollo de nuevas propuestas para abordar el problema de clasificación en bases de datos no balanceadas y con múltiples clases.

Publicaciones asociadas al trabajo desarrollado en esta memoria

Revistas internacionales

- Pérez-Godoy, M. D.; Rivera, A. J.; Berlanga, F. J. y del Jesus, M. J.: «CO^2RBFN: An evolutionary cooperative-competitive RBFN design algorithm for classification problems». *Soft Computing*, 2009, doi: 10.1007/s00500-009-0488-z.

- Pérez-Godoy, M. D.; Pérez, P.; Rivera, A. J.; del Jesus, M. J.; Frías, M. P. y Parras, M.: «CO^2RBFN for short-term forecasting of the extra virgin olive oil price in the Spanish market». *International Journal of Hybrid Intelligent Systems*, **7(1)**, pp. 75–87, 2010.

- Pérez-Godoy, M. D.; Fernández, A.; Rivera, A. J. y del Jesus, M. J.: «An Analysis of the CO2RBFN Performance for Imbalanced Data-Sets». *Patterns Recognition Letters (submit)*, 2010.

Congresos Internacionales

- Pérez-Godoy, M. D.; Rivera, A. J.; del Jesus, M. J. y Rojas, I.: «CoEvRBFN: An Approach to Solving the Classification Problem with a Hybrid Cooperative-Coevolutive Algorithm». *of the 9th International Work-Conference on Artificial Neural Network (IWANN'07)*, **4507**, pp. 324–332, 2007.

- Pérez-Godoy, M. D.; Aguilera, J. J.; Berlanga, F. J.; Rivas,

V. M. y Rivera, A. J.: «A preliminary study of the effect of feature selection in evolutionary RBFN design». *Proceedings of the Information Processing and Management of Uncertainty in Knowledge-Based System (IPMU'08)* , pp. 1151-1158, 2008.

- Pérez-Godoy, M. D.; Frías, M. P.; Rivera, A. J.; del Jesus, M. J.; Parras, M. y Torres, F. J: «An study on data mining methods for short-term forecasting of the extra virgin olive oil price in the Spanish market». *Proceedings of the Hybrid Intelligent Systems (HIS'08)*, pp. 943–946, 2008.

- Pérez-Godoy, M. D.; Rivera, A. J.; Fernández, A.; del Jesus, M. J.; y Herrera, F.: «A Preliminar Analysis of CO^2RBFN in Imbalanced Problems». *Proceedings of the 10th International Work-Conference on Artificial Neural Network (IWANN'09)*, **5517**, pp. 57-64, 2009.

- Pérez-Godoy, M. D.; Pérez-Recuerda, P.; Frías, M.P.; Rivera, A. J.; Carmona, C. J. y Parras, M.: «CO^2RBFN for short and medium term forecasting of the extra-virgin olive oil price». *Proceedings of the International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO'10)*, 2010. (Aceptado, no publicado)

Congresos Nacionales

- Pérez-Godoy, M. D.; Rivera, A. J.; del Jesus, M. J. y Rojas, I.: « Optimización de CoEvRBF para aumentar su eficiencia en tareas

de clasificación». *Actas del Simposio de Inteligencia Computacional (SICO'07)*, pp. 193–199, 2007.

- Pérez-Godoy, M. D.; Rivera, A. J.; del Jesus, M. J. y Berlanga, F. J.: «Utilización de un sistema basado en reglas difusas para la aplicación de operadores en un algoritmo cooperativo-competitivo». *Actas del Congreso Español sobre Tecnologías y Lógica Fuzzy (ESTYLF'08)*, pp. 689–694, 2008.

- Pérez-Godoy, M.D.; Pérez, P; Frías, M. P.; Gutiérrez, M.; Rivera, A. J.; del Jesus, M. J.: «Predicción de la Evolución del Precio del Aceite de Oliva Virgen Extra en España Mediante Técnicas de Minería de Datos.» *Actas del XIV Simposium Científico-Técnico de Expoliva'09*, 2009.

- Pérez-Godoy, M. D.; Rivera, A. J. y del Jesus, M. J.: «CO^{2}RBFN: predicción de series temporales con un enfoque cooperativo-competitivo». *Actas del Congreso Español sobre Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB'09)*, pp. 269–276, 2009.

Apéndice A

Tablas de resultados de CO^2RBFN para clasificación

Las tablas que se muestran a continuación contienen los resultados de las ejecuciones completas hechas con CO^2RBFN, para las bases de datos Car, Credit, Glass, Hepatitis, Ionosphere, Iris, Pima, Sonar, Vehicle, Wbcd, Wine.

Las ejecuciones se han realizado con un número de RBFs que oscila entre el número de clases de la base de datos con la que se trabaja y cuatro veces este número de clases.

Tabla A.1: *Base de datos Car*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
4	0,129 ± 0.007	0.207 ± 0.061	87.104	79.350
5	0.122 ± 0.007	0.190 ± 0.048	87.823	**81.007**
6	0.116 ± 0.008	0.204 ± 0.045	88.354	79.572
7	0.112 ± 0.007	0.198 ± 0.046	88.753	80.197
8	0.108 ± 0.009	0.198 ± 0.051	89.186	80.241
9	0.102 ± 0.009	0.200 ± 0.056	89.784	79.998
10	0.098 ± 0.009	0.207 ± 0.053	90.203	79.270
11	0.095 ± 0.009	0.191 ± 0.052	90.543	80.925
12	0.093 ± 0.011	0.200 ± 0.040	90.678	80.011
13	0.089 ± 0.008	0.202 ± 0.047	91.075	79.804
14	0.087 ± 0.010	0.207 ± 0.049	91.279	79.261
15	0.079 ± 0.008	0.220 ± 0.061	92.085	77.974
16	0.079 ± 0.010	0.199 ± 0.045	92.103	80.127

Tabla A.2: *Base de datos Credit*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
2	0.123 ± 0.006	0.158 ± 0.065	87.655	**84.232**
3	0.120 ± 0.004	0.158 ± 0.081	87.974	84.203
4	0.118 ± 0.004	0.177 ± 0.100	88.235	82.261
5	0.116 ± 0.005	0.175 ± 0.096	88.432	82.522
6	0.115 ± 0.003	0.172 ± 0.088	88.470	82.812
7	0.114 ± 0.004	0.167 ± 0.084	88.577	83.275
8	0.113 ± 0.004	0.179 ± 0.094	88.686	82.116

Tabla A.3: *Base de datos Glass*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
7	0.328 ± 0.020	0.358 ± 0.113	67.223	64.216
8	0.319 ± 0.016	0.373 ± 0.103	68.145	62.699
9	0.310 ± 0.017	0.354 ± 0.104	68.987	64.575
10	0.296 ± 0.014	0.360 ± 0.120	70.410	63.990
11	0.290 ± 0.015	0.354 ± 0.111	71.034	64.635
12	0.282 ± 0.017	0.333 ± 0.105	71.812	66.694
13	0.277 ± 0.014	0.332 ± 0.111	72.299	66.778
14	0.275 ± 0.016	0.356 ± 0.116	72.547	64.389
15	0.266 ± 0.015	0.330 ± 0.109	73.399	66.976
16	0.262 ± 0.015	0.343 ± 0.107	73.826	65.654
17	0.258 ± 0.015	0.346 ± 0.104	74.230	65.425
18	0.255 ± 0.016	0.335 ± 0.103	74.490	66.487
19	0.251 ± 0.014	0.340 ± 0.117	74.925	65.980
20	0.250 ± 0.017	0.349 ± 0.109	74.977	65.086
21	0.248 ± 0.014	0.354 ± 0.118	75.164	64.602
22	0.248 ± 0.014	0.323 ± 0.111	75.175	**67.710**
23	0.242 ± 0.016	0.328 ± 0.114	75.798	67.223
24	0.240 ± 0.015	0.332 ± 0.118	75.997	66.763
25	0.236 ± 0.014	0.342 ± 0.116	76.389	65.820
26	0.233 ± 0.015	0.326 ± 0.102	76.700	67.391
27	0.234 ± 0.019	0.329 ± 0.115	76.587	67.065
28	0.235 ± 0.015	0.337 ± 0.107	76.536	66.332

Tabla A.4: *Base de datos Hepatitis*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
2	0.087 ± 0.016	0.168 ± 0.145	91.270	83.228
3	0.074 ± 0.009	0.168 ± 0.138	92.632	83.157
4	0.067 ± 0.011	0.151 ± 0.094	93.261	84.905
5	0.065 ± 0.011	0.151 ± 0.071	93.548	84.914
6	0.064 ± 0.008	0.128 ± 0.107	93.563	87.187
7	0.063 ± 0.010	0.139 ± 0.075	93.749	86.137
8	0.057 ± 0.007	0.126 ± 0.074	94.280	**87.399**

Tabla A.5: *Base de datos Ionosphere*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
2	0.149 ± 0.017	0.171 ± 0.053	85.078	82.926
3	0.134 ± 0.020	0.160 ± 0.049	86.648	84.003
4	0.105 ± 0.015	0.134 ± 0.050	89.485	86.579
5	0.097 ± 0.017	0.119 ± 0.049	90.294	88.069
6	0.087 ± 0.014	0.111 ± 0.043	91.270	88.907
7	0.074 ± 0.013	0.099 ± 0.048	92.643	90.111
8	0.065 ± 0.011	0.086 ± 0.039	93.485	**91.411**

Tabla A.6: *Base de datos Iris*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
3	0.017 ± 0.008	0.045 ± 0.047	98.252	95.467
4	0.012 ± 0.004	0.048 ± 0.055	98.770	95.200
5	0.011 ± 0.004	0.045 ± 0.052	98.919	95.467
6	0.010 ± 0.004	0.037 ± 0.042	99.007	**96.267**
7	0.010 ± 0.005	0.037 ± 0.050	99.007	96.267
8	0.009 ± 0.004	0.044 ± 0.054	99.067	95.600
9	0.009 ± 0.004	0.052 ± 0.050	99.081	94.800
10	0.009 ± 0.004	0.044 ± 0.049	99.141	95.600
11	0.009 ± 0.004	0.048 ± 0.050	99.111	95.200
12	0.008 ± 0.004	0.040 ± 0.046	99.230	96.000

Tabla A.7: *Base de datos Pima*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
2	0.239 ± 0.024	0.260 ± 0.051	76.068	74.001
3	0.227 ± 0.010	0.248 ± 0.043	77.341	75.218
4	0.221 ± 0.007	0.240 ± 0.049	77.873	**75.950**
5	0.218 ± 0.006	0.247 ± 0.047	78.224	75.252
6	0.215 ± 0.006	0.243 ± 0.047	78.516	75.716
7	0.213 ± 0.006	0.244 ± 0.045	78.675	75.638
8	0.212 ± 0.006	0.242 ± 0.044	78.814	75.796

Tabla A.8: *Base de datos Sonar*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
2	0.247 ± 0.020	0.282 ± 0.111	75.299	71.762
3	0.235 ± 0.019	0.261 ± 0.114	76.517	73.910
4	0.222 ± 0.015	0.283 ± 0.097	77.756	71.705
5	0.219 ± 0.020	0.285 ± 0.092	78.120	71.514
6	0.210 ± 0.013	0.279 ± 0.099	78.952	72.114
7	0.206 ± 0.016	0.271 ± 0.090	79.412	72.948
8	0.201 ± 0.012	0.249 ± 0.098	79.882	**75.086**

Tabla A.9: *Base de datos Vehicle*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
4	0.432 ± 0.021	0.446 ± 0.048	56.782	55.391
5	0.400 ± 0.019	0.415 ± 0.054	60.032	58.489
6	0.389 ± 0.021	0.405 ± 0.054	61.135	59.520
7	0.369 ± 0.026	0.381 ± 0.045	63.092	61.912
8	0.357 ± 0.019	0.386 ± 0.043	64.303	61.390
9	0.343 ± 0.016	0.370 ± 0.050	65.721	62.954
10	0.332 ± 0.016	0.350 ± 0.047	66.840	64.979
11	0.318 ± 0.015	0.353 ± 0.046	68.201	64.703
12	0.311 ± 0.013	0.334 ± 0.045	68.873	66.645
13	0.307 ± 0.010	0.326 ± 0.041	69.320	67.414
14	0.296 ± 0.014	0.318 ± 0.038	70.433	68.201
15	0.292 ± 0.014	0.316 ± 0.044	70.846	68.433
16	0.287 ± 0.012	0.308 ± 0.047	71.253	**69.192**

Tabla A.10: *Base de datos Wbcd*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
2	0.024 ± 0.004	0.032 ± 0.023	97.581	96.769
3	0.023 ± 0.002	0.033 ± 0.021	97.743	96.741
4	0.023 ± 0.002	0.033 ± 0.021	97.749	96.739
5	0.022 ± 0.002	0.029 ± 0.018	97.797	**97.083**
6	0.022 ± 0.002	0.032 ± 0.019	97.794	96.792
7	0.022 ± 0.002	0.033 ± 0.020	97.803	96.740
8	0.022 ± 0.002	0.029 ± 0.020	97.813	97.054

Tabla A.11: *Base de datos Wine*

#Nodos	Error Entrenamiento	Error Test	% Acierto Entrenamiento	% Acierto Test
3	0.008 ± 0.006	0.051 ± 0.060	99.189	94.915
4	0.004 ± 0.004	0.057 ± 0.066	99.575	94.261
5	0.003 ± 0.003	0.043 ± 0.061	99.738	95.732
6	0.002 ± 0.003	0.038 ± 0.045	99.813	96.157
7	0.001 ± 0.002	0.033 ± 0.046	99.913	**96.739**
8	0.001 ± 0.002	0.036 ± 0.040	99.950	96.412
9	0.000 ± 0.001	0.037 ± 0.046	99.975	96.281
10	0.000 ± 0.000	0.045 ± 0.054	100.000	95.484
11	0.000 ± 0.001	0.038 ± 0.047	99.975	96.196
12	0.000 ± 0.000	0.038 ± 0.050	100.000	96.190

Apéndice B

Descripción del algoritmo GeneticRBFN

Al ser CO^2RBFN un método evolutivo de diseño de RBFNs, con un enfoque en el que cada individuo de la población es una sola neurona o RBF se considera oportuno compararlo con otro método evolutivo de diseño de RBFNs que tenga un enfoque evolutivo tipo *Pittsburgh* en el que cada uno de los individuos de la población es una red completa. GeneticRBFN, es el algoritmo que se ha desarrollado, en el trabajo de esta memoria, siguiendo dicho enfoque. Las líneas de diseño que se han seguido son las clásicas que se dan para este tipo de algoritmos [Harpham y otros, 2004]. Para que las comparaciones con CO^2RBFN se hagan bajo las mismas condiciones, se han establecido en la implementación de GeneticRBFN ciertas características de operación iguales a las de él, tales como analogía en los operadores y la métrica HVDM como medida de distancia.

El método sigue la tradicional aproximación evolutiva con enfoque *Pittsburgh* para el diseño de RBFNs: cada individuo es una red completa.

El objetivo del proceso evolutivo es minimizar el error en la clasificación. Los principales pasos de este algoritmo se muestran en la la figura B.1. A

```
Inicialización
Mientras (No Fin) Hacer
    Selección
    Recombinación
    Mutación
    Evaluación/Entrenamiento de la RBFN
Fin_Mientras
```

Figura B.1: *Principales pasos de GeneticRBFN*

continuación se va a describir cada una de las fases del algoritmo:

- Inicialización: La etapa de inicialización es la misma que la utilizada por CO^{2}RBFN. Las RBFs se centrarán, de forma equidistante, para cada individuo o RBFN.

- Operadores Genéticos: El algoritmo utiliza un operador de cruce y varios de mutación.

    - Operador de cruce. El número de RBFs de cada individuo (red) se establece como un valor dentro de un rango delimitado por un valor mínimo y un máximo. El valor mínimo es el número de RBFs del padre con menor número de RFBs, mientras que el valor máximo coincidirá con el número de RBFs del padre que tenga más RBFs.

    - Operadores de mutación. Han sido implementados seis operadores de mutación, que son usados frecuentemente en la bibliografía

especializada ([Harpham y otros, 2004]. Dichos operadores los vamos a clasificar como operadores aleatorios e informados. Los operadores aleatorios son:

- DelRandRBFs: elimina aleatoriamente k RBFs, donde k es un porcentaje, pm, del número total de RBFs en la red.
- InsRandRBFs: agrega aleatoriamente k RBFs, siendo k un porcentaje, pm, del número total de RBFs en la red.
- ModCentRBFs: modifica aleatoriamente el centro de k RBFs, siendo k un porcentaje, pm, del número total de RBFs en la red. El centro de la función base se modificará en un porcentaje, pr, de la medida de su radio.
- ModWidtRBFs: modifica aleatoriamente el centro de k RBFs, siendo k un porcentaje, pm, del número total de neuronas de la red. El radio de la neurona se modificará en un porcentaje, pr, sobre su radio antiguo.

Los operadores informados explotan la información local del entorno de las neuronas.

- DelInfRBFs: elimina k RBFs, donde k es un porcentaje, pm, del número total de RBFs en la red.
- InsInfRBFs: inserta k RBFs en la red en zonas no cubiertas por ninguna de las neuronas que forman parte de la red. La cantidad k se obtiene como un porcentaje, pm, del total de neuronas de la red.

La nueva población se obtiene mediante un torneo entre la población intermedia formada por los padres y los hijos. Para promover la diversidad de la población se usa un valor bajo para

el tamaño del torneo ($k = 3$).

- Entrenamiento de los pesos Los pesos se entrenan mediante el algoritmo LMS. Los parámetros usados en dicho algoritmo son los valores estándar.

- Evaluación de los individuos El *fitness* que se define para cada individuo/RBFN es el error de clasificación que comete sobre el problema dado.

Para incrementar la eficiencia del algoritmo genético, el espacio de búsqueda del algoritmo se ha reducido drásticamente. Esto se hace así ya que como es bien conocido, en el enfoque *Pittsburgh*, donde el único objetivo es optimizar el error de clasificación, la complejidad de los individuos (es decir, el número de RBFs) crece de una forma incontrolada (ya que normalmente una red con más neuronas consigue mejores porcentajes de error que una con menos neuronas). En la experimentación realizada se ha reducido el espacio de búsqueda fijando la complejidad máxima (y así el tamaño del cromosoma) entre un número mínimo y máximo de RBFs. El número mínimo de neuronas se ha hecho igual al número de clases existentes en el problema y el número máximo igual a cuatro veces dicho número de clases.

En la tabla B.1 se muestran los valores dados a los parámetros del algoritmo en las experimentaciones realizadas.

Tabla B.1: *Parámetros de GeneticRBFN*

Parámetro	*Valor*
Generaciones del ciclo principal	200
Individuos	40
Longitud del cromosoma	Min = número de clases Max = 4 · número de clases
Probabilidad cruce	0.6
Probabilidad mutación	0.1
Porcentaje mutación radio	0.2
Porcentaje mutación centro	0.2
Tamaño del torneo	3

Apéndice C

Parámetros de los algoritmos usados en la experimentación e implementados en Keel

En este apéndice se muestran los parámetros de los algoritmos utilizados en la experimentación. Los valores son los aconsejados por los autores de los algoritmos y se muestran como aparecen en la herramienta KEEL que es la que se ha utilizado para la ejecución de los mismos.

Tabla C.1: *Parámetros de los algoritmos en clasificación*

Algoritmo	*Parámetro*	*Valor*
C4.5	pruned	true
	condidence	0.25
	instancesPerLeaf	2
MLP-Back	hidden_layers	2
	hidden_nodes	15
	transfer	Htan
	eta	0.15
	alpha	0.10
	lambda	0.0
	test_data	true
	validation_data	false
	cross_validation	false
	cycles	10000
	improve	0.01
	problem	
	tipify_inputs	true
	verbose	false
	saveAll	false
RBFN-Decr	percent	0.1
	nNeuronsIni	20
	alpha	0.3
RBFN-Incr	epsilon	0.1
	alpha	0.3
	delta	0.5

Tabla C.2: *Parámetros de los algoritmos en clasificación no balanceada*

Algoritmo	*Parámetro*	*Valor*
LVQ	iterations	100
	neurons	20
	alpha	0.3
	nu	0.8
MLP-Back	hidden_layers	2
	hidden_nodes	15
	transfer	Htan
	eta	0.15
	alpha	0.10
	lambda	0.0
	test_data	true
	validation_data	false
	cross_validation	false
	cycles	10000
	improve	0.01
	problem	Classification
	tipify_inputs	true
	verbose	false
	saveAll	false
MLP-Grad	topologymlp	10
RBFN-Decr	percent 0.1	
	nNeuronsIni	20
	alpha	0.3
RBFN-Incr	epsilon	0.1
	alpha	0.3
	delta	0.5

Tabla C.3: *Parámetros de los algoritmos en predicción de series temporales*

Algoritmo	*Parámetro*	*Valor*
Fuzzy-GAP	numlabels	3
	numrules	8
	popsize	30
	numisland	2
	steady	1
	numitera	10000
	toursize	4
	probmuta	0.01
	amplmuta	1
	probmigra	0.001
	proboptimlocal	0.00
	numoptimlocal	0
	idoptimlocal	0
	nichinggap	0
	maxindniche	8
	probintraniche	0.75
	probcrossga	0.5
	probmutaga	0.5
	lenchaingag	10
	maxtreeheight	8
MLP-Grad	topologymlp	10
NU-SVR	KERNELtype	RBF
	C	100.0
	eps	0.001
	degree	1
	gamma	0.01
	coef0	0.0
	nu	0.1
	p	1.0
	shinking	1
RBFN-LMS	neurons	50

Apéndice D

Tests de contraste de hipótesis no paramétricos

Un contraste o test de hipótesis es una técnica de inferencia estadística que permite, a partir de los datos obtenidos de una (o varias) muestra observada, decidir si se acepta o no una hipótesis formulada sobre una (o varias) población.

Una hipótesis estadística es una asunción relativa a una o varias poblaciones, que puede ser cierta o no. Las hipótesis estadísticas se pueden contrastar con la información extraída de las muestras y se puede cometer un error, tanto si se aceptan como si se rechazan.

Una hipótesis estadística puede ser:

- Paramétrica: es una afirmación sobre los valores de los parámetros poblacionales desconocidos.

- No paramétrica: es una afirmación sobre alguna característica estadística de la población en estudio. Por ejemplo, las observaciones son

independientes, la distribución de la variable en estudio es normal, la distribución es simétrica, etc.

La hipótesis que se formula se denomina hipótesis de trabajo o nula y se denota H_0, a la hipótesis contraria se le denomina hipótesis alternativa, H_1. El test de hipótesis decidirá, basándose en la muestra observada, si se acepta o no la hipótesis nula formulada frente a la hipótesis alternativa.

En un contraste de hipótesis se pueden cometer dos tipos de errores:

- Error tipo I, se rechaza la hipótesis H_0 cuando es cierta.
- Error tipo II, se acepta la hipótesis H_0 cuando es falsa.

Sólo se puede cometer uno de los dos tipos de error y, en la mayoría de las situaciones, se desea controlar la probabilidad de cometer un error de tipo I. Se denomina *nivel de significación o confianza* de un contraste a la probabilidad de cometer un error tipo I, se denota por α (es decir, la probabilidad de que el estadístico de contraste caiga en la región de rechazo):

$$\alpha = P(rechazar H_0 | H_0 es cierta) \tag{D.1}$$

El nivel de confianza se ha de decidir de antemano, de forma que se establece la probabilidad máxima que se está dispuesto a asumir de rechazar la hipótesis nula cuando es cierta. Dicho nivel lo elige el usuario. La selección de dicho nivel conduce a dividir en dos regiones el conjunto de posibles valores del estadístico de contraste: la región de Rechazo, con probabilidad α, bajo H_0 y la región de Aceptación, con probabilidad $1 - \alpha$, bajo H_0.

Si el estadístico de contraste calculado por el test (también conocido

como *p-valor*), $\hat{d}$, toma un valor perteneciente a la región de aceptación, entonces no existen evidencias suficientes para rechazar la hipótesis nula con un nivel de significación α y se dice que el contraste *estadísticamente no es significativo*. Si, por el contrario, el estadístico cae en la región de rechazo entonces se asume que los datos no son compatibles con la hipótesis nula y se rechaza a un nivel de significación α. En este supuesto se dice que el contraste es *estadísticamente significativo* (ecuación D.2).

$$\begin{array}{lll} \text{Si} \quad \hat{d} \in \text{Región de Aceptación} & \Longrightarrow & \text{Se acepta} \quad H_0 \\ \text{Si} \quad \hat{d} \in \text{Región de Rechazo} & \Longrightarrow & \text{Se rechaza} \quad H_0 \end{array} \tag{D.2}$$

En [Sheskin, 2000], la distinción que se hace entre test *paramétricos* y *no paramétricos* se basa en el nivel de medida representado por los datos que van a ser analizados. De esta manera, un test paramétrico es aquel que utiliza datos con valores reales pertenecientes a un intervalo. Esto no implica que siempre que dispongamos de este tipo de datos, haya que usar un test paramétrico. Puede darse el caso de que una o más suposiciones iniciales para el uso de los test paramétricos se incumplan, haciendo que el análisis estadístico pierda credibilidad.

Para utilizar los test paramétricos es necesario que cumplan las siguientes condiciones [Sheskin, 2000; Zar, 1999]:

- Independencia: En estadística, dos sucesos son independientes cuando el que haya ocurrido uno de ellos no modifica la probabilidad de ocurrencia del otro.

- Normalidad: Una observación es normal cuando su comportamiento sigue una distribución normal o de Gauss con una determinada media

μ y varianza σ.

- Homocedasticidad: Se dice que existe homocedasticidad cuando la varianza de los errores estocásticos de la regresión son los mismos para cada observación.

En la práctica es difícil conocer la forma funcional de la distribución de donde proceden los datos, por lo que se necesita aplicar métodos que no requieran el conocimiento de esa distribución pero que sí nos permitan hacer inferencias sobre la población. A estos métodos los llamamos métodos no paramétricos o de libre distribución, ya que no se basan en la hipótesis de que los datos sigan una determinada distribución de probabilidad. Las condiciones de aplicación de éstos métodos son menos restrictivas que las exigidas en los métodos paramétricos.

En general los contrastes no paramétricos, necesitan pocas hipótesis para su planteamiento y la mayoría de las veces son más fáciles de aplicar que los contrastes paramétricos. Además, hasta ahora, en los contrastes paramétricos se analizan caracteres cuantitativos y por tanto perfectamente cuantificables, sin embargo en los contrastes no paramétricos podemos trabajar con características (o variables) ordinales, en las que solo interesa el orden o rango, e incluso nominales, en las que los valores se utilizan para indicar las distintas modalidades o categorías. Esto nos permite ampliar el campo de aplicación de los test de hipótesis. Como norma general, un test no paramétrico es menos restrictivo que un paramétrico, aunque menos robusto que un paramétrico cuya aplicación se realiza sobre datos que cumplen todas las condiciones necesarias.

En nuestro caso los test de contraste los vamos a utilizar para comparar

los resultados obtenidos por los diferentes algoritmos y ver si existen o no diferencias significativas entre ellos. Es decir:

$$\begin{aligned} H_0 &= \text{Los resultados de los diferentes métodos son similares} \\ H_1 &= \text{Los resultados de los diferentes métodos difieren} \end{aligned} \tag{D.3}$$

Al aplicar el test, si la hipótesis nula se rechaza existirán diferencias significativas entre los resultados obtenidos por los métodos y por el contrario si no se rechaza la hipótesis nula, no existen diferencias significativas entre los resultados, es decir, los métodos obtienen resultados equivalentes.

A continuación se va a proceder a describir los métodos estadísticos no paramétricos que se han utilizado.

Test de Friedman

Es el test no paramétrico equivalente al test paramétrico ANOVA. Calcula un ranking de los resultados obtenidos por cada algoritmo (r_j para cada algoritmo j, habiendo k algoritmos) y por cada base de datos, y le asigna al mejor el ranking 1 y al peor el ranking k. Bajo la hipótesis nula de que los resultados de todos los algoritmos son equivalentes, y por tanto sus rankings son similares. El estadístico de Friedman (ecuación D.4)

$$\chi_F^2 = \frac{12N_{ds}}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{D.4}$$

se distribuye de acuerdo con una distribución χ_F^2 con $k-1$ grados de libertad, siendo $R_j = \frac{1}{N_{ds}} \sum_i r_i^2$ y N_{ds} el número de bases de datos. El valor crítico par el estadístico de Friedman coincide con los establecidos en la distribución χ^2 cuando $Nds > 10$ y $k > 5$.

Test de Iman-Davenport

El test de Iman-Davenport [Sheskin, 2006] se deriva a partir del test de Friedman, dado que este último produce efectos conservativos indeseables. Este estadístico se calcula como indica la ecuación D.5

$$F_F = \frac{(N_{ds} - 1)\chi_F^2}{N_{ds}(k-1) - \chi_F^2} \tag{D.5}$$

y sigue una distribución F con $k-1$ y $(k-1)(N_{ds}-1)$ grados de libertad. Las tablas que muestran los valores críticos del test se pueden encontrar en [Sheskin, 2006; Zar, 1999].

Test de Holm

El test de Holm [Holm, 1979] sirve para hacer comparaciones múltiples, trabaja con un algoritmo de control (el mejor algoritmo) y compara éste con el resto de métodos. El estadístico que calcula para comparar el i-ésimo y j-ésimo método es el que se muestra en la ecuación D.6.

$$z = \frac{R_i - R_j}{\sqrt{\frac{k(k+1)}{6N_{ds}}}} \tag{D.6}$$

El valor z se usa para encontrar la probabilidad correspondiente en una tabla que represente la distribución normal, luego se compara con el nivel de confianza (α) apropiado. Se ordenan los procedimientos secuencialmente por su importancia. Sean $p_1, p_2, \ldots$ los $p-$valores ordenados, de forma que $p_1 \leq p_2 \leq \ldots \leq p_{k-1}$. El test de Holm compara cada p_i with $\alpha/(k-i)$ comenzando por el que tiene el valor p menos significativo. Si p_1 es menor que $\alpha/(k-1)$, la correspondiente hipótesis se rechaza y se pasa a comparar p_2 con $\alpha/(k-2)$. Si se rechaza la segunda hipótesis, el test continúa con la

tercera y así sucesivamente. En el momento en el que la hipótesis nula no se puede rechazar ya el resto tampoco.

Test de Ranking de Signos de Wilcoxon

El test de Wilcoxon [Wilcoxon, 1945] es el análogo al *paired t-test* no paramétrico. Es un test aplicado a parejas de algoritmos y permite detectar la existencia de diferencias significativas entre el comportamiento de ambos. Su funcionamiento se basa en calcular las diferencias entre los resultados de dos algoritmos y calcular un ranking utilizando dicho valor, ignorando signos. En este caso el ranking va desde 1 a N, en vez de hasta k.

Sea d_i la diferencia entre las medidas de ejecución de los dos algoritmos en la i-ésima ejecución considerando los N_{ds} conjuntos de datos. Se ordenan las diferencias obtenidas en un ranking, según su valor absoluto. Se calcula R^+ como la suma de los ranking de los conjuntos de datos en los que el primer algoritmo supera al segundo, y R^- la suma de los otros ranking. Los rankings con valor $d_i = 0$ se reparten uniformemente entre las dos sumas anteriores, si hay un número impar se desecha alguno (ecuación D.7).

$$\begin{aligned} R^+ &= \sum_{di>0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i), \\ R^- &= \sum_{di<0} rank(d_i) + \frac{1}{2}\sum_{d_i=0} rank(d_i) \end{aligned} \tag{D.7}$$

Sea T el valor menor de las sumas, $T = min(R^+, R^-)$, si T es menor o igual que el valor de la distribución de T de Wilcoxon para N_{ds} grados de libertad (las tablas se pueden encontrar en [Zar, 1999]), se rechaza la hipótesis nula de igualdad de las medias y el algoritmo asociado al mayor de los valores es el mejor.

Bibliografía

Akaike, H.: «A new look at statistical model identification». *IEEE. Transactions on Automatic Control*, 1974, **19**, pp. 716–723.

Al-Haddad, L.; Morris, C.W. y Boddy, L.: «Training radial basis function neural networks: effects of training set size and imbalanced training sets». *Journal of Microbiological Methods*, 2000, **43**, pp. 33–44.

Alcalá-Fdez, J.; Sánchez, L.; García, S.; Del Jesus, M.J.; Ventura, S.; Garrell, J.M.; Otero, J.; Romero, C.; Bacardit, J.; Rivas, V.; Fernández, J.C. y Herrera, F.: «KEEL: A Software Tool to Assess Evolutionary Algorithms for Data Mining Problems». *Soft Computing*, 2009, **13(3)**, pp. 307–318.

Alejo, R.; García, V.; Sotoca, J.M.; Mollineda, R.A. y Sánchez, J.S.: «Improving the performance of the RBF neural networks trained with imbalanced samples». En: *Proceedings of the 9th International Work-Conference on Artificial Neural Networks (IWANN'07)*, volumen 4507, pp. 162–169, 2007.

Ampazis, N. y Perantonis, S.J.: «Two highly efficient second-order

algorithms for training feedforwards networks». *IEEE Transactions on Neural Networks*, 2002, **13(3)**, pp. 1064–1074.

ANDERSEN, H. y TSOI, C.: «A constructive algorithm for the training of a multilayer perceptron based on the genetic algorithm». *Complex Systems*, 1993, **7(4)**, pp. 249–268.

ANDERSON, J. y MURPHY, G.: «Psychological conceps in a parallel system». *Physica D*, 1986, **2**, pp. 318–336.

ASUNCION, A. y NEWMAN, D.J.: «UCI Machine Learning Repository». *University of California, Irvine, School of Information and Computer Science*, 2007a.
`http://www.ics.uci.edu/ mlearn/MLRepository.html`

ASUNCION, A. y NEWMAN, D.J.: «UCI Machine Learning Repository», 2007b.
`http://www.ics.uci.edu/ mlearn/MLRepository.html`

AZADEH, A.; SABERI, M.; GHADERI, S.F.; GITIFOROUZ, A. y EBRAHIMIPOUR, V.: «Improved estimation of electricity demand function by integration of fuzzy system and data mining approach». *Energy Conversion and Management*, 2008, **49(8)**, pp. 2165–2177.

BALAKRISHNAN, K. y HONAVAR, V.: «Evolutionary design of neural architecture-a preliminary taxonomy and guide to literature». *Technical report, AI Research Group, CS-TR 95-01 Billings SA, Zheng GL*, 1995.
`citeseer.ist.psu.edu/balakrishnan95evolutionary.html`

BARANDELA, R.; SÁNCHEZ, J.S.; GARCÍA, V. y RANGEL, E.: «Strategies for

learning in class imbalance problems». *Pattern Recognition*, 2003, **36(3)**, pp. 849–851.

Batista, G.E.A.P.A.; Prati, R.C. y Monard, M.C.: «A study of the behaviour of several methods for balancing machine learning training data». *SIGKDD Explorations*, 2004, **6(1)**, pp. 20–29.

Beasly, D.; Bull, D. y Martin, R.: «A sequential niche technique for multimodal function optimization». *Evolutionary Computation*, 1993, **1**, pp. 101–125.

Belew, R.; McInerney, J. y Schaudolph, N.: «Evolving Networks: Using Genetic Algorithm with Connectionist Learning». En: *Proceedings of the Second Artificial Life Conference*, pp. 511–547, 1991.

Bernier, J.; Ortega, J.; Ros, E.; Rojas, I. y Prieto, A.: «A Quantitative Study of Fault Tolerance, Noise Immunity and Generalization Ability of MLPs». *Neural Computation*, 2000, **12**, pp. 2941–2964.

Bezdek, J.: *Pattern Recognition with Fuzzy Objective Function Algorithms*. Springer, 1981.

Bezdek, J.C. y Kuncheva, L.I.: «Nearest prototype classifier designs: An experimental study». *International Journal of Intelligent Systems*, 2001, **16(12)**, pp. 1445–1473.

Billings, S.A y Zheng, G.L.: «Radial basis function network configuration using genetic algorithms». *Neural Networks*, 1995, **8(6)**, pp. 877–890.

Bonissone, P.: «Hybrid sont computing system: where are we going?» En: *14th European Conference on Artificial Intelligente (ECAI 2000)*, pp. 739–746, 2000.

Box, G. y Jenkins, G.: *Time series analysis: forecasting and control*. Revised edition. San Francisco: Holden Day, 1976.

Bradley, A.P.: «The Use of the Area Under the ROC Curve in the Evaluation of Machine Learning Algorithms». *Pattern Recognition*, 1997, **30(7)**, pp. 1145–1159.

Broomhead, D. y Lowe, D.: «Multivariable functional interpolation and adaptive networks». *Complex Systems*, 1988, **2**, pp. 321–355.

Bruzzone, L. y Serpico, S.B.: «Classification of imbalanced remote-sensing data by neural networks». *Patterns Recognition Letters*, 1997, **18(11–13)**, pp. 1323–1328.

Buchtala, O.; Klimek, M. y Sick, B.: «Evolutionary optimization of radial basis function classifiers for data mining applications». *IEEE Transactions on System, Man, and Cybernetics, B*, 2005, **35(5)**, pp. 928–947.

Burdsall, B. y Giraud-Carrier, C.: «GA-RBF: a self optimising RBF network». En: *Proceedings of the Conference on Artificial Neural Networks and Genetic Algorithms*, pp. 348–351. Springer, Berlin Heidelberg New York, 1997.

Carpenter, G. y Grossberg, S.: «A massively parallel architecture for a self-organizing neural pattern recognition machine». *Computer Vision, Graphics, and Image Processing*, 1983, **37**, pp. 54–115.

Carpenter, G. y Grossberg, S.: «Art 2: Self-organization of stable category recognition codes for analog output patterns». *Applied Optics*, 1987a, **26**, pp. 4919–4930.

Carpenter, G. y Grossberg, S.: «Art 3: hierarchical search: Chemical trnsmitter in self-organizing pattern recognition architectures». En: *Proceedings of the International Joint Conference on Neural Networks*, volumen 2, pp. 30–33, 1987b.

Castillo, P. A.; Castellano, J. G.; Merelo, J. J. y Prieto, A.: «Diseño de Redes Neuronales Artificiales mediante Algoritmos Evolutivos». *Inteligencia artificial: Revista Iberoamericana de Inteligencia Artificial*, 2001, **5(14)**, pp. 2–32.

Chaiyaratana, N. y Zalzala, A.M.S.: «Evolving hybrid RBF-MLP networks using combined genetic/unsupervised/supervised learning». En: *Proceedings of the UKACC International Conference on CONTROL, Swansea, UK*, volumen 1, pp. 330–335, 1998.

Chakaravathy, S. V. y Ghosh, J.: «Scale based clustering using a radial basis function network». *IEEE Transaction on Neural Networks*, 1996, **2(5)**, pp. 1250–1261.

Chalmers, D.: «The evolution of learning: an experiment in genetic connectionism». En: *Proceedings of the 1990 Connectionist Models Summer School*, pp. 81–90, 1990.

Chawla, N.V.; Bowyer, K.W.; Hall, L.O. y Kegelmeyer, W.P.: «Smote: synthetic minority over-sampling technique». *Journal of Artificial Intelligent Research*, 2002, **16**, pp. 321–357.

Chawla, N.V.; Cieslak, D. A. y Joshi, A.: «Automatically countering imblance and its empirical relationship to cost». *Data Minig and Knowledge Dicovery*, 2008, **17(2)**, pp. 225–252.

Chawla, N.V.; Japkowicz, N. y Kolcz, A.: «Special issue on learning from imbalanced data sets». *SIGKDD Explorations NewsLetters*, 2004, **6(1)**, pp. 1–6.

Chen, M.-C.; Chen, L.-S.; Hsu, C.-C. y Zeng, W.-R.: «An information granulation based data mining approach for classifying imbalanced data». *Information Sciences*, 2008, **178(16)**, pp. 3214–3227.

Chen, S.; Billings, S.A.; Cowan, C.F.N. y Grant, P.W.: «Practical identification of narmax models using radial basis functions». *International Journal of Control*, 1990, **52(6)**, pp. 1327–1350.

Chen, S.; Chung, E. y Alkadhimi, K.: «Regularized orthogonal least squares algorithm for constructing radial basis function networks». *International Journal of Control*, 1996, **64(5)**, pp. 829–837.

Chen, S.; Cowan, C. y Grant, P.: «Orthogonal least squares learning algorithm for radial basis function networks». *IEEE Transactions on Neural Networks*, 1991, **2**, pp. 302–309.

Chen, S.; Wu, Y. y Luk, B.L.: «Combined genetic algorithm optimization and regularized orthogonal least squares learning for radial basis function networks». *IEEE Transactions on Neural Networks*, 1999, **10(5)**, pp. 1239–1243.

Cheng, V.; Li, c.H.; Kwok, J.T. y Li, C.K.: «Dissimilarity learning for nominal data». *Pattern Recognition*, 2004, **37**, pp. 1471–1477.

Chien, C.: «Fuzzy logic in control systems: fuzzy logic controller». *IEEE Transaction on System , Man and Cybernetics*, 1990, **20(2)**, pp. 404–435.

Cliff, L.: «Visualization of Matrix Singular Value Decomposition». *Mathematics Magazine*, 1983, pp. 161–167.

Co, H.C. y Boosarawongse, R.: «Forecasting Thailand's rice export: Statistical techniques vs. artificial neural networks». *Computers and Industrial Engineering*, 2007, **53(4)**, pp. 610–627.

Darwin, C.: *The Origin of Species*. John Murray, 1859.

Dawson, C.W.; Wilby, R.L.; Harpham, C.; Brown, M.R.; Cranston, E. y Darby, E.J.: «Modelling Ranunculus presence in the Rivers test and Itchen using artificial neural networks». En: *Proceedings of the International Conference on GeoComputation, Greenwich, UK*, , 2000.

Deb, K.: *Evolutionary Computation 1: Basic Algorithms and Operators*. capítulo Introduction to selection, pp. 166–171. Institute of Physic Publishing, 2000.

Deb, K.: *Multi-objective optimization using evolutionary algorithms*. Wiley, 1st edition, 2001.

Demšar, J.: «Statistical Comparisons of Classifiers over Multiple Data Sets». *Journal of Machine Learning Research*, 2006, **7**, pp. 1–30.

Domingos, P.: «Metacost: a general method for making classifiers cost sensitive». En: *Proceedings of the 5th International Conference on Knowledge Discovery and Data Mining*, pp. 155–164, 1999.

Drummond, C. y Holte, R. C.: «Explicitly representing expected cost: an alternative to ROC representation». En: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining (KDD'00)*, pp. 198–207. ACM, New York, NY, USA, 2000.

Du, H. y Zhang, N.: «Time series prediction using evolving radial basis function networks with new enconding scheme». *Neurocomputing*, 2008, **71**, pp. 1388–1400.

Duda, R.O. y Hart, P.E.: *Pattern Classification and Scene Analysis*. Wiley, New York, 1973.

Edwards, A.W.F.: *Likelihood*. Cambridge University Press, Cambridge, United Kingdom, 1972.

Eiben, A. E. y Smith, J.E. : *Introduction to Evolutionary Computing*. Springer, 2003.

Er, M.J.; Chen, W. y Wu, S.: «High-speed face recognition base on discrete cosine transform and RBF neural networks». *IEEE Transactions on Neural Networks*, 2005, **16(3)**, pp. 679–691.

Eshelman, L. y Schaffer, J.: «Preventing premature convergence in genetic algorithms by preventing incest». En: Morgan Kaufmann (Ed.), *Proceedings in the Fourth International Conference on Genetic Algorithms*, pp. 115–122, 1991.

Esposito, A.; Marinaro, M.; Oricchio, D. y Scarpetta, S.: «Approximation of continuous and discontinuous mappings by a growing neural RBF-based algorithm». *Neural Networks*, 2000a, **13(6)**, pp. 651–665.

Esposito, F.; Malerba, D.; Tamma, V. y Bock, H.H.: «Classical resemblance measures». *In H.-H. Bock and E. Diday (Eds.). Analysis of Symbolic Data. Exploratory methods for extracting statistical information*

from complex data, Series: Studies in Classification, Data Analysis, and Knowledge Organization. Springer-Verlag, Berlin, 2000b, **15**, pp. 139–152.

Estabrooks, A.; Jo, T. y Japkowicz, N.: «A multiple resampling method for learning from imbalanced data-sets». *Computational Intelligence*, 2004, **20(1)**, pp. 18–36.

Fahlman, S. y Lebiere, C.: «The Cascade-Correlation Learning Architecture». En: *Proceedings in Advances in Neural Information Processing Systems 2*, pp. 524–532. Morgan Kaufmann, 1990.

Fan, R. E.; Chen, P. H. y Lin, C. J.: «Working set selection using the second order information for training SVM». *Journal of Machine Learning Research*, 2005, **6**, pp. 1889–1918.

Fayyad, U.; Piatetsky-Shapiro, G. y Smyth, P.: «From data mining to knowledge discovery: An overview». *Advances in Knowledge Discovery and Data Mining*, 1996a, pp. 1–34.

Fayyad, U.; Piatetsky-Shapiro, G. y Smyth, P.: «The KDD process for extracting useful knowledge from volumes of data». *Communications of the ACM*, 1996b, **39(11)**, pp. 27–34.

Fernández, A.; García, S.; del Jesus, M.J. y Herrera, F.: «A Study of the Behaviour of Linguistic Fuzzy Rule Based Classification Systems in the Framework of Imbalanced Data Sets». *Fuzzy Sets and Systems*, 2008, **159(18)**, pp. 2378–2398.

Ferreira, P.M.; Ruano, A.E. y Fonseca, C.M.: «Genetic assisted selection of RBF model structures for greenhouse inside air temperature prediction». *IEEE Control Applications*, 2003, **1**, pp. 576–581.

Fogel, D. B; Fogel, L. J. y Porto, V. W.: «Evolving neural networks». *Biological Cybernetics*, 1990, **63**, pp. 487–493.

Fogel, L. J.: «Autonomous automata». *Industrial Research*, 1962, **4**, pp. 14–19.

Frances, P.H. y Dijk, D. Van: *Non-linear time series models in empirical finance*. Cambridge University Press, 2000.

Frean, M.: «The upstart algotihm: a method for constructing and training feedforward neural networks». *Neural Computation*, 1990, **2**, pp. 198–209.

Freitas, A.: *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer, 2002.

Fu, X. y Wang, L.: «A GA-based novel RBF classifier with class-dependent features». En: *Proceedings of the Evolutionary Computation*, volumen 2, pp. 1964–1969, 2002.

Fu, X. y Wang, L.: «Data dimensionality reduction with application to simplifying RBF network structure and improving classification performance». *IEEE Transactions on System, Man, and Cybernetics, B*, 2003, **33(3)**, pp. 399–409.

García, S.; Fernández, A.; Luengo, J. y Herrera, F.: «A Study of Statistical Techniques and Performance Measures for Genetics–Based Machine Learning: Accuracy and Interpretability». *Soft Computing*, 2009a, **13(10)**, pp. 959–977.

García, S. y Herrera, F.: «An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets" for all Pairwise Comparisons». *Journal of Machine Learning Research*, 2008, **9**, pp. 2677–2694.

García, S.; Molina, D.; Lozano, M. y Herrera, F.: «A study on the use of non-parametric test for analyzinh the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization». *Journal of Heuristics*, 2009b, **15**, pp. 617–644.

García, V.; Mollineda, R.A. y Sánchez, J. S.: «On the k-NN performance in a challenging scenario of imbalance and overlapping». *Pattern Analysis Applications*, 2008, **11(3–4)**, pp. 269–280.

Ghosh, A. y Jain, L.: *Evolutiorary Computation in Data Mining*. Studies in Fuzziness and Soft Computing. Springer, 2005.

Ghost, J.; Deuser, L. y Beck, S.: «A neural network based hybrid system for detection, characterization and classification of short-duration oceanic signals». *IEEE Journal of Ocean Enginering*, 1992, **17(4)**, pp. 351–363.

Giordana, A.; Saitta, L. y Zini, F.: «Learning disjunctive concepts by means of genetic algorthms». En: *Proceedings of the 11 International Conference on Machine Learning*, pp. 96–104, 1994.

Goldberg, D.: *Genetic Algorithms*. Addison-Wesley, Reading, MA, 1978.

Goldberg, D. y Richardson, J.: «Genetic algorithms with sharing for multimodal function optimization». En: *Proceedings of the Second International Conference on Genetic Algorithms*, pp. 41–49, 1987.

Golub, G. y Van Loan, C.: *Matrix computations*. Hopkins University Press. 3rd edition, 1996.

González, J.; Rojas, I.; Ortega, J.; Pomares, H.; Fernández, J. y Díaz, A.F.: «Multiobjective evolutionary optimization of the size, shape,

and position parameters of radial basis function networks for function approximation». *IEEE Transactions on Neural Networks*, 2003, **14(6)**, pp. 1478–1495.

González, J.; Rojas, I.; Ortega, J. y Prieto, A.: «A new clustering technique for function approximation». *IEEE Transactions on Neural Networks*, 2002, **13(1)**, pp. 132–142.

Grönroos, Marko: *Evolutionary Design of Neural Networks.* Tesina o Proyecto, Computer Science, Department of Mathematical Sciences, University of Turku, Finland, 1998.

Grossberg, S.: «Adaptative pattern classification and universal recording, I: Pararell development and coding of neural feature detectors». *Biological Cybernetics*, 1976a, **23**, pp. 121–134.

Grossberg, S.: «Adaptative pattern classification and universal recording, II: Feedback, expectation, olfaction and illusions». *Biological Cybernetics*, 1976b, **23**, pp. 187–202.

Gruau, F.: «Genetic synthesis of boolean neural networks with a cell rewriting developmental process». En: *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks*, pp. 55–74, 1992.

Guillén, A.; Pomares, H.; Rojas, I.; González, J.; Herrera, L.J.; Rojas, F. y Valenzuela, O.: «Output value-based initialization for radial basis function neural networks». *Neural Processing Letters. In Press*, 2007. doi: 10.1007/s11063-007-9039-8.

HAN, J. y KAMBER, M.: *Data Mining: Concepts and Techniques*. Morgan Kaufmann Publishers, 2000.

HARP, S.; SAMAD, S. y GUHA, A.: «Designing application-specific neural networks using the genetic algorithm». *Advances in Neural Information Processing Systems 2*, 1990, pp. 447–454.

HARP, S.; SAMAD, T. y GUHA, A.: «Towards the genetic synthesis of neural networks». En: *Proceedings of the Third International Conference on Genetic Algorithms and Their Applications*, pp. 360–369, 1989.

HARPHAM, C.; DAWSON, C.W. y BROWN, M.R.: «A review of genetic algorithms applied to training radial basis function networks». *Neural Computing and Applications*, 2004, **13**, pp. 193–201.

HART, P. E.: «The condensed nearest neighbor rule». *IEEE Transactions on Information Theory*, 1968, **14**, pp. 515–516.

HARTIGAN, J. A.: *Clustering Algorithms*. Willey, New York, 1975.

HAYKIN, S.: *Neural Networks: A Comprehensive Foundation, 2nd Edition*. Prentice Hall, 1999.

HEBB, D.: *Organization of Behavior*. John Wiley & Sons, 1949.

HERNÁNDEZ, J.; RAMÍREZ, M.J. y FERRI, C.: *Introducción a la Minería de Datos*. Pearson, 2004.

HILLIS, D. W.: «Co-evolving parasites improve simulated evolution as an optimization procedure». *Artificial Life II, SFI Studies in the Sciences of Complexity*, 1991, **10**, pp. 313–324.

HINTON, G.; ACKLEY, D. y SEJNOWSKI, T.: «Boltzmann machines: Constraint satisfaction networks that learn». *Informe técnico*, Carnegie-Mellon University, 1984.
`http://www.computerhistory.org/collections/accession/102618169`

HOBBS, B.F.; HELMAN, U.; JITPRAPAIKULSARN, S.; KONDA, S. y MARATUKULAM, D.: «Artificial neural networks for short-term energy forecasting: Accuracy and economic value». *Neurocomputing*, 1998, **23(1–3)**, pp. 71–84.

HOLCOMB, T. y MORARI, M.: «Local training for radial basis function networks: towards solving the hidden unit problem». En: *Proceedings of the American Control Conference, Boston*, pp. 2331–2336, 1991.

HOLLAND, J.H.: *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, 1975.

HOLM, S.: «A simple sequentially rejective multiple test procedure». *Scandinavian Journal of Statistics*, 1979, **6**, pp. 65–70.

HOPFIELD, J.: «Neural Networks and phisycal systems with emergent collective computational abilities». En: *Proceedings of the National Academy of Science*, volumen 81, pp. 3088–3092, 1982.

HOWARD, L. y D'ANGELO, D.: «The GA-P: A Genetic Algorithm and Genetic Programming Hybrid». *IEEE Intelligent Systems*, 1995, **10(3)**, pp. 11–15.

HUANG, S.N.; TAN, K.K. y LEE, T.H.: «Adaptive neural network algorithm for control design of rigid-link electrically driven robots». *Neurocomputing*, 2008, **71(4-6)**, pp. 885–894.

Huang, Y. M.; Hung, C. M. y Jiau, H. C.: «Evaluation of Neural Networks and Data Mining Methods on a Credit Assessment Task for Class Imbalance Problem». *Nonlinear Analysis: Real World Applications*, 2006, **7(4)**, pp. 720–747.

Husbands, P. y Mill, F.: «Simulated co-evolution as the mechasnism for emergent planing abd scheduling». En: *Proceedings of the Fourth International Conference on Genetic Algotithms*, pp. 264–270, 1991.

Isasi, P. y Galván, I. M.: *Redes de Neuronas Artificiales. Un Enfoque Práctico*. Pearson, 2004.

Jang, J.R.: «ANFIS: Adaptative-Network-based Fuzzy Inference System». *IEEE Transactions Systems, Man and Cybernetics*, 1993, **23(3)**, pp. 665–685.

Jang, J.S.R. y Sun, C.T.: «Functional equivalence between radial basis functions and fuzzy inference systems». *IEEE Transactions on Neural Networks*, 1993, **4**, pp. 156–158.

Japkowicz, N. y Stephen, S.: «The class imbalance problem: A systematic study». *Intelligent Data Analalisys*, 2002, **6(5)**, pp. 429–449.

Jiang, N.; Zhao, Z.Y. y Ren, L.Q.: «Design of structural modular neural networks with genetic algorithm». *Advances in Engineering Software*, 2003, **1**, pp. 17–24.

Jin, Y. y Sendhoff, B.: «Extracting interpretable fuzzy rules from RBF networks». *Neural Processing Letters*, 2003, **17(2)**, pp. 149–164.

Jo, T. y Japkowicz, N.: «Class Imbalances versus Small Disjuncts». *SIGKDD Explorations*, 2004, **6(1)**, pp. 40–49.

Jong, K. De: *An analysis of the behaviour of a class of genetic adaptative system.* Tesis doctoral, University of Michigan, 1975.

Kadirkamanathan, V.: «A function estimation approach to sequential learning with neural networks». *Neural Computation*, 1993, **5**, pp. 954–975.

Karayiannis, N.B. y Mi, G.W.: «Growing radial basis neural networks: Merging supervised and unsupervised learning with network growth techniques». *IEEE Transactions on Neural Networks*, 1997, **8(6)**, pp. 1492–1506.

Khashei, M.; Reza Hejazi, S. y Bijari, M.: «A new hybrid artificial neural networks and fuzzy regression model for time series forecasting». En: *Proceedings of the Fuzzy Sets and Systems*, volumen 159, pp. 769–786, 2008.

Kilic, Kemal; Uncu, Özge y Türksen, I. Burhan: «Comparison of different strategies of utilizing fuzzy clustering in structure identification». *Information Sciences*, 2007, **177(23)**, pp. 5153–5162.

Kim, H.; Jung, S.; Kim, T. y Park, K.: «Fast learning method for backpropagation neural network by ecolutionary adaptation of learning rates». *Neurocomputing*, 1996, **11(1)**, pp. 101–106.

Kitano, K.: «Designing neural networks by genetic algortihms using graph generation systems». *Complex Systems*, 1990, **4**, pp. 461–476.

Klir, G. y Yuan, B.: *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prencice-Hall, 1995.

KOHAVI, R.: «A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection». En: *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 1137–1143, 1995.

KOHONEN, T.: «Self-organized formation of topologically correct feature maps». *Biological Cybernetics*, 1982, **43(1)**, pp. 59–69.

KOSKO, B.: «Bidirectional associative memories». *IEEE Transactions on Systems, Man, and Cybernetics*, 1988, **18**, pp. 42–60.

KOZA, R. J.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, 1992.

KOZA, R. J.: *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, 1994.

KUBAT, M. y MATWIN, S.: «Addressing the Curse of Imbalanced Training Sets: One-Sided Selection». En: *Proceedings of the Fourteenth International Conference on Machine Learning*, pp. 179–186. Morgan Kaufmann, 1997.

LACERDA, E.; CARVALHO, A.; BRAGA, A. y LUDERMIR, T.: «Evolutionary Radial Functions for Credit Assessment». *Applied Intelligence*, 2005, **22**, pp. 167–181.

LARSEN, P.: «Industrial applications of fuzzy logic control». *International Journal of Man Machine Studies*, 1980, **12**, pp. 3–10.

LAURIKKALA, J.: «Improving Identification of Difficult Small Classes by Balancing Class Distribution». En: *Proceedings of the 8th Conference on AI in Medicine in Europe (AIME'01)*, pp. 63–66. Springer-Verlag, 2001.

Lee, S. y Kil, R.M.: «A Gaussian potential function network with hierarchically seft-organising learning». *Neural Networks*, 1991, **4**, pp. 207–224.

Leung, H.; Dubash, N. y Xie, N.: «Detection of small objects in clutter using a GA-RBF neural network». *IEEE Transactions on Aerospace and Electronic Systems*, 2002, **38(1)**, pp. 98–118.

Li, B.; Peng, J.; Chen, Y. y Jin, Y.: «Classifying unbalanced pattern groups by training neural network». En: *Proceedings of the Third International Symposium on Neural Networks (ISNN'06)*, volumen 3972, pp. 8–13, 2006.

Li, M.; Tian, J. y Chen, F.: «Improving multiclass pattern recognition with a co-evolutionary RBFNN». *Pattern Recognition Letters*, 2008, **29(4)**, pp. 392–406.

Lipmann, R.: «An introduction to computing with neural nets». *IEEE Transaction on Acoustics, Speech, and Signal Processing*, 1987, **2(4)**, pp. 4–22.

Liu, J.; McKenna, T.M.; Gribok, A.; Beidleman, B.A.; Tharion, W.J. y Reifman, J.: «A fuzzy logic algorithm to assign confidence levels to heart and respiratory rate time series». *Physiological Measurement*, 2008, **29(1)**, pp. 81–94.

MacQueen, J. B.: «Some methods for classification and analysis of multivariate observations». En: *Proceedings of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volumen 1, pp. 281–297, 1967.

Maglogiannis, I.; Sarimveis, H.; Kiranoudis, C.T.; Chatziioannou, A.A.; Oikonomou, N. y Aidinis, V.: «Radial basis function neural netwroks classification for the recognition of idiopathic pulmonary fibrosis in microscopic images». *IEEE Transactions on Information Technology in Biomedicine*, 2008, **12(1)**, pp. 42–54.

Mahfoud, S. W.: «Crowding and preselection revised». En: *Proceedings of the Parallel Problem Solving from Nature*, pp. 27–36, 1992.

Maimon, O. y Rokach, L.: *The Data Mining and Knowledge Discovery Handbook*. Springer, 2005.

Mamdani, E.: «Applications of fuzzy algorithms for symple dinamycs plant». En: *Proceedings of the IEEE*, volumen 121, pp. 1585–1588, 1974.

Mamdani, E. y Assilian, S.: «An experiment in linguistic synthesis with a fuzzy logic controller». *International Journal of Man-Machine Studies*, 1975, **7(1)**, pp. 1–13.

Marcos, J.V.; Hornero, R.; Álvarez, D.; Del Campo, F.; López, M. y Zamarrón, C.: «Radial basis function classifiers to help in the diagnosis of the obstructive sleep apnoea syndrome from nocturnal oximetry». *Medical and Biological Engineering and Computing*, 2008, **46**, pp. 323–332.

Mazurowski, M.A.; Habas, P.A.; Zurada, J.M.; Lo, J.Y.; Baker, J.A. y Tourassi, G.D.: «Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance». *Neural Networks*, 2008, **21(2-3)**, pp. 427–436.

McCulloch, J. y Pitts, W.: «A logical calculus of the ideas immanent in nervous activity». *Bulletin of Mathematical Biophysics*, 1943, **7**, pp. 115–133.

Merelo, J.; Patón, M.; Canas, A.; Prieto, A. y Morán, F.: «Genetic optimization of a multilayer neural network for cluster classification tasks». *Neural Network World*, 1993, **3**, pp. 175–186.

Miller, G. F.; Todd, P. M. y Hegde, S. U.: «Designing neural networks using genetic algorithms». En: *Proceedings of the Third International Conference on Genetic Algorithms*, pp. 379–384, 1989.

Minsky, M.: *Neural-analog networks and the brain model problem.* Tesis doctoral, Princenton University, 1954.

Minsky, M. y Papert, S.: *Perceptron, An introduction to computational geometry.* MIT press, 1969.

Moechtar, M.; Farag, A.S.; Hu, L. y Cheng, T.C.: «Combined genetic algorithms and neural network approach for power system transient stability evaluation». *European Transactions on Electrical Power*, 1999, **9(2)**, pp. 115–122.

Moller, F.: «A scaled conjugate gradient algorithm for fast supervised learning». *Neural Networks*, 1993, **6**, pp. 525–533.

Montana, D. y Davis, L.: «Training feedforward neural networks using genetic algorithms». En: *Prooceedings of the 11th International Conference on Artificial Intelligence*, pp. 762–767, 1989.

Moody, J. y Darken, C.J.: «Fast learning in networks of locally-tuned processing units». *Neural Computation*, 1989, **1(2)**, pp. 281–294.

Moriarty, D. y Miikkulainen, R.: «Forming neural networks through efficient and adaptive coevolution». *Evolutionary Computation*, 1997, **5(4)**, pp. 373–399.

Murhphey, Y.L. y Guo, H.: «Neural learning from umbalanced data». *Applied Intelligence*, 2004, **21**, pp. 117–128.

Musavi, M.T.; Ahmed, W.; Chan, K.H.; Faris, K.B. y Hummels, D.M.: «On the training of radial basis function classifiers». *Neural Networks*, 1992, **5**, pp. 595–603.

Neruda, R. y Kudová, P.: «Learning methods for radial basis function networks». *Future Generation Computer Systems*, 2005, **21(7)**, pp. 1131–1142.

Novak, P. K.; Lavrac, N. y Webb, G. I.: «Supervised descriptive rule discovery: a unifying survey of contrast set, emerging pattern and subgroup mining». *Journal of Machine Learning Research*, 2009, **10**, pp. 377–403.

Orr, M.: «Regularized center recruitment in Radial Basis Function Networks». *Informe técnico 59*, Center for cognitive science. University of Edinburgh, 1993.

Orr, M.: «Regularization on the selection of radial basis function centers». *Neural Computation*, 1995, **7**, pp. 606–623.

Orr, M.: «Introduction to radial basis function networks». *Informe técnico*, Center for cognitive science. University of Edinburgh, 1996.

Orriols-Puig, A. y Bernadó-Mansilla, E.: «Evolutionary rule-based

systems for imbalanced data-sets». *Soft Computing*, 2009, **13(3)**, pp. 213–225.

Padmaja, T.M.; Dhulipalla, N.; Krishna, P.R.; Bapi, R.S. y Laha, A.: «An unbalanced data classification model using hybrid sampling technique for fraud detection». En: *Second International Conference on Pattern Recognition and Machine Intelligence (PReMI'07)*, volumen 4315, pp. 341–438, 2007.

Padmaja, T.M.; Krishna, P.R. y Bapi, R.S.: «Majority filter-based minority prediction (MFMP): An approach for unbalanced datasets». En: *Proceedings of the IEEE Region 10 Annual International Conference (TENCON)*, 4766705, pp. 1–6, 2008.

Paredis, J.: «Coevolutionary computation». *Artificial Life*, 1995, **2**, pp. 355–375.

Park, J. y Sandberg, I.: «Universal approximation using radial-basis function networks». *Neural Compututation*, 1991, **3(2)**, pp. 246–257.

Park, J. y Sandberg, I.: «Universal approximation and radial basis function network». *Neural Compututation*, 1993, **5(2)**, pp. 305–316.

Peña, D.: *Análisis de series temporales.* Alianza Editorial, 2005.

Pedrycz, W.: «Conditional fuzzy C-means». *Pattern Recognition Letters*, 1996, **17**, pp. 625–632.

Pedrycz, W.: «Conditional fuzzy clustering in the design of radial basis function neural networks». *IEEE Transactions on Neural Networks*, 1998, **9(4)**, pp. 601–612.

Peng, J.X.; k. Li y Huang, D.S.: «A hybrid forward algorithm for RBF neural network construction». *IEEE Transactions on Neural Networks*, 2006, **17(6)**, pp. 1439–1451.

Peng, X. y King, I.: «Robust BMPM training based on second-order cone programming and its application in medical diagnosis». *Neural Networks*, 2008, **21(2–3)**, pp. 450–457.

Pino, R.; Parreno, J.; Gomez, A. y Priore, P.: «Forecasting next-day price of electricity in the Spanish energy market using artificial neural networks». *Engineering Applications of Artificial Intelligence*, 2008, **21(1)**, pp. 53–62.

Plat, J.: «A resource allocating network for function interpolation». *Neural Computation*, 1991, **3(2)**, pp. 213–225.

Poggio, T. y Girosi, F.: «Networks for approximation and learning». En: *Proceedings of the IEEE*, volumen 78, pp. 1481–1497, 1990.

Potter, M. y De Jong, K.: «Cooperative Coevolution: an architecture for evolving coadapted subcomponents». *Evolutionary Compututation*, 2000, **8(1)**, pp. 1–29.

Powell, M.: «Radial basis functions for multivariable interpolation: A review». *In IMA. Conf. on Algorithms for the approximation of functions and data*, 1985, pp. 143–167.

Procyk, R. y Mamdani, E.: «A linguistic self-organizing process controller». *Automatica*, 1979, **15(1)**, pp. 15–30.

Provost, Foster y Fawcett, Tom: «Robust classification for imprecise environments». *Machine Learning*, 2001, **42(3)**, pp. 203–231.

Quilan, J.R.: *C4.5: Programs for Machine Learning.* Morgan Kauffman Publishers, San Mateo, CA, 1993.

Rechenberg, I.: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Tesis doctoral, Technical University of Berling, 1971.

Rivas, V.; Merelo, J.J.; Castillo, P.; Arenas, M.G y Castellano, J.G.: «Evolving RBF neural networks for time-series forecasting with EvRBF». *Information Science*, 2004, **165**, pp. 207–220.

Rivera, A.J.; Rojas, I.; Ortega, J. y del Jesus, M.J.: «A new hybrid methodology for cooperative-coevolutionary optimization of radial basis function networks». *Soft Computing*, 2007, **11(7)**, pp. 655–668.

Rojas, I.; Pomares, H.; Bernier, J.; Ortega, J.; Pino, B.; Pelayo, F. y Prieto, A.: «Time series analysis using normalized PG-RBF network with regression weigths». *Neurocomputing*, 2002, **42(1)**, pp. 267–285.

Rojas, I.; Valenzuela, O. y Prieto, A.: «Statistical Analysis of the Main Parameters in the Definition of Radial Basis Function Networks». *LNCS*, 1997, **1240**, pp. 882–891.

Rojas, R. y Feldman, J.: *Neural Networks: A Systematic Introduction.* Springer, 1996.

Rosenblatt, F.: «The perceptron: A perceiving and recognizing automation». *Informe técnico 85–460–1*, Cornell Aeronautical Laboratory, 1957.

Rosenblatt, F.: *Principles of Neurodynamics.* Spartan Books, Washington, 1962.

Rosin, C. D y Belew, R. K.: «New methods for competitive coevolution». *Evolutionary Computation*, 1997, **5**, pp. 1–29.

Rosipal, R.; Koska, M. y Frakaš, I.: «Prediction of chaotic time series with a resource allocating RBF networks». *Neural Processing Letters*, 1998, **7**, pp. 185–197.

Rui, L. y Minghu, J.: «Chinese text classification based on the BVB model». En: *Proceedings of the 4th International Conference on Semantics, Knowledge, and Grid (SKG'08)*, pp. 376–379, 2008.

Rumelhart, D.; McClelland, J. y the PDP Research Group: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*. MIT Press: Cambrige, Mass, 1986.

Runkler, T. A. y Bezdek, J. C.: «Alternating cluster estimation: A new tool for clustering and function approximation». *IEEE Tansaction on Fuzzy System*, 1999, **7(4)**, pp. 377–393.

Ruspini, E. H.: «A new approach to clustering». *Information and Control*, 1969, **15**, pp. 22–32.

Russo, M. y Patanè, G.: «Improving the LBG Algorithm». *LNCS*, 1999, **1606**, pp. 624–630.

Salmerón, M.; Ortega, J.; Puntonet, C. y Prieto, A.: «Improved RAN sequiential prediction using orthogonal techniques». *Neurocomputing*, 2001, **41(1–4)**, pp. 153–172.

Sánchez, L. y Couso, I.: «Fuzzy Random Variables-Based Modeling with GA-P Algorithms». *In: B. Bouchon, R.R. Yager, L. Zadeh (Eds.) Information, Uncertainty and Fusion*, 2000, pp. 245–256.

Sanchez, V.D.: «A searching for a solution to the automatic RBF network design problem». *Neurocomputing*, 2002, **42**, pp. 147–170.

Schwefel, H.: *Evolutionsstrategie und numerische Optimierung.* Tesis doctoral, Technical University of Berling, 1975.

Sejnowski, T. y Hinton, G.: «Separating figure from ground with a Boltzmann machine». *Vision, Brain and Cooperative Computation*, 1986, pp. 703–724.

Sergeev, S.A.; Mahotilo, K.V.; Voronovsky, G.K. y Petrashev, S.N.: «Genetic algorithm for training dynamical object emulator based on RBF neural network». *International Journal of Applied Electromagnetics and Mechanics*, 1998, **9(1)**, pp. 65–74.

Sheskin, D.: *Handbook of parametric and nonparametric statistical procedures.* Chapman & Hall/CRC, 2nd[a] edición, 2006.

Sheskin, D.J.: *Handbook of Parametric and Nonparametric Statistical Procedures.* CRC Press, 2000.

Sheta, A.F. y De Jong, K.: «Time-series forecasting using GA-tuned radial basis functions». *Information Sciencie*, 2001, **133**, pp. 221–228.

Siddiqui, A.M.; Masood, A. y Saleem, M.: «A locally constrained radial basis function for registration and warping of images». *Pattern Recognition Letters*, 2009, **30(4)**, pp. 377–390.

Stanfill, C. y Waltz, D.: «Towards memory-based reasoning, Commnun». *ACM*, 1986, **29(12)**, pp. 1213–1228.

Sugeno, M. y Kang, G.: «Structure identification of fuzzy model». *Fuzzy Sets and System*, 1988, **28**, pp. 15–33.

Sumathi, S.; Sivanandam, S.N. y Ravindran, R.: «Design of a soft computing hybrid model classifier for data mining applications». *International Journal of Engineering intelligent systems for electrical engineering and communications*, 2001, **9(1)**, pp. 33–56.

Sun, Yanmin; Kamel, Mohamed S.; Wong, Andrew K.C. y Wang, Yang: «Cost-sensitive boosting for classification of imbalanced data». *Pattern Recognition*, 2007, **40**, pp. 3358–3378.

Sun, Y.F.; Liang, Y.C.; Zhang, W.L.; Lee, H.P.; Lin, W.Z. y Cao, L.J.: «Optimal partition algorithm of the RBF neural network and its application to financial time series forecasting». *Neural Computing and Applications*, 2005, **14(1)**, pp. 36–44.

Sundararajan, N.; Saratchandran, P. y Yingwei, L.: *Radial basis function neural network with sequential learning: MRAN and its application*. World Scientifics, New York, 1999.

Suresh, S.; Sundararajan, Narasimhan y Saratchandran, Paramasivan: «Risk-sensitive loss functions for sparse multi-category classification problems». *Information Sciences*, 2008, **178(12)**, pp. 2621–2638.

Sutton, R. S.: «Two problems with backpropagation and other steepest-descent learnig procedures for networks». En: *Proceedings of thr Eighth Annual Conference of the Cognitive Science Society*, pp. 823–831, 1986.

Tan, P.N.; Steinbach, M. y Kumar, V.: *Introduction to Data Mining.* Pearson Education, 2006.

Teixeira, C.A.; Ruano, M.G.; Ruano, A.E. y Pereira, W.C.A.: «A soft-computing methodology for noninvasive time-spatial temperature estimation». *IEEE Transactions on Biomedical Engineering*, 2008, **55(2)**, pp. 572–580.

Tettamanzi, A. y Tomassini, M.: *Soft Computing. Integrating Evolutionary, Neural, and Fuzzy Systems.* Springer, 2001.

Tomek, I.: «Two Modifications of CNN». *IEEE Transactions on Systems Man and Cybernetics*, 1976, **6(11)**, pp. 769–772.

Topchy, A.; Lebedko, O.; Miagkikh, V. y Kasabov, N.: «Adaptive training of radial basis function networks based on co-operative evolution and evolutionary programming». En: *Proceedings of the International Conference Neural Information Processing (ICONIP)*, pp. 253–258, 1997.

Tsukamoto, Y.: *Advances in Fuzzy Set Theory and Applications.* capítulo An approach to fuzzy reasoning method, pp. 137–149. Amsterdam: North-Holland, 1979.

Ture, M. y Kurt, I.: «Comparison of four different time series methods to forecast hepatitis A virus infection». *Expert Systems with Applications*, 2006, **31(1)**, pp. 41–46.

Uriel, E. y Peiró, A.: *Introducción al análisis de series temporales.* Alfa Centauro, S.A., 2000.

Vesin, J.M. y Gruter, R.: «Model selection using a simplex reproduction genetic algorithm». *Sigal Processing*, 1999, **78**, pp. 321–327.

Weiss, Gary y Provost, Foster: «Learning when training data are costly: The effect of class distribution on tree induction». *Journal of Artificial Intelligence Research*, 2003, **19**, pp. 315–354.

Werbos, P.: *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Tesis doctoral, Harvard University, 1974.

Whitehead, B. y Choate, T.: «Cooperative-competitive genetic evolution of Radial Basis Function centers and widths for time series prediction». *IEEE Transactions on Neural Networks*, 1996, **7(4)**, pp. 869–880.

Whitley, D.; Starkweather, T. y Bogart, C.: «Genetic algorithms and neural networks: optimizing connections and connectivity». *Parallel Computing*, 1990, **14(3)**, pp. 347–361.

Widrow, B. y Hoff, M: «Adaptive switching circuits». En: *Proceedings of the IRE WESCON Convention*, volumen 4, pp. 96–104, 1960.

Widrow, B. y Lehr, M.A.: «30 Years of adaptive neural networks: perceptron, madaline and backpropagation». En: *Proceedings of the IEEE*, volumen 78, pp. 1415–1442, 1990.

Wilcoxon, F.: «Individual comparisons by ranking methods». *Biometrics*, 1945, **1**, pp. 80–83.

Wilson, D. L.: «Asymptotic Properties of Nearest Neighbor Rules Using Edited Data». *IEEE Transactions on Systems Man and Cybernetics*, 1972, **2(3)**, pp. 408–421.

Wilson, D.R. y Martinez, T.R.: «Improved heterogeneous distance functions». *Journal on Artificial Intelligence Research*, 1997, **6(1)**, pp. 1–34.

WITTEN, I.H. y E.FRANK: *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan-Kaufmann, 2005.

WOLD, H.: *A Study in the Analysis of Stationary Time Series. (First edition 1938)*. Almquist and Wicksell, 1954.

WU, G. y CHANG, E.Y.: «KBA: Kernel boundary alignment considering imbalanced data distribution». *IEEE Transactions on Knowledge Data Engineering*, 2005, **17(6)**, pp. 786–795.

WU, S. y CHOW, T.W.S.: «Induction Machine Fault Detection Using SOM-Based RBF Neural Networks». *IEEE Transactions on Industrial Electronics*, 2004, **51(1)**, pp. 183–194.

XU, L.; CHOW, M.Y. y TAYLOR, L.S.: «Power distribution fault cause identification with imbalanced data using the data mining-based fuzzy classification e-algorithm». *IEEE Transactions on Power Systems*, 2007, **22(1)**, pp. 164–171.

XUE, Y. y WATTON, J.: «Dynamics modelling of fluid power systems applying a global error descent algorithm to a selforganising radial basis function network». *Mechatronics*, 1998, **8(7)**, pp. 727–745.

YANG, QIANG y WU, XINDONG: «10 Challenging Problems in Data Mining Research». *International Journal of Information Technology and Decision Making*, 2006, **5(4)**, pp. 597–604.

YAO, X.: «Evolving artificial neural networks». En: *Proceedings of the IEEE*, volumen 87, pp. 1423–1447, 1999.

YEN, G.G.: «Multi-Objective evolutionary algorithm for radial basis

function neural network design». *Studies in Computational Intelligence*, 2006, **16**, pp. 221–239.

YINGWEI, L.; SUNDARAJAN, N. y SARATCHANDRAN, P.: «A sequential learning scheme for function approximation using minimal radial basis function neural network». *Neural Computation*, 1997, **9**, pp. 361–478.

YU, T. y WILKINSON, D.: «A co-evolutionary fuzzy system for reservoir well logs interpretation». *Evolutionary computation in practice*, 2008, pp. 199–218.

YULE, G. U.: «On the time-correlation problems with special reference to the variate difference correlation method». *Journal of the Royal Statistical Society*, 1921, **84**, pp. 497–526.

YULE, G. U.: «Why do we sometimes get nonsense correlations between time series?: a study in sampling and the nature of time series». *Journal of the Royal Statistical Society*, 1926, **80**, pp. 1–64.

YULE, G. U.: «On a method for investigating periodicities in disturbed series with special reference to Wolfer's sunspot numbers». *Philosophical Transactions*, 1927, **226**, pp. 267–298.

ZADEH, L.: «Fuzzy sets». *Information Control*, 1965, **8**, pp. 338–353.

ZADEH, L.: «Outline of a new approah to the analysis complex system and decision processes». *IEEE Transactions on System, Man and Cybernetic*, 1973, **3**, pp. 28–44.

ZADEH, L.: «Fuzzy logic and soft computing: issues, contentions and perspectives». En: *Proceedings of the 3rd. International conference on fuzzy logic, neural nets and soft computing (Iizuka'94)*, pp. 1–2, 1994.

ZAR, J.H.: *Biostatistical Analysis*. Prencice Hall, Upper Saddle River, New Jersey, 1999.

ZHANG, C.; JIANG, J. y KAMEL, M.: «Intrusion detection using hierarchical neural networks». *Pattern Recognition Letters*, 2005, **26(6)**, pp. 779–791.

ZHANG, P. G.: *The Data Mining and Knowledge Discovery Handbook*. capítulo Neural Networks, pp. 487–516. Springer, 2005.

ZHAO, Z. Q.: «A novel modular neural network for imbalanced classification problems». *Pattern Recognition Letters*, 2009, **30(9)**, pp. 783–788.

ZHOU, Z. H. y LIU, X. Y.: «Training cost-sensitive neural networks with methods addressing the class imbalance problem». *IEEE Transactions on Knowledge Data Engineering*, 2006, **18(1)**, pp. 63–77.