Proyecto Fin de Carrera

Sistema de Minería de Datos
basado en Algoritmos Genéticos Multiobjetivo
para Reducción de la Dimensionalidad

Autor:
Manuel Chica Serrano

Tutores:
María José del Jesus Díaz
José Joaquín Aguilera García

 Universidad de Jaén Septiembre de 2006

Índice general

Índice de figuras

Índice de cuadros

Introducción y objetivos del Proyecto

La Minería de Datos es un área de la Inteligencia Artificial que se encarga de obtener conocimiento relevante y útil a partir de grandes cantidades de información albergadas en bases de datos o almacenes de datos empresariales. Dicho conocimiento es desconocido a priori y se descubre a partir del análisis de esa información plana.

Existe un tipo de Minería de Datos llamada Minería de Datos Predictiva que realiza una predicción sobre valores de datos nuevos a partir de resultados conocidos o pasados. Y dentro de esta Minería de Datos Predictiva es en dónde se encuentra la tarea de **clasificación**.

La **clasificación** permite *adivinar* la clase a la que pertenece un ejemplo o instancia de una base de datos teniendo en cuenta la información que tenemos sobre otros ejemplos de la misma. Para aprender cualquier sistema de clasificación debemos conocer datos históricos o pasados, es por ello por lo que se enmarca dentro del aprendizaje supervisado.

Es este el **objetivo principal del *Proyecto Fin de Carrera*: diseñar una herramienta que desarrolle un sistema de clasificación** que sea realmente preciso y bueno. Y además, que cuando nos enfrentemos a problemas complejos y con una alta dimensionalidad responda mejor que la mayoría de otros sistemas. Y para que nuestro sistema de clasificación tenga éxito en su labor con problemas que contienen cantidades ingentes de información nos vemos abocados a aplicar una tarea de preprocesamiento previa: **la reducción de la dimensionalidad**.

La tarea de reducción en un problema grande y complejo nos ayuda a:

- **Reducir la complejidad del proceso posterior de Minería**, en nuestro caso, la obtención del sistema de clasificación que permita clasificar nuevos ejemplos. No serán los mismos los resultados obtenidos a partir de un gran conjunto de datos, con la mayoría de éstos irrelevantes y redudantes, que los obtenidos ante datos filtrados y sintetizados.

- **Reducir la tendencia al sobreaprendizaje**. Esto suele ocurrir cuando el algoritmo de Minería de Datos intenta adaptarse y aprender sobre bases de datos grandes y voluminosas. Al tener datos que no siguen la tendencia general del problema el sistema se ajusta demasiado a los datos con los que aprende, viéndose incapaz de clasificar igual de bien con datos nuevos y distintos a los que ha aprendido.

Existen primordialmente dos vías a seguir en cuanto a la reducción de la dimensionalidad se refiere:

- **Selección de atributos o características:** en dónde se eliminan ciertas características de los datos que no tienen influencia en la clasificación posterior, o que son redudantes con otras.
- **Selección de instancias o ejemplos:** puede que existan muchos ejemplos parecidos en una misma base de datos, por lo que podremos quedarnos sólo con los más representativos.

Dada su mayor importancia e impacto en el sistema que se pretende construir nos centraremos en la primera tarea, en la **selección de características**.

La mayoría de las propuestas de selección de características existentes en la bibliografía especializada reducen la dimensionalidad de la base de datos original obteniendo un único conjunto de variables. A partir de este conjunto de características se diseña el sistema de clasificación, que sólo tendrá en cuenta estas variables.

Es de suponer que gran parte del éxito del sistema final sea debida a la bondad del conjunto de características sobre el que se trabaje. Por muy bueno que sea el método de Minería de Datos, si trabaja sobre un conjunto de características poco convenientes, su resultado no será el esperado.

Como nuestra propuesta es la de diseñar un sistema de clasificación para problemas complejos que clasifique nuevas instancias independientemente de las características que se hayan seleccionado, nos plantearemos los siguientes dos subobjetivos:

- **Conseguir reducir las características originales del problema obteniendo no un único conjunto de las mismas, sino los mejores subconjuntos.**

  Esto representa una gran ventaja: si nos encontramos ante una nueva instancia que clasificar, por ejemplo un nuevo paciente, y a esta instancia le falta alguna característica por ser ésta complicada de obtener, o simplemente por error humano a la hora de anotarla (presión sanguínea, sexo...), clasificando con un único conjunto de características podríamos tener problemas ya que no tendríamos en cuenta ese factor de clasificación. Sin embargo, devolviendo varios conjuntos de características distintos no existiría tal problema, esto es, al tener más de un conjunto de características para clasificar la instancia podremos clasificarla sin necesidad de usar esa característica perdida.

  Para este objetivo se hace necesario utilizar técnicas de selección de características más avanzadas. Es por ello por lo que decidimos aplicar la **Optimización Evolutiva Multiobjetivo** (*Evolutionary Multiobjective Optimization*, **EMO**) a tal fin. Este tipo de algoritmos no devuelven una única solución, sino el conjunto de las mejores soluciones. Este planteamiento es el adecuado para conseguir nuestro objetivo.

Esta aplicación de la Optimización Evolutiva Multiobjetivo al problema de la selección de características nos lleva a plantearnos las siguientes tareas en el *Proyecto*:

- Desarrollar distintas propuestas multiobjetivo EMO para la selección de características y analizar el comportamiento de las mismas, comparando cuáles de las versiones implementadas son las mejores, quedándonos finalmente con las que tengan un comportamiento más adecuado.
- Comparar con otras técnicas clásicas de selección de características ya existentes. Para ello realizaremos un estudio muy amplio de técnicas existentes, eligiendo las más actuales y utilizadas, y viendo si realmente son superadas por nuestra propuesta evolutiva.

- **Diseñar un sistema de clasificación apropiado que se ajuste a las peculiaridades de trabajar con varios subconjuntos de características devueltos por los algoritmos evolutivos multiobjetivo**.

  Para ello, nada mejor que **utilizar un modelo de multiclasificación** formado por varios clasificadores. Cada uno de ellos se basará en un subconjunto de características de los devueltos por el EMO.

  Además, un algoritmo genético aprenderá automáticamente la configuración y parámetros más apropiados para el multiclasificador, tanto en relación a los clasificadores que habrá que utilizar, como a la configuración en conjunto.

Tendremos muy en cuenta estos dos objetivos primordiales, que deberán marcar el desarrollo del *Proyecto Fin de Carrera*. Pero además tendremos que hacer hincapié en la creación de un sistema e interfaz que envuelva toda la propuesta que aquí realizamos. Deberá de contar con las siguientes características generales:

- Cumplir los estándares actuales de ingeniería del software y diseño de patrones.
- Ser amigable y fácil de usar para los distintos usuarios.
- Que permita realizar y poner en práctica de forma cómoda todas las acciones y distintas configuraciones posibles que admite el sistema propuesto.

Aparte de las características nombradas, se han especificado ciertos requisitos que debe tener el sistema creado. Estos requisitos funcionales y no funcionales se encuentran detallados ampliamente en la sección 1 del 4º capítulo.

Planificación global del Proyecto Fin de Carrera

El desarrollo de todo el *Proyecto* está dividido en un conjunto de tareas para las cuáles se ha fijado su duración en semanas. La tabla muestra cuáles son estas y cuál es la duración total.

ACTIVIDAD	**ESTIMACIÓN (sem.)**
Búsqueda de información	3
Revisión bibliográfica	5
Estudio de propuesta evolutiva propia	4
Desarrollo Software del sistema de Minería de Datos	17
Evaluación de la herramienta, toma y análisis de resultados	4
Completar memoria	6
TOTAL	**39**

Es conveniente fijar ciertos hitos. Los hitos son objetivos intermedios en el proceso de desarrollo del *Proyecto* y constituyen pasos previos para la consecución de la meta final. Hemos identificado los siguientes:

- **Puesta al día bibliográfica**
- **Desarrollo y diseño del Sistema de Minería de Datos**
- **Evaluación de la herramienta y memoria. Fin del Proyecto**

Para poder visionar perfectamente todas las tareas junto a su paralelismo y los hitos definidos, hemos construido un diagrama de red (ver figura 1) para completar toda la información sobre la planificación global del *Proyecto*. En dicha figura 1 se muestran coloreados en rojo y verde los hitos, mientras que en naranja aparece el camino crítico de la planificación.

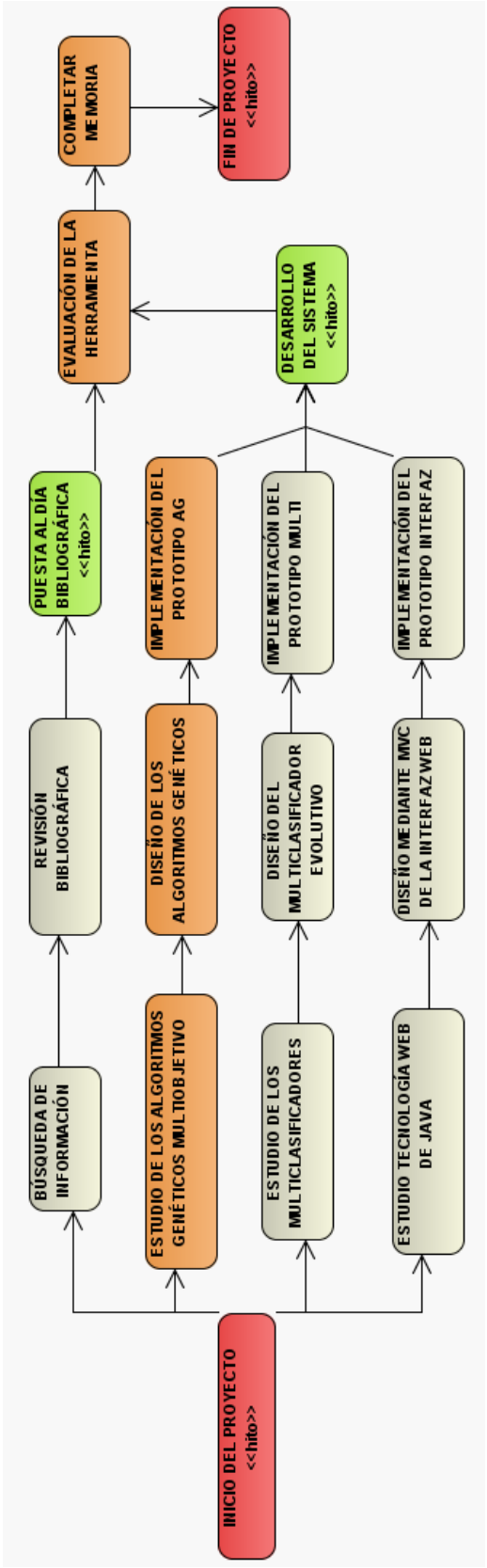

Figura 1: Diagrama de red con las tareas de todo el *Proyecto*.

Capítulo 1

El proceso de Minería de Datos

En este capítulo estudiaremos el área de la Minería de Datos. Lógicamente no podremos abarcarla en su totalidad ya que existen muchas técnicas, principios y temáticas relacionadas y existe además una amplia bibliografía al respecto que puede ser consultada para una mayor profundización. Nos centraremos en las partes de la Minería de Datos que vamos a utilizar en el *Proyecto*, esto es, preprocesamiento, y sobre todo **selección de características**, clasificador **kNN** y **sistemas de múltiples clasificadores**.

Desarrollando esta idea dedicaremos la primera sección para hacer una introducción en conjunto a la Minería de Datos, mientras que las tres secciones siguientes se dedicarán más profundamente a las partes en las que se centra el *Proyecto*.

1.1. ¿Qué es la Minería de Datos?

1.1.1. Definición. Descubrimiento de conocimiento

La mayoría de los datos que almacenan los ordenadores y las bases de datos (BBDD) crecen a un ritmo exponencial. Al mismo tiempo, los usuarios de esos datos necesitan información más sofisticada sobre los mismos. Así por ejemplo, un director de *marketing* no estaría satisfecho con obtener una simple lista con todos sus contactos de *marketing*, sino que querría información detallada sobre las compras o intenciones de los consumidores, así como predicciones sobre futuras compras.

Una consulta en lenguaje estructurado como *SQL* no es adecuada para soportar este tipo de necesidades de información. La Minería de Datos intenta resolver estas necesidades. Se suele definir como un *proceso de descubrimiento de información oculta de una base de datos* (figura 1.1). También se suele referenciar como *análisis explorativo de los datos* o *aprendizaje deductivo*.

Se deben desterrar ciertos mitos sobre la Minería de Datos. La Minería de Datos no es un procesamiento deductivo de consultas en bases de datos, no es un sistema experto, ni

un análisis puramente estadístico de los datos, ni por supuesto un pequeño programa de aprendizaje.

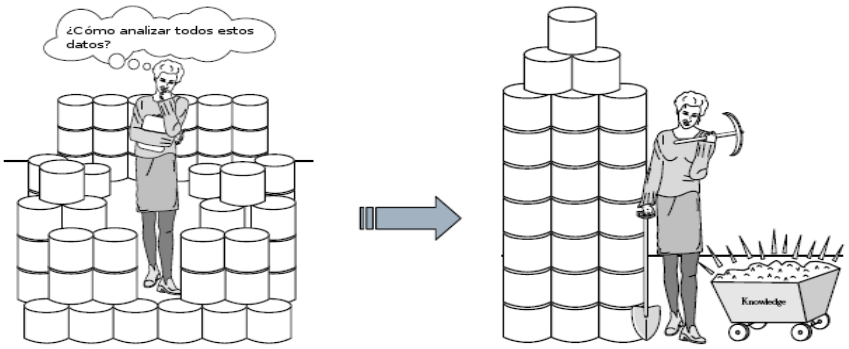

Figura 1.1: Objetivo de la Minería de Datos.

Las consultas tradicionales a bases de datos acceden a ellas mediante un lenguaje bien formado y estructurado. La salida de la consulta está formada por datos de la base de datos que satisfacen dicha consulta, normalmente un subconjunto de la base de datos, vistas sobre esa base de datos o ciertas agregaciones entre varias de ellas. La Minería de Datos difiere de este enfoque en lo siguiente:

- **Consulta:** no tiene que estar bien formada o bien definida, ya que es normal que sea más imprecisa. Normalmente el minero de datos no sabe qué esta buscando o no tiene la seguridad de qué es lo que tiene que consultar exactamente.
- **Datos:** los datos a los que se acceden suelen diferir de los originales por haberse aplicado técnicas de transformación para facilitar la labor de extracción de información.
- **Salida:** la salida de un proceso de Minería de Datos no suele ser un subconjunto de la base de datos, sino que será un análisis de los contenidos de la base de datos.

La minería de datos comprende muchos algoritmos distintos para diferentes tareas de las que hablaremos posteriormente. Todos estos algoritmos intentan formar un modelo que se acople bien a los datos. Los algoritmos examinarán los datos y devolverán el modelo que más se ajuste a sus características. Normalmente, los algoritmos de Minería de Datos tienen tres partes:

- **Modelo:** como ya sabemos, el objetivo del algoritmo será encontrar un modelo que se ajuste a los datos.
- **Preferencia:** algún criterio que nos permita decantarnos entre un modelo u otro.
- **Búsqueda:** todos los algoritmos requieren alguna técnica para buscar en los datos.

Un ejemplo de uso de minería de datos podría ser el siguiente:

Una compañía de tarjetas de crédito quiere determinar si autoriza o no las compras mediante tarjeta. Cada compra se situa en una de las siguientes cuatro clases: (1) autorizarla, (2) preguntar por cierta información de autorización antes de autorizar, (3) no autorizar o (4) no autorizar y contactar con la policía.

Para conseguir que ante un nuevo cliente que haga una compra con tarjeta de crédito se realice alguna de las 4 opciones anteriores hay que obtener un modelo a partir de datos históricos de compras con tarjetas de crédito. No nos valdría una simple consulta estructurada a los datos. La búsqueda requiere que se examinen los datos pasados sobre compras con tarjeta de crédito y los gastos de los clientes para determinar que criterio seguimos para definir la estructura del modelo. Así por ejemplo, nosotros podríamos querer autorizar una compra cuando ésta sea por una pequeña cantidad de dinero y la tarjeta pertenezca a un cliente antiguo. O no autorizaremos la compra cuando se haya notificado el robo de la tarjeta.

Como vemos, el ejemplo anterior utiliza una especie de predicción, ya que ante una nueva compra con tarjeta de crédito vamos a actuar de una forma u otra. Este tipo de minería de datos está dentro de la llamada **predictiva**. Junto a la **descriptiva**, forman los dos tipos de modelos en los que se compone la minería de datos:

- Un **modelo predictivo** realiza una predicción sobre valores de datos nuevos a partir de resultados conocidos o pasados. Suele estar basado en el uso de archivos históricos, por ejemplo, una compra con tarjeta de crédito puede ser rechazada no sólo por el historial del propio usuario de la tarjeta, sino porque la compra es similar a alguna anterior a ella, y que resultó peligrosa para los intereses de la compañía. La minería de datos predictiva comprende tareas como clasificación, regresión, análisis de series temporales y predicción.

- Por contra, un **modelo descriptivo** no predice nuevos casos, sólo identifica patrones o relaciones que existen en los datos. Es una manera de explorar las propiedades de los datos examinados, no de predecir nuevas propiedades. Dentro de este grupo se incluyen técnicas como reglas de asociación, clustering, descubrimiento de secuencias etc...

Los términos **Minería de Datos** y de **Descubrimiento de Conocimiento en BBDD** (*Knowledge Discovery from Databases*, **KDD**) son a menudo intercambiables. De hecho, hay otros nombres dados a este proceso de descubrimiento de patrones ocultos de los datos, como extracción de conocimiento, descubrimiento de información, reconocimiento no supervisado de patrones etc... En los últimos años, el KDD ha sido utilizado para referirse a un proceso consistente en varios pasos, mientras que la minería de datos es sólo uno de estos pasos:

Definición 1 *El **descubrimiento de conocimiento en bases de datos (KDD)** es el proceso de encontrar información útil y patrones en los datos.*

Definición 2 *La **minería de datos** es el uso de algoritmos para extraer la información y patrones derivados del proceso de KDD.*

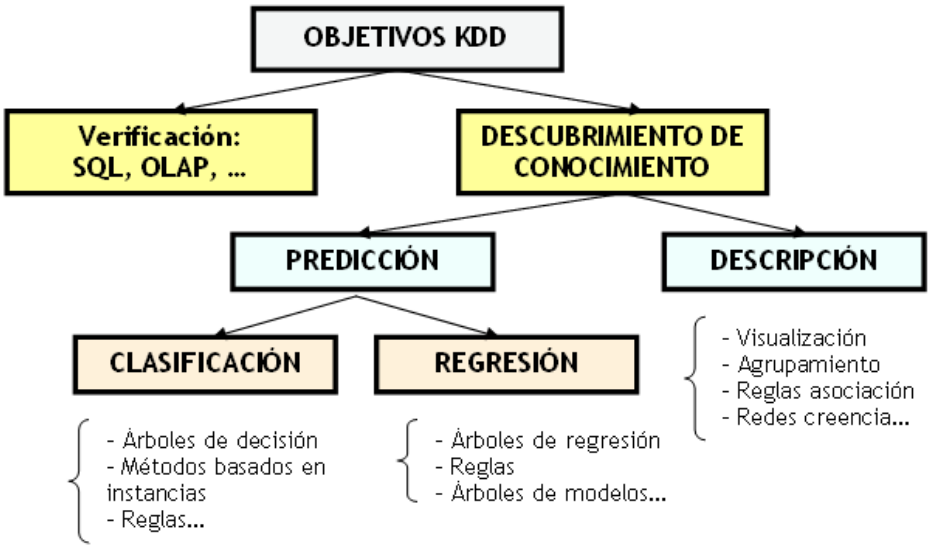

Figura 1.2: Clasificación de las distintas técnicas de KDD.

Como ya hemos dicho anteriormente, el proceso de KDD consta de varios pasos, en dónde la entrada son los datos, y la salida es la información útil pedida por los usuarios. Sin embargo, el objetivo puede ser poco claro e inexacto, siendo iterativo y requiriendo bastante tiempo. Para asegurarnos de la precisión y utilidad de los resultados del proceso son necesarios los conocimientos expertos.

1.1.2. Las distintas etapas del proceso de KDD

Hemos estado comentando cómo al proceso global de descubrimiento del conocimiento a partir de datos lo denominábamos KDD, incluyendo tanto el almacenamiento (***datawarehouse***) y acceso a los datos, utilización de algoritmos escalables para aplicarlos a cantidades enormes de datos, cómo se pueden interpretar y visualizar los resultados conseguidos, y de qué manera dar soporte a la interacción hombre-máquina durante todo el proceso.

El proceso de KDD contiene, un acceso y uso de la base de datos junto con técnicas de preprocesamiento, un muestreo y transformación sobre esa base de datos, algoritmos para obtener patrones y/o modelos a partir de esos datos (que sería la Minería de Datos en sentido estricto), y una evaluación del resultado de los algoritmos, seleccionando aquellos que puedan considerarse conocimiento.

Es un **proceso iterativo** e **interactivo**. Iterativo porque a veces son necesarias varias iteraciones para extraer conocimiento de alta calidad, e interactivo debido a que un experto en el dominio del problema debe ayudar tanto a la preparación de los datos, validación del conocimiento extraído etc...

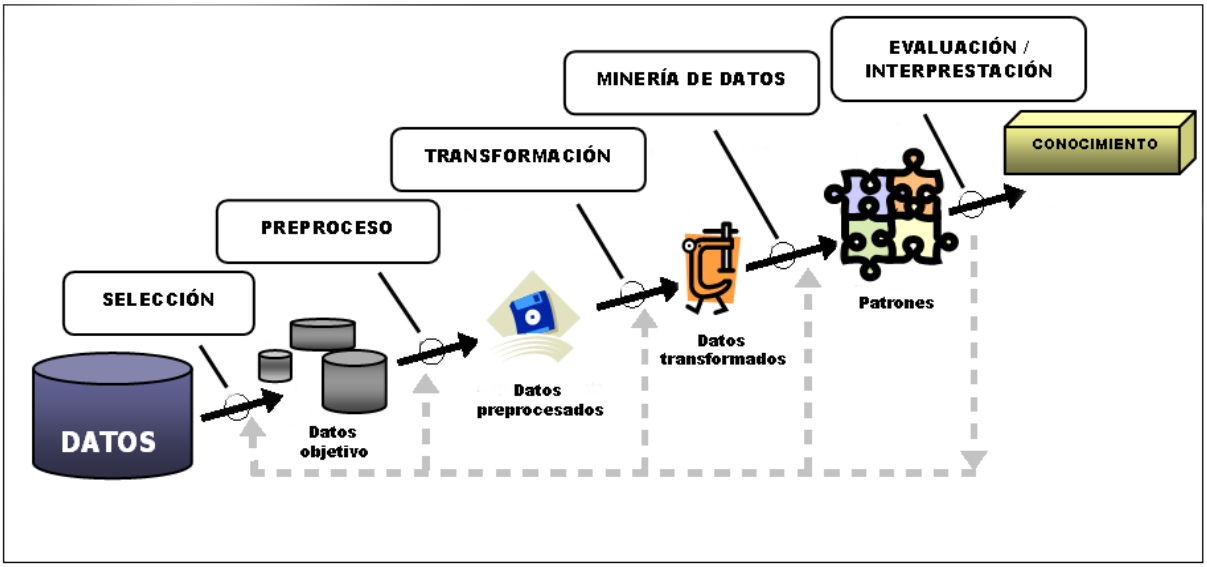

Figura 1.3: Distintas etapas del KDD.

Las distintas etapas forman el proceso de descubrimiento de conocimiento sobre bases de datos, cada una de las cuales con un objetivo claro. Son las siguientes y se esquematizan en la figura 1.3:

1. **Integración y recopilación de los datos**
2. **Preprocesamiento**
3. **Selección y aplicación de técnicas de Minería**
4. **Evaluación, interpretación y presentación de resultados**
5. **Difusión y utilización del modelo**

1. Integración y recopilación de los datos

La familiarización con el dominio del problema y la obtención de conocimiento a priori disminuye el espacio de soluciones posibles, consiguiendo más eficiencia en el resto de las etapas del proceso de KDD.

Además, en KDD se suele trabajar sobre problemas con datos de diferentes departamentos o ubicaciones, siendo conveniente agrupar y unificar la información. Es este el principal propósito de esta etapa, unificar la información en un almacén de datos o ***datawarehouse*** a partir de:

- Información interna como puedan ser distintas BBDD, u otro tipo de información (hojas de cálculo, informes de la organización, etc...).
- Estudios o documentos publicados: de demografía, catálogos, revistas o enciclopedias con información relevante para nuestro cometido.
- Otras bases de datos compradas, de industrias o empresas afines etc...

El resto del proceso de KDD será más cómodo si la fuente de datos está unificada, es accesible de forma interna y dedicada al proceso de extracción de conocimiento (no hace falta parar la marcha de la organización al ser diferente del acceso transaccional de la misma).

El almacén de datos o ***datawarehouse*** que hemos indicado es bastante conveniente para KDD aunque no es imprescindible, se puede trabajar directamente sobre las BBDD principales o con formatos heterogéneos.

2. Preprocesamiento

En esta etapa nos vamos a detener un poco más, ya que es la parte central del *Proyecto*. Además, dos secciones más adelante en este mismo capítulo nos centraremos en una de las técnicas fundamentales dentro de esta etapa, la **selección de características** (SC).

Decimos que las BBDD reales en la actualidad suelen contener datos ruidosos, perdidos y/o consistentes, fundamentalmente por su gran tamaño. En esta etapa se aplicarán distintos procesos a los datos para mejorar la eficiencia y la facilidad de aplicación del proceso de Minería de Datos. Podemos clasificar estas técnicas en cuatro puntos:

- Integración de datos
- Limpieza de datos
- Transformación de los datos
- Reducción de la dimensionalidad

Integración de datos

Los datos pueden obtenerse de diferentes fuentes y hay que considerar ciertas cuestiones al integrarlos todos en una. Así, tendremos que asegurarnos de que entidades equivalentes se emparejan correctamente cuando se produce esa fusión de información. Para lograr esto echaremos mano a los metadatos que normalmente se almacenan en las BBDD y los *datawarehouses*.

Otro aspecto importante es el de las redundancias. Es normal que ciertos datos se hallen en distintas fuentes. Diremos que un dato es redundante si puede obtenerse a partir de otros, una forma de detectarlos es mediante el **análisis de correlaciones**. El objetivo del análisis de correlaciones es el de medir la fuerza con la que un atributo implica a otro en función de

los datos disponibles. De esta forma, si varios atributos están muy correlacionados con otro, podremos prescindir de ellos ya que con la información de uno de ellos nos basta.

La correlación entre dos atributos **A** y **B** puede medirse como:

$$r_{A,B} = \frac{\sum(A - \overline{A}) \times (B - \overline{B})}{(n-1)\sigma_A \sigma_B} \tag{1.1}$$

Si el factor de correlación $r_{A,B}$ es menor que 0 significará que están correlacionados negativamente, es decir, si un atributo aumenta el otro disminuye. Si es mayor que 0 lo estarán positivamente, los dos tienen un comportamiento similar. Y si $r_{A,B} = 0$ serán dos variables independientes.

Además, para integrar los datos correctamente hay que detectar y resolver conflictos en los valores de los datos ya que puede que un atributo difiera según la fuente de procedencia. Estos valores distintos pueden deberse a diferencias de representación, escala o formas de codificar.

Limpieza de datos

Los objetivos de la limpieza de datos son:

- **Rellenar valores perdidos:** Cuando nos encontramos con valores perdidos podemos tomar distintos opciones. Una de ellas sería eliminar e ignorar todo el ejemplo que tenga valores perdidos, haciendo esto estaríamos perdiendo información. También podríamos utilizar una constante global, como *?*, para indicar que el valor no existe. O la que suele ser la más utilizada, rellenar ese valor perdido con medias y desviaciones estadísticas a partir de los valores del resto de tuplas.

- **Suavizar el ruido de los datos:** El ruido en los datos es un error o varianza aleatoria en la medición de una variable. Dependiendo del formato y del origen del campo puede ser detectado, así si es un valor numérico podemos ver si se ha salido del rango de valores posibles. Este ruido puede ser eliminado mediante técnicas de suavizado como el conocido *binning*, o mediante *regresión*, ajustando los valores a una función con técnicas de regresión.

- **Identificar y eliminar *outliers*:** Estos valores son correctos pero anómalos estadísticamente. Pueden ser un inconveniente para ciertos métodos de minería como las redes neuronales. Existen varias técnicas para conseguir detectarlos como definir una distancia y ver los individuos con mayor distancia media al resto de individuos, o realizar un clustering parcial y eliminar los elementos que se quedan fuera de los grupos.

Transformación de los datos

El objetivo de la transformación de datos es *moldear* los datos de la mejor forma posible para la aplicación de algoritmos de minería. Algunas técnicas típicas de transformación son la **agregación de atributos** a partir de otros, **normalización** de los valores de los atributos a un intervalo, normalmente el $[0, 1]$, o la **discretización** de valores.

En el *Proyecto* vamos a necesitar tanto **normalizar** como **discretizar** los datos. La normalización se suele realizar para pasar los valores de un intervalo a otro para por ejemplo, poder calcular adecuadamente las distancias en el algoritmo **kNN**[27]. En nuestro caso utilizaremos la **normalización min-max**, que opera así:

$$\text{Sea la conversión } [min_A, max_A] \longrightarrow [nuevo_{min_A}, nuevo_{max_A}]$$

$$\text{entonces un valor v será calculado como sigue:}$$

$$v\prime = \frac{v - min_A}{max_A - min_A} \times (nuevo_{max_A} - nuevo_{min_A}) + nuevo_{min_A}$$

La **discretización** es necesaria porque algunas medidas estadísticas o algoritmos que vamos a emplear para nuestro cometido, como el **ratio de inconsistencia**, necesitan que todos los atributos sean de tipo nominal o categórico. Para conseguir tal fin dividiremos todo el rango de posibles valores numéricos en cierto número de intervalos, asignándole a cada intervalo una etiqueta. El conjunto de todas las etiquetas formará el nuevo atributo discreto.

Nosotros utilizaremos una discretización uniforme, con 10 intervalos para convertir los atributos numéricos en categóricos. Sin embargo, los algoritmos que no necesiten esa discretización no recibirán como entrada los datos discretizados, ya que esta discretización hace que se pierda cierta precisión en la información.

Reducción de la dimensionalidad

Intentaremos obtener una representación reducida del conjunto de datos, de volumen mucho menor, pero sin perder en gran medida la integridad de los datos originales. La minería sobre el conjunto reducido resultante debe ser mucho más eficiente pero obteniendo las mismas o casi las mismas conclusiones. Sin duda es la operación más importante que existe en esta etapa. Le dedicaremos una amplia sección a un tipo de reducción como es la **selección de características** (SC).

El otro tipo de reducción es la **selección de instancias**, que consiste en obtener una representación más pequeña del conjunto de ejemplos que componen los datos. Podemos distinguir entre dos tipos de técnicas que consiguen ese objetivo:

- **Técnicas paramétricas:** Estiman un modelo a partir de los datos de forma que se almacenan sólo los parámetros y no los datos reales. Dentro de este tipo de selección de instancias tenemos la regresión lineal, en donde se estima un modelo de la forma: $y = \beta + \alpha_1 X_1 + \alpha_2 X_2 + ... + \alpha_n X_n$.
- **Técnicas no paramétricas:** Reducen la cantidad de datos mediante el uso de clustering y muestreo. Luego se intentará seleccionar un conjunto M del total de los N casos presentes en la BD original, siendo $M < N$. Podemos usar un muestreo sin o con reeemplazamiento, incremental o promediado.

3. Selección y aplicación de técnicas de Minería

El objetivo de esta etapa, una vez que tenemos los datos bien preparados y adecuadamente tratados, es el de producir nuevo conocimiento que pueda ser utilizado y serle útil al usuario. Esto se consigue construyendo un modelo, basándonos en los datos recopilados, que sea una descripción de los patrones y relaciones que existen entre los datos con los que se puedan hacer predicciones (*minería predictiva*), entender mejor los datos que poseemos o explicar situaciones pasadas (*minería descriptiva*).

Los pasos que tenemos que dar en esta etapa son los siguientes, primero tendremos que ver qué tipo de conocimiento queremos o buscamos, es decir, ¿queremos usar la *minería de datos descriptiva* para por ejemplo ver distintos grupos de clientes que existen entre los consumidores de una empresa?, o por el contrario deseamos utilizar *minería predictiva* para saber si un nuevo paciente tiene posibilidades de tener una enfermedad dada a partir del estudio de pacientes pasados.

Una vez que sabemos qué tipo de conocimiento queremos extraer, el siguiente paso a dar será el de elegir la técnica más adecuada. Así, podremos utilizar clasificación, reglas de asociación sobre los datos, agrupamiento o *clustering* etc...

Por último tendremos que saber qué modelo exacto deseamos aplicar (si estamos clasificando, podremos utilizar clasificadores basados en árboles de decisión, en reglas o redes neuronales) y qué algoritmo sería el más adecuado, ya que dentro de cada técnica existen bastantes algoritmos cada uno con sus virtudes y defectos.

4. Evaluación, interpretación y presentación de resultados

En la fase anterior de Minería de Datos se pueden producir varios modelos. Necesitamos saber cuales de ellos son los mejores. Para decidirnos entre uno u otro modelo nos basaremos en criterios como son la **precisión del modelo**, su **comprensibilidad**, y si resultan **interesantes** o **novedosos**. Hay unos modelos que son más comprensibles que otros, de esta manera una red neuronal por ejemplo no ofrece ninguna comprensibilidad, ya que no justifica de ninguna forma la decisión adoptada ni da razones o conocimiento sobre ella. Un conjunto de reglas de asociación sí será más comprensible ya que pueden ser entendidas más fácilmente por los humanos.

La técnica que se suele utilizar para poder evaluar a los distintos modelos generados consiste en la separación de los datos en dos conjuntos, uno llamado de **entrenamiento** o ***training*** y el otro de **prueba** o ***test*** (ver figura 1.4). El conjunto de datos de entrenamiento lo utilizaremos para extraer el conocimiento, es decir, aprender el modelo sobre ellos. Por contra, el conjunto de datos de prueba o *test* servirá para validar el modelo una vez que este ha sido creado, así, conseguiremos probar la validez del mismo ante datos que son totalmente nuevos y distintos respecto a los que habíamos aprendido.

El método más importante y que más veces aparece en la bibliografía especializada para validar un modelo es el de la **n-validación cruzada**. Su funcionamiento se basa en dividir

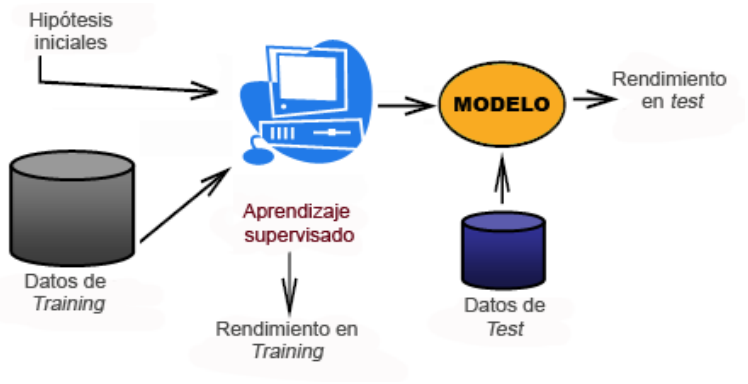

Figura 1.4: Esquema de evaluación de un modelo.

los datos originales en ***n*** particiones equitativas, ***n-1*** particiones formarán los datos de entrenamiento y la restante formará el conjunto de prueba.

Aparte de la **n-validación cruzada** que será la que nosotros utilicemos para validar nuestros modelos existen otras técnicas como ***bootstrapping*** o **validación simple**.

Estas validaciones que realizamos a los datos nos darán como resultado ciertas medidas de evaluación, éstas dependerán de las tareas realizadas:

- Para problemas de **clasificación** la medida utilizada será la precisión predictiva, igual al número de errores del modelo entre el número total de ejemplos probados.
- En cuanto a **regresión**, debido a sus peculiaridades continuas, será más adecuado utilizar el error cuadrático medio.
- Las técnicas de **agrupamiento** suelen ser validadas utilizando como medida la cohesión y la separación que existe entre los distintos grupos que se crean.
- Y las **reglas de asociación** mediante cobertura, confianza, etc...

5. Difusión y utilización del modelo

Una vez construido y validado el modelo puede utilizarse para recomendar acciones como si de un sistema experto se tratara, o aplicar el modelo creado a diferentes conjuntos de datos. En cualquier caso es necesario realizar:

- Difusión de los resultados, como por ejemplo elaborando informes para distribuir la información que ha aportado el modelo.
- Utilización del nuevo conocimiento de una manera independiente.
- Incorporación del modelo a sistemas ya existentes.

Además, habrá que monitorizar el sistema cuando esté en acción, dando lugar a una realimentación del proceso de KDD. Esto hará que las conclusiones iniciales puedan variar, invalidando el modelo previo.

1.1.3. Tareas básicas en Minería de Datos

Clasificación

La tarea de **clasificación** establece para cada instancia de los datos una *clase* o etiqueta. Se incluye dentro del área de aprendizaje supervisado porque las clases son determinadas antes de que se examinen los datos. Los algoritmos de clasificación requieren que las clases estén definidas basándose en los atributos de los datos. El proceso de clasificación se produce mirando los valores de los atributos de los datos conocidos que pertenecen a las clases que queremos establecer.

El reconocimiento de patrones es un tipo de clasificación en dónde un patrón de entrada es clasificado en varias clases basándose en similaridades a esas clases predefinidas.

Un ejemplo de técnica de clasificación son los **árboles de decisión** que mostramos en la figura 1.5.

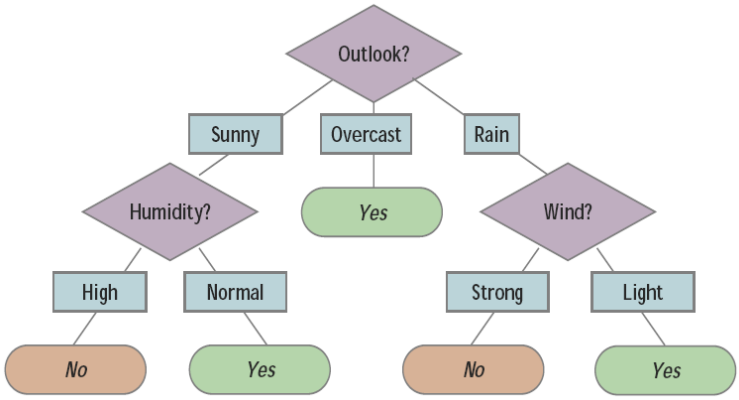

Figura 1.5: Diagrama de un árbol de decisión.

Regresión

La regresión (ver figura 1.6) se utiliza para agrupar elementos de datos en variables de predicción reales y continuas. La regresión contiene el aprendizaje de la función que realiza la predicción. Asumimos que los datos objetivo se ajustan a un determinado tipo de función como pueda ser logarítmica, lineal etc..., y determina la mejor función de ese tipo que modela los datos dados. Algunos tipos de análisis de error como el *error cuadrático medio* son utilizados para ver la validez del modelo de regresión.

A partir de la regresión obtenemos una función matemática que va a modelar los datos reales.

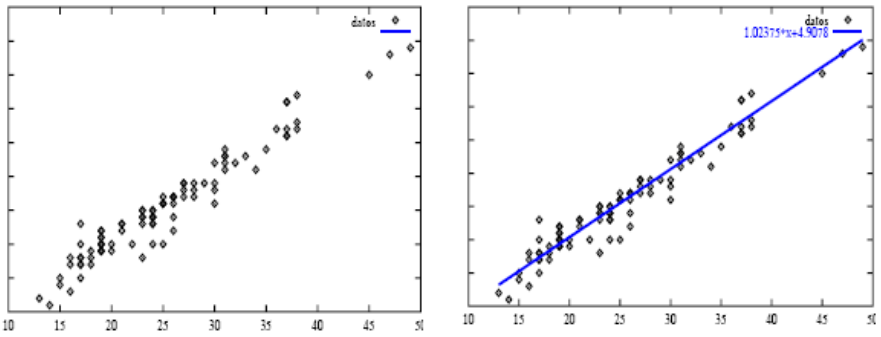

Figura 1.6: Ejemplo de función de regresión.

Análisis de series temporales

Con el análisis de series temporales, el valor de un atributo es examinado varias veces a lo largo del tiempo. Esos valores son obtenidos durante ciertos espacios de tiempo como diariamente, semanalmente o cada hora. Una gráfica de serie temporal es utilizada para visualizar la evolución de los datos como se puede ver en la figura 1.7.

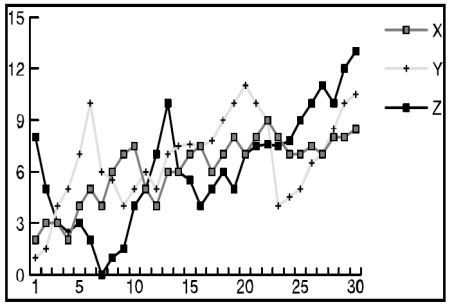

Figura 1.7: Distintas series temporales representadas mediante gráficas de tiempo.

Existen tres funciones básicas para ver el rendimiento de una serie temporal. En un primer caso, se utilizan medidas de distancia para ver esa similaridad entre diferentes series. En un segundo caso, la estructura de la línea de la serie temporal es examinada para ver su comportamiento. Una tercera aplicación es la utilización de series temporales para predecir valores futuros.

Predicción

Muchas aplicaciones reales pueden ser vistas como predicciones de datos futuros basándose en datos pasados y actuales. La predicción puede ser vista como un tipo de clasificación. La diferencia es que la predicción predice un estado futuro en vez de un estado actual. Aquí nos estamos refiriendo a un tipo de aplicación más que a un modelo de minería de datos. La predicción incluye reconocimiento de patrones, aprendizaje automático, etc...

Aunque los valores futuros pueden ser predichos usando análisis de series temporales o técnicas de regresión, también pueden utilizarse otros enfoques.

Clustering o agrupamiento

El *clustering* es similar a la clasificación salvo en que los grupos no están predefinidos, pero se definen mediante los datos. También es referida alternativamente como aprendizaje no supervisado o segmentación. Puede asemejarse a un particionamiento o segmentación de los datos en grupos que pueden ser disjuntos o no (figura 1.8).

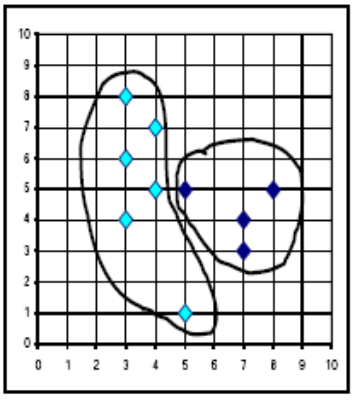

Figura 1.8: Dos *clusters* o grupos.

Debido a que los *clusters* o grupos no están predefinidos se requiere un conocimiento experto para interpretar el significado de los grupos creados.

Reglas de asociación

Las reglas de asociación no son muy diferentes de las reglas de clasificación excepto en que estas pueden predecir cualquier atributo, no sólo la clase de una instancia, y además predicen combinaciones de atributos. Además, las reglas de asociación no están entendidas como un conjunto de ellas, mientras que las reglas de clasificación son un conjunto de reglas. Expresan diferentes singularidades que existen en los datos, y pueden predecir distintas cosas.

A partir de un pequeño conjunto de datos se pueden inferir muchas y muy distintas reglas de asociación por lo que el usuario podrá restringir un umbral de precisión para que sólo se devuelven aquellas que lo superen. La **cobertura** de una regla nos dirá el número de instancias que puede predecir correctamente la regla, mientras que la **confidencia** nos dirá las instancias que consigue predecir correctamente la regla.

Por ejemplo, en la regla:

```
if temperatura=templada then humedad=normal
```

la **cobertura** de la regla será el número de días en que hace temperatura media y humedad normal, y la **confidencia** es la proporción de días templados que tienen la humedad normal. Es normal especificar un mínimo de cobertura y confidencia y buscar sólo las reglas que cumplen esos requisitos.

Resúmenes

Esta técnica asocia a subconjuntos de los datos breves descripciones sobre ellos. También se le suele llamar caracterización o generalización. Extrae información representativa de la base de datos. Puede ir acompañada de alguna recuperación de información.

Descubrimiento de secuencias

El análisis de secuencias o descubrimiento de secuencias se suele utilizar para determinar patrones secuenciales en los datos. Estos patrones están basados en un tiempo de acción. Son similares a las reglas de asociación, pero se diferencian en que las relaciones están basadas en el tiempo. Así por ejemplo el análisis de la cesta de la compra se pueden realizar acciones sobre productos que compran los clientes en un orden de tiempo, cosa que no se podía hacer con las reglas de asociación.

Un último ejemplo que puede dejar claro el modelo es el de la compra de CDs, podemos obtener conocimiento que nos diga que los clientes que compran un determinado CD comprarán otro CD dentro de la semana siguiente.

1.1.4. Aspectos a considerar en un proceso de Minería de Datos

Para finalizar con esta sección vamos a reseñar aspectos que hay que tener en cuenta cuando aplicamos un proceso de Minería de Datos independientemente de la técnica o algoritmo que utilicemos. Será importante conocerlos bien para tener éxito:

1. **Interacción humana:** Hay necesidad de interfaces con expertos en el domino y expertos técnicos.

2. **Sobreaprendizaje u *overfitting*:** Si sólo utilizamos los mismos datos tanto para aprender el modelo como para probarlo podemos estar cayendo en cierto sobreaprendizaje ya que el sistema puede que no responda correctamente ante datos nuevos. Para ello utilizamos técnicas como la **n-validación cruzada**, que separa los datos en ***test*** y ***training*** convenientemente.

   Existen determinados tipos de algoritmos, como algunos algoritmos de árboles de decisión, que al ajustarse tanto a los datos de entrenamiento no se comportan muy bien cuando se les introduce datos novedosos.

3. **Datos anómalos u *outliers*:** Es un gran problema para el proceso de descubrimiento de conocimiento. Son datos que no se ajustan a la mayoría de los datos haciendo que el proceso sea más dificultoso y llevándolo a errores.

4. **Interpretación de los resultados:** Suele requerir el uso de conocimiento experto.

5. **Visualización de resultados:** Es un punto muy importante a tener en cuenta ya que de poco nos servirá crear un buen modelo si luego no mostramos la información en un formato de visualización comprensible, de manera que intente satisfacer las necesidades del usuario.

6. **Grandes conjuntos de datos:** Grandes bases de datos o BBDD masivas crean problemas en algunos algoritmos de minería. En estos casos sería mejor perder precisión en detrimento de modelos que sean más escalables.

7. **Alta dimensionalidad:** Casi en todas las ocasiones nos enfrentaremos con datos con una alta dimensionalidad, bien sea por tener muchas variables o muchos ejemplos o instancias. La solución está clara, aplicar técnicas de preprocesamiento como selección de características o de instancias para reducir los datos de entrada.

8. **Datos multimedia:** Hay veces en las que no sólo tenemos datos textuales sino que nos enfrentaremos con imágenes (***spatial mining*** es un área de la minería de datos que trabaja sobre BBDD espaciales), sonidos u otro contenido multimedia.

9. **Datos perdidos:** Normalmente los datos provienen de archivos históricos y muchos de ellos no se encontraban en formato electrónico. Por esta y por más razones, también es lógico encontrarse con datos perdidos. Existen también algoritmos de preprocesamiento que rellenan esos valores perdidos utilizando, por ejemplo, la media de los que se encuentran en la base de datos.

10. **Datos irrelevantes:** No todos los datos de entrada que tenemos nos sirven para obtener conocimiento. El ejemplo más claro que se puede poner es una base de datos médica en la que cada paciente tiene un número identificativo tal como el DNI, si nosotros queremos predecir pacientes con cáncer ese campo no nos servirá absolutamente para nada. Este problema puede ser resuelto de nuevo gracias a algoritmos de preprocesamiento.

11. **Datos con ruido:** Algunos valores de variables pueden ser inválidos o incorrectos. Deben corregirse antes de la aplicación de algoritmos de Minería de Datos.

12. **Datos cambiantes:** Muchos algoritmos de Minería de Datos asumen datos estáticos. Sería interesante ver si nuestros datos van a cambiar en un futuro o no.

13. **Integración:** La integración de algoritmos de Minería de Datos dentro de sistemas tradicionales de BBDD es siempre un objetivo bastante deseable.

14. **Aplicación:** Determinar la intención de uso final de los datos es un reto.

1.1.5. Aplicaciones

A continuación vamos a ver algunos ejemplos sencillos de aplicación de la Minería de Datos a diversos campos. Por supuesto, el abanico de problemas potenciales es enorme, sólo pretendemos hacer ver su importancia y su gran valor práctico.

- **Entornos empresariales e industriales.** Sistemas de toma de decisiones en banca, seguros, finanzas, control de calidad detectando productos defectuosos o localizando los defectos prematuramente, automatizando procesos industriales favoreciendo así la reducción de gastos de la empresa.
- **Investigación científica.** Campos como medicina, meteorología, genética o geografía.
- ***Web Mining.*** Gracias al gran auge de Internet es uno de los campos más importantes. Publicidad personalizada según los intereses de cada usuario, incluso diferente *software spyware* que se basa en técnicas de minería.
- **Análisis y gestión de mercados.** También ofrece mucho interés para este área, resolviendo problemas como identificación de objetivos para *marketing* encontrando grupos que identifiquen un modelo de cliente con características comunes, determinando patrones de compra en el tiempo, analizando cestas de mercado, perfiles de cliente etc...
- **Análisis de riesgo en banca y seguros.** Detectando patrones de uso fraudulento en tarjetas, estudiando concesiones de créditos y/o tarjetas, identificando grupos y patrones de riesgo en el ámbito de los seguros, o por supuesto, identificando clientes leales y fuga de clientes.

1.2. Selección de Características

Definición 3 *La selección de características es un proceso que elige un subconjunto óptimo de características de acuerdo a algún determinado criterio.*

La SC se enmarca dentro de la etapa de preproceso de datos. El objetivo de esta etapa, como ya sabemos, es obtener una buena idea sobre los datos y preparar esos datos para la siguiente fase. La mejor forma de hacernos una idea sobre los datos es viendo sus atributos, comprendiendo cuáles son los más importantes para la clasificación y cuáles no intervienen para nada en el descubrimiento de conocimiento de los mismos etc...

La selección de características nos puede ayudar en las cinco fases del KDD, selecciona las características más relevantes, elimina irrelevantes y/o redudantes, mejora la calidad de los datos, hace que los algoritmos de minería de datos trabajen más rápido sobre datos de un gran tamaño y consiguen más comprensibilidad de los resultados obtenidos. Puede ser considerada como un conjunto de herramientas especiales que consiguen rellenar ciertas lagunas y aceleran el rendimiento de las herramientas existentes.

Características irrelevantes o redundantes pueden tener negativos efectos en la clasificación ya que:

- Tener más características implica la necesidad de tener más instancias para construir el modelo de clasificación correctamente.
- Variables redudantes o irrelevantes pueden producir un sobre-entrenamiento del modelo o un modelo erróneo.
- El clasificador obtenido será casi con toda seguridad un clasificador más complejo.

Básicamente vamos a querer elegir características que son relevantes para nuestra aplicación u objetivo final de conseguir el máximo rendimiento con el mínimo coste de computación. Los beneficios que produce la aplicación de una técnica de selección de características a una base de datos concreta son los de una **reducción de los datos** que tendremos que manejar, una **más alta precisión** final, **resultados más simples** y **menos características redundantes o innecesarias**.

Nosotros nos vamos a centrar en la SC aplicada a problemas de clasificación. La SC puede analizarse a través de varias perspectivas:

- Cuál será nuestro método de búsqueda de la mejor característica en cada momento.
- Qué deberíamos utilizar para determinar las mejores características, o cuál sería nuestro criterio de evaluación.
- Cómo generaremos el conjunto de características, lo haremos una a una secuencialmente, o construiremos conjuntos de ellas directamente.

En primer lugar tenemos que decir que la selección de características puede ser considerada como un problema de búsqueda de un subconjunto óptimo con la ayuda de algún criterio de evaluación que analizará dichas características para ir diciéndonos en cada momento qué subconjunto de las mismas será más prometedor o conveniente.

1.2.1. Dirección y estrategia de búsqueda

Como decimos, podemos mirar al proceso de SC como un problema de búsqueda en el que cada estado del espacio de búsqueda es un subconjunto de posibles características. Así por ejemplo, un conjunto de datos con tres características A_1, A_2, A_3, representado como un *array* binario, tendrá $2^3 = 8$ posibles soluciones o subconjuntos de características entre dos valores límite, el conjunto vacío (ninguna característica seleccionada) o el conjunto entero (las 3 características seleccionadas). En realidad tendríamos que considerar las soluciones entre estos dos casos extremos ya que no podremos devolver un conjunto de 0 características, y tampoco será normal devolver uno en el que no hayamos eliminado ninguna de ellas.

Un primer punto interesante a tener en cuenta en casos en los que tengamos un gran número de características a analizar sería ver el estado inicial. Normalmente, la SC se aplica sobre problemas de gran dimensionalidad de más de 20 características, y tendríamos que plantearnos por dónde empezar y qué estrategia de búsqueda emplear.

Tenemos ciertas direcciones de búsqueda que podemos utilizar en un algoritmo de SC, estas son:

- Generación secuencial hacia delante: empezamos con un conjunto de características seleccionadas vacío, y en cada paso vamos a ir eligiendo la característica más apetecible atendiendo a algún criterio. Añadiremos esa característica y seguiremos añadiendo una a una hasta completar un número de características deseado.

- Generación secuencial hacia atrás: es la dirección opuesta a la anterior. Partimos de un conjunto con todas las características posibles seleccionando y vamos quitando una a una siguiendo también algún mecanismo de decisión.

- Generación bidireccional: utiliza las dos anteriores. Tendremos dos flujos de dirección, hacia delante y hacia atrás. Se encontrarán en algún punto devolviendo ese conjunto final de características.

- Generación aleatoria: empezamos desde un punto de búsqueda aleatorio. En vez de tener inicialmente un conjunto vacío o lleno, obtenemos un conjunto de características aleatorias.

Una vez que sabemos qué dirección tomar para realizar la búsqueda, tendremos que tener en cuenta qué estrategia de búsqueda utilizar. Destacamos las más importantes aplicadas a este campo, recordando que en el capítulo 2 veremos más estrategias y técnicas de búsqueda y optimización:

- Búsqueda exhaustiva: consideramos todas las combinaciones posibles de características. No es un método adecuado cuando tenemos un gran número de atributos ya que se convierte en un problema NP-completo.

- Búsqueda heurística: utilizando alguna heurística para conducir la búsqueda. Obtenemos una solución razonablemente buena y con un buen tiempo de ejecución.
- Búsqueda no determinística: esta estrategia no consigue el siguiente conjunto de características según una regla determinística, sino que lo hace aleatoriamente.

1.2.2. Criterio de evaluación o selección

Al principio de la sección hemos comentado que un proceso de SC debería tener tanto una estrategia de búsqueda como algún tipo de función que nos fuera diciendo como de buenas o malas son las características que ibamos eligiendo.

La necesidad de evaluar si un atributo es bueno o no es común a todas las estrategias de búsqueda. Es una evaluación compleja y multidimensional, ya que tenemos que tener en cuenta si esa característica reduce el error de precisión, pero también si reduce la complejidad del modelo por ejemplo. De todas maneras, como nos estamos centrando en la SC para la clasificación, es normal que nuestro objetivo principal sea maximizar la precisión.

Las siguientes medidas son usadas con este propósito de ver qué característica o características se ajustan mejor para ser elegidas.

Medidas de Información

Utilizamos la **teoría de la información** para ver qué característica es la que nos proporciona más información sobre la clase. Ejemplos de medidas de este tipo son la *ganancia de información* [41] o el *ratio de ganancia* [37].

Medidas de consistencia. Ratio de inconsistencia

Tratan de encontrar las mejores características que separen totalmente las clases consistentemente, tal como lo haría el conjunto total de características.

Definición 4 *Para dos instancias o ejemplos existirá una* ***inconsistencia*** *siempre que las dos instancias tengan los mismos valores para sus atributos, y pertenezcan a distinta clase.*

Normalmente se utiliza el *ratio de inconsistencia* como medida de evaluación. Esta medida está basada en el concepto de inconsistencia y se calcula como sigue:

- Para un conjunto de N instancias con iguales valores en sus atributos, c_1 instancias pertenecerán a la clase 1 y c_2 instancias a la clase 2. Luego $c_1 + c_2 = N$, y la cuenta de inconsistencia será igual al número de instancias N menos el mayor número de instancias de una clase.
- El ratio de inconsistencia será igual a la suma de todas las cuentas de inconsistencia partido por el número total de instancias de todo el *dataset*.

Medidas de precisión

Esta tipo de medidas son obtenidas gracias a los clasificadores. Para un modelo de clasificación dado se elige el subconjunto de características que obtienen una mejor precisión. A la hora de elegir entre un modelo u otro de clasificación se tienen en cuenta factores como el sobreentrenamiento y la lentitud del proceso de clasificación. Suelen utilizarse clasificadores más rápidos en detrimento de una menor precisión.

1.2.3. Enfoque Filtro versus Enfoque Envolvente

La forma más simple de seleccionar características es utilizar la precisión de un clasificador como medida de su rendimiento. Si nuestro objetivo es minimizar el error de clasificación entonces, ¿por qué no usar la predicción de un clasificador? Luego lo que tendríamos que hacer es construir un clasificador cuyo objetivo sea el de conseguir la mayor precisión posible y seleccionar características usando este clasificador como guía para obtener el mejor subconjunto posible. Este modelo es conocido como **modelo envolvente** (figura 1.9).

Sin embargo, por las razones ya comentadas de lentitud o de posible sobreaprendizaje, se ha investigado también en el uso de otras medidas de rendimiento, la mayoría de ellas basadas en distancias o en medidas de información para seleccionar la característica más adecuada. En este caso, obtenemos el **modelo filtro** (figura 1.10).

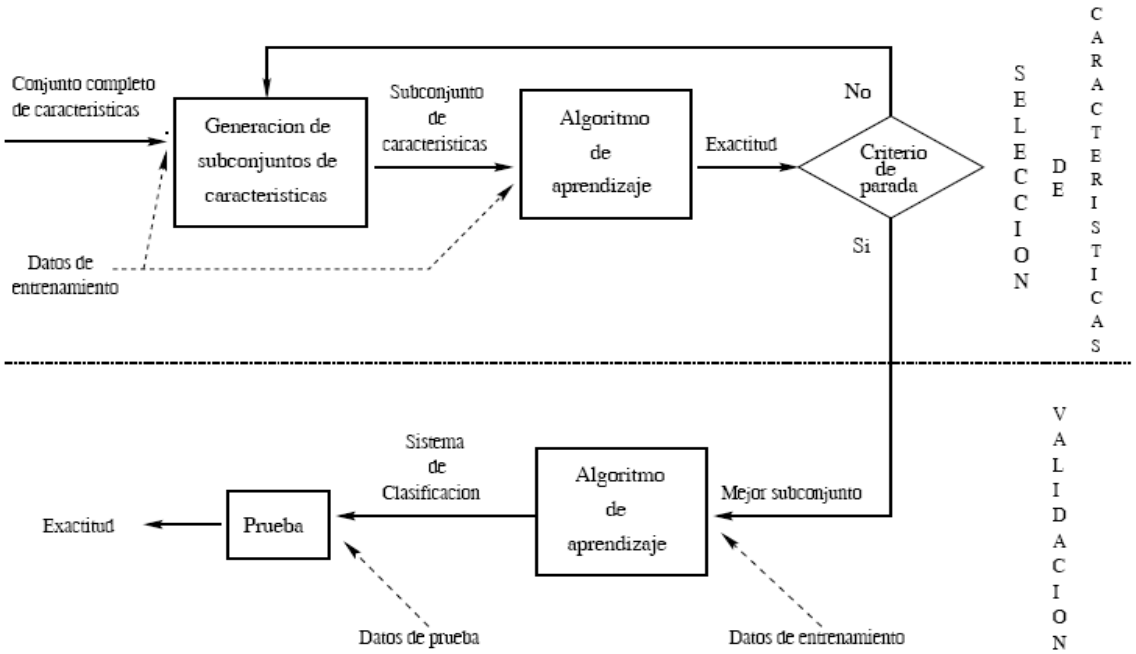

Figura 1.9: Modelo *envolvente* para SC.

La capacidad del modelo *envolvente* para manejar datos con gran dimensionalidad está sujeto a las limitaciones del clasificador. En Minería de Datos nos encontramos muy a menudo con conjuntos de datos inmensos, lo que hace casi imposible aplicar directamente un clasificador a los datos por lo que hay que utilizar antes otra técnica de preprocesamiento. Luego un modelo *envolvente* nos asegura la mejor precisión en la clasificación, siendo menos adecuado por su gran tiempo de cómputo, limitaciones debidas al tipo de clasificador utilizado (árbol de decisión, reglas...) e incapacidad para manejar datos muy grandes.

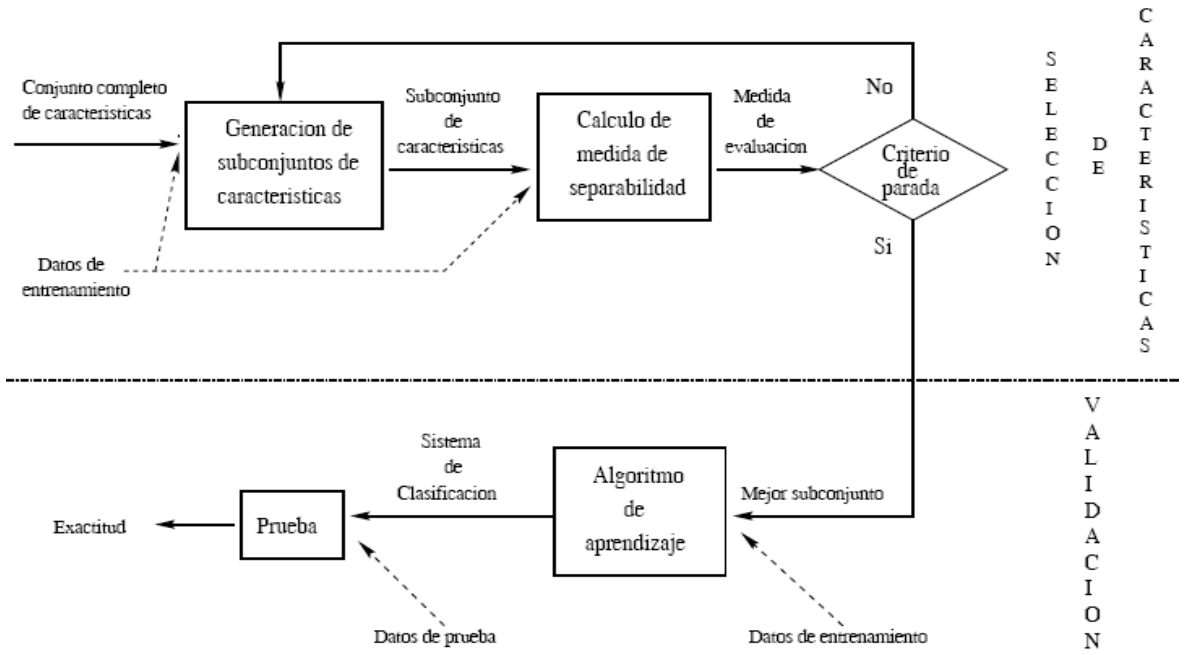

Figura 1.10: Modelo *filtro* para SC.

Un modelo *filtro* tiene las características de no ser dependiente de ningún clasificador en especial, de manejar medidas estadísticas que son usualmente menos costosas computacionalmente que la precisión de un clasificador, y además tiene la posibilidad de manejar datos muy grandes por las características anteriores.

1.2.4. Algunos algoritmos de Selección de Características

Algoritmos Voraces

Los algoritmos voraces destacan porque en cada paso del mismo se intenta buscar la mejor opción posible, sin pensar en la solución global. Según utilicemos una dirección de búsqueda u otra, tendremos dos posibles algoritmos de esta naturaleza. El primero de ellos presenta una **estrategia hacia adelante**, es decir, en cada paso va añadiendo una característica al

conjunto de seleccionadas hasta que se dé cierta condición de parada. El esquema del mismo es presentado en el algoritmo 1.1:

Algoritmo 1.1 (Voraz hacia adelante)

Inicializamos *el cto. de características seleccionadas a vacío:* $S = \{\}$
Repetimos
Obtenemos la mejor característica posible según algún criterio: f =*FindNext(F)*
Incluimos la característica en el conjunto: $S = S \cup \{f\}$
Eliminamos la característica del cto. principal: $F = F - \{f\}$
Hasta que se satisfaga la condición U
Devolvemos S

Análogamente podemos definir el mismo algoritmo voraz para una **estrategia hacia atrás** (algoritmo 1.2). Este algoritmo suele seleccionar más características que el de enfoque hacia adelante, ya que desde un principio parte con todas las características seleccionadas.

Algoritmo 1.2 (Voraz hacia atrás)

Inicializamos *el cto. de características desechadas a vacío:* $S = \{\}$
Repetimos
Obtenemos la peor característica posible según algún criterio: $f = GetNext(F)$
La excluimos del cto. principal de características: $F = F - \{f\}$
Incluimos la característica en el cto. de desechadas: $S = S \cup \{f\}$
Hasta que se satisfaga la condición U
Devolvemos F

Otro algoritmo de la misma familia es el implementado a partir de una dirección de búsqueda bidireccional. Puede considerarse como una mezcla de los dos enfoques anteriores.

LVF

Este método puede ser encuadrado dentro de los algoritmos estocásticos para selección de características. Desarrollado por *Liu* y *Setiono* en 1996 [35] es uno de los algoritmos filtro más importantes y famosos que existen. Sus siglas derivan de *Las Vegas Algorithm for Filter feature selection*, y su funcionamiento es bastante sencillo.

Se genera aleatoriamente un subconjunto de características, evaluándose su bondad respecto a alguna medida, originalmente el ratio de inconsistencia. Si el subconjunto de características supera cierto valor de bondad (especificado como parámetro de entrada) y además tiene un buen número de características seleccionadas, se devuelve como salida. Si no, seguimos repitiendo el bucle hasta un número determinado de intentos (ver algoritmo 1.3).

Algoritmo 1.3 (Las Vegas *Filter*)

$Card_{best} = N$
for *maxLoops* ***do***
    Aleatoriamente seleccionamos un subcto. de características $S = RandomSet(seed)$
    Calculamos su cardinalidad: $Card = |S|$
    if $(Card < Card_{best} \wedge CalcInconsistency(S) < \gamma)$
        Establecemos como mejor subcto: $S_{best} = S$
        Y mejor cardinalidad: $C_{best} = C$
    end if
Hasta que se satisfaga la condición U
Devolvemos S_{best}

También se puede cambiar algún componente del algoritmo LVF, así por ejemplo si en vez de usar una medida de evaluación utilizamos la precisión de algún clasificador obtendremos el algoritmo LVW (*Las Vegas algorithm for Wrapper feature selection*) [34].

Focus

Propuesto por *Almuallim* y *Dietterich* en 1991 [25], considera todas las combinaciones posibles de N características empezando por el conjunto vacío, luego por $\begin{pmatrix} N \\ 1 \end{pmatrix}$ subconjuntos de 1 elemento, por $\begin{pmatrix} N \\ 2 \end{pmatrix}$ subconjunto de 2 elementos etc... Cuando el algoritmo Focus descubre un subconjunto que satisface cierta medida de consistencia, para. Se trata de un método que va explorando exhaustivamente todo el espacio de soluciones por niveles hasta encontrar una solución que satisface la condición de parada, en este caso, que el ratio de inconsistencia sea igual a 0 (ver algoritmo 1.4).

Algoritmo 1.4 (Focus)

Inicializamos *el cto. de características a vacío:* $S = \{\}$
for $i = 1$ ***to*** N
    for each *subconjunto* S *de tamaño* i
        if $CalcInconsistency(S) = 0$
            Devolvemos S
end for

Relief

El algoritmo Relief selecciona las características que son estadísticamente relevantes [31]. Aunque su objetivo sigue siendo el de escoger atributos, no genera subconjuntos explícitamente como lo hacían los algoritmos anteriores. En vez de esa generación, lo que trata es de muestrear instancias y ver qué características son las que teniendo en cuenta sus valores pueden diferenciar instancias que son cercanas entre ellas.

El concepto se basa en obtener para una instancia (I) las dos más cercanas, cada una de una clase, la instancia H compartiendo la clase de I y la instancia J no. Entonces una característica será relevante si sus valores son los mismos entre I y H, y distintos entre I y J. Esta comprobación de valores se puede hacer utilizando medidas de distancia, la distancia entre I y H debe ser mínima y máxima entre I y J.

Para cada instancia seleccionada al azar se calcularán esas distancias, acumulándolas en un vector **w**. Las características relevantes serán aquellas cuyo peso w supere cierto umbral de relevancia τ. El algoritmo se muestra a continuación:

Algoritmo 1.5 (Relief)

Inicializamos $w = 0$
for $i = 1$ ***to*** m
Seleccionamos aleatoriamente una instancia I
Encontramos sus instancias más cercanas H(misma clase) y J(distinta clase)
for $j = 1$ ***to*** N_{caracs}
$w(j) = w(j) - diff(j, I, H)^2/m + diff(j, I, J)^2/m$
end for
Devolvemos las características cuyo valor $w(j)$ sea mayor que el umbral τ

1.3. Clasificación basada en instancias: kNN

kNN [27] son las siglas de *k-nearest neighbours* (los k vecinos más cercanos). Es un clasificador no paramétrico en el cuál no existe prácticamente modelo que construir, por eso se encuadra dentro de los clasificadores denominados como *perezosos*. Esto significa que por ejemplo, en los árboles de decisión se construyen nodos, hojas etc... que luego se usarán para clasificar nuevas instancias. Pero en el **kNN** no hay que construir y almacenar nada, simplemente se decide la clase de la nueva instancia inspeccionando los datos de entrenamiento y realizando comparaciones entre éstos y los nuevos datos a clasificar.

Sólo posee un parámetro, **k**, que determina el número de vecinos más cercanos con los que comparar la nueva instancia. El proceso de clasificación del **kNN** tiene también una gran ventaja: es bastante sencillo.

Tenemos un conjunto de instancias de entrenamiento, el objetivo será clasificar una nueva instancia. El algoritmo **kNN** irá examinando una a una todas las instancias de entrenamiento

y comparándolas con la nueva (para la comparación se utilizan funciones de distancia, por ejemplo, la distancia euclídea). Una vez examinadas todas las instancias de entrenamiento, se cogen las **k** instancias que más se parecen a la nueva y se observan sus clases. La clase más votada por las **k** instancias de entrenamiento más cercanas será la que se asigne a la nueva instancia. Normalmente el valor **k** suele ser un número impar para que no haya empate entre clases (si $k = 4$, dos clases pueden indicar la clase A, y las otras dos la clase B, por lo que no se sabría qué clase asignar).

En la imagen 1.11 se aprecian dos procesos **kNN** para $k = 1$ y para $k = 3$, existiendo dos posibles clases (círculo rojo y cuadrado azul) que se asignarán a la nueva instancia (valor en negro) según las instancias más cercanas a ella.

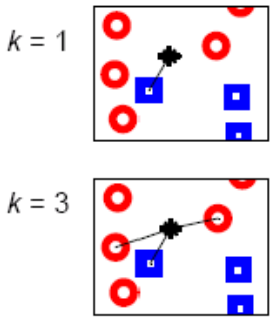

Figura 1.11: Funcionamiento del kNN con distintos valores de k.

Las características más importantes del **kNN** son las siguientes:

- **Proceso de clasificación bastante costoso.** Para determinar el vecino más cercano a una instancia hay que calcular la distancia entre ella y los N ejemplos de entrenamiento. Se puede ganar eficiencia con una pre-ordenación utilizando estructuras de datos eficientes (*kd-trees*). También podemos ahorrar tiempo de computación utilizando sólo una distancia aproximada o eliminando datos redundantes.

- **Requisitos de almacenamiento.** Se deben guardar las instancias de entrenamiento para clasificar después (los árboles no necesitan los datos de entrenamiento una vez que se han construido).

- **Elección de medida de distancia.** Hay que elegir qué medida de distancia utilizar, la más común es la distancia euclídea entre dos puntos, pero también podemos optar por métricas locales o por técnicas más complicadas que no utilicen datos numéricos.

- Normalmente el **1NN** (**kNN** con el parámetro $k = 1$) consigue en la práctica el mejor rendimiento.

1.4. Multiclasificadores

Se pueden construir distintos clasificadores para un mismo conjunto de datos Z en vez de usar un único clasificador. Sea $D = D_1, D_2, ..., D_L$ un conjunto de L clasificadores, la idea es combinar ese conjunto de clasificadores para incrementar la precisión de clasificación que tendríamos con un único clasificador. Aunque teóricamente está demostrado que un multiclasificador obtiene mejores resultados que un único clasificador ésto no tiene porqué ser siempre así, nada nos lo garantiza.

Al igual que ocurre en la vida real, cuando se va a tomar una decisión importante se consulta no a un único experto, sino a un conjunto de ellos que darán una respuesta de consenso. Este paradigma se puede aplicar a la multiclasificación, ya que cada clasificador se corresponderá con un *experto* que dará su opinión. La decisión final no será resultado de la decisión de un único *experto*, sino que mediante alguna estrategia se intentará dar una opinión en la que todos los *expertos* sean partícipes.

Existen multitud de estudios y bibliografía sobre sistemas de múltiples clasificadores. Los distintos sistemas se diferencian en cuanto al tipo de clasificadores que tenemos, la salida de clasificación, la estrategia de combinación de los mismos o el procedimiento de agregación. Los clasificadores individuales que usaremos en el multiclasificador pueden ser construidos atendiendo a las siguientes opciones:

- **Diferentes conjuntos de variables.** Conveniente sobre todo cuando tenemos un gran número de variables en la base de datos.
- **Diferentes conjuntos de datos.** Podemos separar y crear distintos subconjuntos de datos a partir del conjunto total Z. Cada clasificador se ocupará de un subconjunto de datos.
- **Diferentes tipos de clasificadores.** Un clasificador puede ser un árbol de decisión, otro clasificador una red bayesiana etc...
- **Cualquier combinación de los tres elementos anteriores.**

De acuerdo al tipo de salida de clasificación, el autor *Xu et al.* [40] divide los clasificadores en:

- **Tipo 1: *abstract level classifiers:*** La salida de clasificación será un valor *crisp* que identifica la etiqueta de la clase:
$$D : \Re^n \rightarrow \Omega$$
- **Tipo 2: *rank level classifiers:*** La salida será un conjunto ordenado de etiquetas de clase.

- **Tipo 3: *measurement level classifiers:*** La salida será en este caso un vector c-dimensional con grados de soporte para cada clase:

$$D : \Re^n \rightarrow [0,1]^c$$

Siguiendo con los tipos de multiclasificadores que existen, hay generalmente dos tipos de combinación de clasificaciones, **selección de clasificadores** y **fusión de clasificadores**. La presunción en la selección de clasificadores es que cada clasificador es un *experto* en alguna área local del espacio de características. Cuando un vector de características $x \in \Re^n$ es dado al sistema, el clasificador más apropiado para ese área del espacio es el elegido y el único que dará una respuesta. Sólo un clasificador se tendrá en cuenta para cada subconjunto de características, lo que se denomina *experto local*. Por contra, con la fusión de clasificadores asume que todos los clasificadores son expertos para todo el conjunto del espacio de características. En este caso, los clasificadores se consideran **competidores** en vez de los **complementarios** de la selección de clasificadores.

Estas dos ideas pueden ser mezcladas. En vez de nominar un único *experto* por dominio local, podemos nominar un conjunto pequeño de ellos. Luego podemos darle un peso a cada *experto* del conjunto, o considerar al que tenga más precisión como el *líder del grupo*. La salida puede ser la clase más votada por el grupo de *expertos*.

1.4.1. Selección de clasificadores

La selección de clasificadores (ver el esquema de la figura 1.12) fue propuesta por *Dasarathy* y *Sheela* en 1978 [26] utilizando un clasificador **kNN** y un clasificador lineal. Sugirieron identificar un dominio de conflicto del espacio de características para utilizar en él el clasificado **kNN**, utilizando el lineal en el demás espacio.

Denotaremos la salida del i-esimo clasificador como $D_i(x) = [d_{i,1}(x), d_{i,2}(x), ..., d_{i,c}(x)]^T$ en dónde $d_{i,j}(x)$ es el *grado de soporte* dado por el clasificador D_i a la clase j según la instancia x. El espacio de características $\Re^n$ lo dividiremos en $K > 1$ regiones, denotándolas por $R_1, ..., R_n$. Usaremos un clasificador del conjunto de clasificadores $D = D_1, ...D_L$ para cada región R_i del espacio de características, y usaremos a este para clasificar todas las instancias x de cada su región R_i.

Hay varias formas de diseñar una selección de clasificadores:

1. **Estática.** Las regiones del espacio son especificadas antes de clasificar la instancia x. Dos enfoques son posibles para llevar a cabo este clasificador estático, la primera es especificar las regiones y luego asignar un clasificador responsable por cada región, y la segunda consiste en que dado un clasificador D hay que encontrar una región en dónde el clasificador sea el mejor de todos. Aunque la segunda forma es posiblemente más eficiente, es más difícil de implementar que la primera.

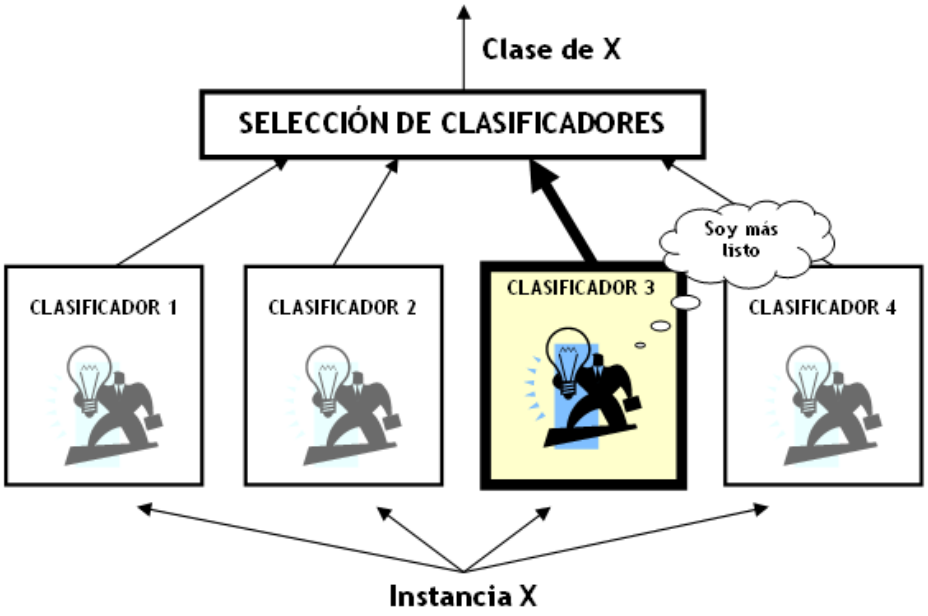

Figura 1.12: Selección de clasificadores.

2. **Dinámica.** La elección de un clasificador para la instancia x se basa en la certeza de la decisión de cada clasificador. Un ejemplo de este enfoque es el que da *Woods* [39], dada una instancia x, la precisión de cada clasificador es estimada sobre las instancias más cercanas a x. El clasificador con la más alta precisión es el autorizado a dar una etiqueta de clase para la instancia x. De esta manera, las regiones son estimadas durante el proceso de clasificación dinámicamente.

1.4.2. Fusión de clasificadores

La figura 1.13 muestra un esquema de un multiclasificador que utiliza fusión de clasificadores. Consideremos de nuevo el conjunto de clasificadores $D = D_1, ..., D_L$. Construiremos $\widehat{D}$, la salida fusionada de los L clasificadores, como:

$$\widehat{D} = F([D_1(x), D_2(x), ..., D_L(x))$$

en dónde F es una regla de agregación

Varios grupos de métodos de fusión puede ser definidos en base a la función de agregación F. A continuación veremos tres de las funciones más importantes de agregación de *opiniones* de los clasificadores.

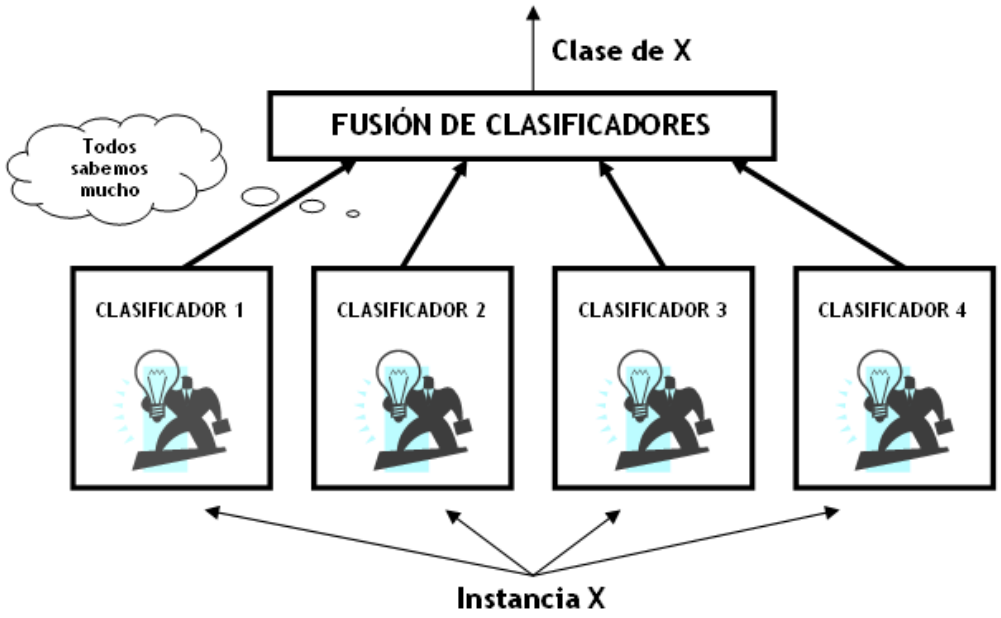

Figura 1.13: Fusión de clasificadores.

Voto de la mayoría

El voto de la mayoría es una técnica de fusión muy popular. Cada clasificador individualmente asigna una única etiqueta de clase a la instancia x, es decir, el clasificador *vota* por la clase. La clase final de x será la clase más votada por el conjunto de todos los clasificadores. Algunas veces puede que sea interesante considerar como la clase más votada a aquella que tenga *mayoría absoluta*, entendiendo que la clase ha sido votada por más de la mitad de los clasificadores.

NaiveBayes

Este esquema asume que los clasificadores son mutuamente independientes. Para cada clasificador D_j se crea una matriz de confusión CM_j aplicando D_j a los datos de entrenamiento.

Para la entrada (k, s) de la matriz, $cm^j_{k,s}$ es el número de elementos del conjunto de datos cuya clase real es w_k y le fue asignada la clase w_s por el clasificador D_j. Por $cm^j_{.,s}$ denotaremos el número de elementos clasificados por D_j como w_s. Usando $cm^j_{.,s}$ crearemos una matriz de $c \times c$ llamada LM^j, en dónde para la (k, s)-esima entrada, $lm^j_{k,s}$ es una estimación de la probabilidad de que siendo la clase real w_k, el clasificador D_j le asigne la clase s:

$$lm^j_{k,s} = \hat{P}(w_k | D_j(x) = w_s) = \frac{cm^j_{k,s}}{cm^j_{.,s}}$$

El numerador de la ecuación anterior es el número de veces que clasificando como w_s, realmente la clase ha sido k, mientras que el denominador representa el número de ocasiones totales en las que se ha clasificado como w_s.

Para cada $x \in \Re^n$, D_j produce un vector $D_j(x)$ que enlaza a una de las posibles clases. Considerando la matriz producida por D_j, LM_j, la fila de w_s sería un vector $[\widehat{P}(w_1|D_j(x) = w_s), ..., \widehat{P}(w_c|D_j(x) = w_s)]^T$, que es la s-esima columna de la matriz.

Luego por la suposición de independencia que hicimos, la probabilidad estimada de que la clase verdadera sea w_i se calcula como:

$$\mu_i(x) = \prod_{j=1}^{L} \widehat{P}(w_i|D_j(x) = s_j) = \prod_{j=1}^{L} lm^j_{i,s_j}, i = 1, ..., c$$

Aunque pueda parecer una técnica de fusión bastante compleja por su alto contenido matemático, demostraremos con el siguiente ejemplo que no es así.

Ejemplo: *Consideremos 2 clasificadores D_1 y D_2, y 3 posibles clases, que son w_1, w_2 y w_3. Tendremos 20 ejemplos de training de los cuáles 8 son de la clase w_1, 9 de w_2 y 3 de w_3. Supongamos las siguientes matrices de confusión para cada uno de los dos clasificadores:*

$$CM^1 = \begin{pmatrix} 6 & 2 & 0 \\ 1 & 8 & 0 \\ 1 & 0 & 2 \end{pmatrix} \text{ y } CM^2 = \begin{pmatrix} 4 & 3 & 1 \\ 3 & 5 & 1 \\ 0 & 0 & 3 \end{pmatrix}$$

Las dos matrices obtenidas a partir de CM^1 y CM^2 son:

$$LM^1 = \begin{pmatrix} 6/8 & 2/10 & 0 \\ 1/8 & 8/10 & 0 \\ 1/8 & 0 & 1 \end{pmatrix} \text{ y } LM^2 = \begin{pmatrix} 4/7 & 3/8 & 1/5 \\ 3/7 & 5/8 & 1/5 \\ 0 & 0 & 3/5 \end{pmatrix}$$

Asumiento también que los clasificadores obtienen $D_1(x) = w_2$ y $D_2(x) = w_1$ para la entrada $x \in \Re^n$, usando la segunda columna de LM^1 y la primera de LM^2, calculamos la salida del multiclasificador NaiveBayes de la siguiente forma:

$$\mu_1(x) = \widehat{P}(w_1|D_1(x) = w_2)\widehat{P}(w_1|D_2(x) = w_1) = \frac{2}{10} \times \frac{4}{7} = \frac{4}{35}$$

$$\mu_2(x) = \widehat{P}(w_2|D_1(x) = w_2)\widehat{P}(w_2|D_2(x) = w_1) = \frac{8}{10} \times \frac{3}{7} = \frac{12}{35}$$

$$\mu_3(x) = \widehat{P}(w_3|D_1(x) = w_2)\widehat{P}(w_3|D_2(x) = w_1) = 0$$

El multiclasificador devolverá que el valor de la clase para x será w_2.

BKS

Sean $(s_1, s_2, ..., s_L) \in \Omega^L$ las clases asignadas a la instancia x por los L clasificadores $D_1, D_2, ..., D_L$ respectivamente. Cada posible combinación de clases es almacenada en una celda de una tabla especial llamada **tabla BKS**. La tabla es diseñada para un conjunto de datos Z, cada z_j es situada en una celda de la tabla, indexada por $D_1(z_j), ..., D_L(z_j)$. La clase más representativa de cada columna es almacenada en la **tabla BKS**. De esta manera, cada entrada de la **tabla BKS** será o una clase, ninguna clase porque no se dé la combinación o un conjunto de clases (si hay empate).

Ejemplo: *Sean 3 clases w_1, w_2, w_3 y 100 instancias, la **tabla BKS** para dos clasificadores sería la siguiente:*

$s_1, s_2 \rightarrow$	1,1	1,2	1,3	2,1	2,2	2,3	3,1	3,2	3,3
w_1	10	3	5	0	1	4	7	0	0
w_2	3	0	4	0	16	4	2	2	0
w_3	3	6	5	0	16	4	4	5	6
Etiqueta	w_1	w_3	w_1, w_3	Ninguna	w_2	w_1, w_2, w_3	w_1	w_3	w_3

Tabla 1.1: Ejemplo de tabla BKS.

La decisión para x del multiclasificador se realiza de acuerdo a la combinación de salidas de los dos clasificadores. Así, si se obtienen las clases 1 y 2, la respuesta del clasificador sería w_3. Los empates se rompen aleatoriamente. Si se obtiene un valor *Ninguno* se elige aleatoriamente una clase de todo el conjunto posible.

1.4.3. Bagging y Boosting

Para finalizar con el tema de los multiclasificadores vamos a ver dos algoritmos que utilizan una **fusión de varios clasificadores**.

La idea del multiclasificador **Bagging** es inducir n clasificadores en lugar de utilizar sólo un clasificador. Se utilizará una combinación de las respuestas o salidas que proporciona cada clasificador para generar la salida final.

Los clasificadores que podemos utilizar pueden estar basados en distintas técnicas, es decir, podemos implementar un multiclasificador que sea combinación de una red bayesiana, de un clasificador basado en instancias como es el caso del **kNN** y de un árbol de decisión **C4.5**.

El clasificador **Bagging** va a ser un multiclasificador típico que tiene como característica principal que cada clasificador se va a inducir independientemente. Frente a otros multiclasificador como el **Boosting**, en el cuál cada clasificador va a tener en cuenta los fallos del anterior para tomar su decisión, en el **Bagging** un clasificador i del conjunto de clasificadores m que lo forman, no va a ser influenciado por ninguno de los demás clasificadores. Como

en todo modelo de clasificación, tanto el **Bagging** como el **Boosting** van a tener dos fases, una primera de aprendizaje y otra de clasificación de instancias.

Los esquemas de cada fase del **Bagging** se muestran a continuación en los algoritmos 1.6 y 1.7.

Algoritmo 1.6 (Generación de modelos)

n: el número de ejemplos de la BD a utilizar en cada modelo
m: el número de modelos de clasificación a utilizar
for $i = 1$ ***to*** m
    Muestrear con reemplazamiento n ejemplos de la BD
    Aprender un modelo con ese conjunto de entrenamiento
    Almacenarlo en modelos[i]
end for

Al final del proceso hemos conseguido obtener m modelos de clasificación distintos (pudiendo ser distintos tipos de modelos) cada uno de los cuáles han aprendido a partir de un subconjunto distinto de instancias del problema.

Una gran ventaja de este modelo de multiclasificadores es la eficiencia, ya que cada clasificador va a aprender a partir de una pequeña parte de la BD, no de toda ella. Como vemos en el esquema, el proceso de creación de un clasificador es totalmente independiente del clasificador o clasificadores anteriores y posteriores en **Bagging**.

Algoritmo 1.7 (Clasificación)

for $i = 1$ ***to*** m
    Predecir la clase utilizando modelos[i]
end for
Devolver la clase predicha con mayor frecuencia

El proceso de clasificación está bastante claro, al tener m clasificadores, vamos a intentar clasificar la nueva instancia con cada uno de los clasificadores. No todos los clasificadores tienen por qué estar de acuerdo, por lo que la clase elegida finalmente será igual a la clase más votada.

En el **Boosting** la diferencia está en que el peso de cada clasificador va variando según acierte o no, influyendo ese peso en el siguiente clasificador.

Capítulo 2

Estudio sobre Optimización Evolutiva Multiobjetivo

El objetivo de este capítulo es realizar un estudio sobre **Optimización Evolutiva Multiobjetivo** (*Evolutionary Multiobjective Optimization*, **EMO**) o **Algoritmos Genéticos Multiobjetivo** para poder aplicar esta técnica evolutiva de optimización y búsqueda al problema ya visto de la **selección de características**.

También describiremos ampliamente los **Algoritmos Genéticos** (**AAGG**) que utilizaremos en otras partes del *Proyecto*.

2.1. Introducción a la Computación Evolutiva

2.1.1. Metaheurísticas

En el mundo real existen muchos problemas en los cuales las técnicas algorítmicas básicas como *Programación Dinámica, Búsquedas Exhaustivas, Branch and Bound...* se nos presentan como insuficientes. Ejemplos de este tipo de problemas son:

- El problema del *Viajante de Comercio*.
- Asignar las tareas de una forma óptima a los trabajadores de una empresa.
- Ver el mejor enrutamiento posible de un paquete en Internet (existe un gran interés actualmente en este área, conocida como **Ingeniería del tráfico**).
- Encontrar la ubicación más correcta para un conjunto de hospitales entre todas las posibles localizaciones.
- Coloreado de un grafo.

Estos problemas, que normalmente requieren agrupaciones, ordenaciones o asignaciones que satisfagan un cierto conjunto de restricciones, son los llamados **NP-completos**, que son aquellos que no han sido capaces de resolverse utilizando un algoritmo con complejidad computacional polinomial, es decir, problemas que necesitan un tiempo factorial o exponencial. Para resolverlos se hace inviable utilizar algoritmos que exploren todo el espacio de soluciones posibles, bien porque no podamos disponer de horas de computación para dar una solución o bien porque incluso se tarden demasiados días en dar con ella.

La solución a este tipo de problemas pasa por el uso de **metaheurísticas**, que dan una buena solución, no necesariamente la mejor, en un tiempo razonable:

Definición 5 *Una metaheurística es un algoritmo aproximado de propósito general consistente en procedimientos iterativos que guían una heurística subordinada, combinando de forma inteligente distintos conceptos para explorar y explotar adecuadamente el espacio de búsqueda.*

Utilizaremos una metaheurística cuando no exista un método exacto para resolver un problema, o bien el que existe necesita muchos recursos. También la utilizaremos cuando no se necesite la solución óptima, sino una solución razonablemente buena. En el caso de que necesitemos *la mejor solución posible* sí sería necesario explorar todas las posibles soluciones.

Limitaciones de las metaheurísticas

A pesar de que las metaheurísticas son buenas herramientas para resolver problemas reales difíciles, no son la panacea. *Wolpert* y *McReady* postularon el siguiente teorema conocido como **Teorema de *No Free Lunch*** [21] (traducido, más o menos, como que *no hay comida gratis*):

Teorema 1 *... para cualquier algoritmo, un rendimiento elevado en un tipo de problemas hace que se pague exactamente en la misma medida con un rendimiento bajo en otro tipo de problemas.*

Lo que viene a reflejar el teorema de **No Free Lunch** es que cuando tenemos un conjunto de problemas suficientemente grande, habrá unas metaheurísticas mejores en unos problemas que en otros. No podemos decir que existe una metaheurística que vaya bien en todos los problemas y que sea mejor que todas las demás metaheurísticas. La figura 2.1 muestra tres metaheurísticas A_1, A_2, A_3 y su comportamiento en el dominio de problemas.

Problemas diferentes van a requerir diferentes técnicas. Y hay que tener en cuenta que cuando estemos diseñando una nueva técnica o metaheurística, ganar en un cierto dominio va a implicar perder en otros en la misma proporción.

Una consecuencia del teorema **No Free Lunch** es que el conocimiento experto sobre el problema debe ser incorporado a la metaheurística. Deberemos acomodar la técnica utilizada al problema a resolver, incluyendo en ella todo el conocimiento específico que sea necesario.

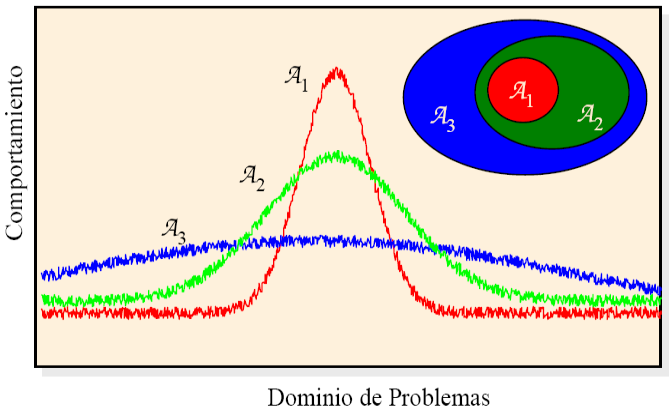

Figura 2.1: Teorema No Free Lunch.

2.1.2. Computación Evolutiva

La Computación Evolutiva [1] es un subconjunto dentro de los algoritmos bioinspirados y de las metaheurísticas. Son modelos de evolución basados en poblaciones cuyos individuos de la población van a representar soluciones a nuestro problema. Cuando simulamos estos modelos en un computador vamos a obtener algoritmos probabilísticos que mejorarán en muchas ocasiones a otras heurísticas y/o algoritmos.

Los hitos históricos más importantes dentro de los AAEE clásicos son los siguientes:

- En 1948, *Alan Turing* propone la *búsqueda genética o evolutiva.*
- En 1962, *Bremermann* investiga sobre la *optimización a través de la evolución y recombinación.*
- Dos años después, en 1964, *Rechenberg* introduce las **Estrategias de Evolución**.
- 1965, *Fogel, Owens* y *Walsh* proponen la **Programación Evolutiva** como una nueva técnica bioinspirada.
- En 1975, *Holland* presenta los **Algoritmos Genéticos**, sin duda alguna los hermanos mayores de la **Computación Evolutiva**.
- Ya en 1992, *Koza* propone una nueva técnica conocida como **Programación Genética**.

[1] También referenciada como Algoritmos Evolutivos o AAEE

Inspiración biólogica

Como su nombre indica, este área de la Inteligencia Artificial tiene su fundamento en la **Teoría de la Evolución de *Darwin*** [5], en la que se postulan las condiciones y características que han de darse en un proceso evolutivo natural:

- Tenemos una población de individuos. Los individuos de la población tienen la habilidad de reproducirse.
- Los individuos de la población no son todos iguales, sino que hay ciertas diferencias entre ellos, luego hay diversidad en la población.
- Algunas de estas diferencias que tienen los individuos van a hacer que unos tengan más posibilidades de permanecer en la población que otros.
- La evolución es un proceso que va a operar sobre los cromosomas de los individuos.
- La selección natural es el enlace entre los cromosomas y las estructuras decodificadas de los individuos.
- El proceso de reproducción es el punto en el cuál la evolución toma parte y actúa.
- La evolución biólogica no tiene memoria.

En estos principios se va a basar, biológicamente, la Computación Evolutiva, aunque como veremos más tarde algunos de ellos son modificados para mejorar el algoritmo. Por ejemplo, el **elitismo** es un mecanismo utilizado en los algoritmos evolutivos que va a ir en contra del principio de que la evolución no tiene memoria.

2.1.3. Tipos de Algoritmos Evolutivos

Como ya se ha mencionado anteriormente existen cuatro tipo principales de algoritmos evolutivos. A pesar de esta clasificación, estos algoritmos tienen unas características comunes. Todos los AAEE utilizan un proceso de aprendizaje colectivo para toda la población, los descendientes se generarán mediante procesos no determinísticos o aleatorios, tratando de modelar los procesos de **mutación** y **cruce** que se dan en la evolución natural en cualquier especie (la mutación será la autorreplicación errónea de los individuos, y el cruce el intercambio de material genético entre individuos). Otra característica común será la asignación de una medida de calidad que servirá de **adaptación** o ***fitness*** a cada individuo.

El funcionamiento básico de cualquier algoritmo evolutivo se basa en mantener una población de posibles soluciones al problema, realizar modificaciones sobre esa población de soluciones y seleccionar un número de individuos según su nivel de adaptación o *fitness*, que se mantendrán en generaciones futuras. La población de individuos irá evolucionando a lo largo de las distintas generaciones con un único objetivo: ir buscando mejores regiones

del espacio de búsqueda mediante esa modificación y selección para conseguir llegar a una solución cada vez mejor.

A continuación mostramos una tabla en la que se puede apreciar la metáfora de la Teoría de la Evolución en los algoritmos evolutivos:

Evolución Natural	**AAEE**
Individuo	Solución candidata
Población	Espacio de soluciones
Adaptación	Calidad de la solución
Entorno	Problema

Tabla 2.1: Metáfora en los Algoritmos Evolutivos

Por último en esta sección, vamos a realizar una breve reseña explicativa sobre cada una de las diferentes técnicas que existen dentro de la **Computación Evolutiva**, aparte de los **Algoritmos Genéticos** (AAGG) que trataremos mucho más profundamente en lo que queda de capítulo.

Estrategias de Evolución

Empiezan a desarrollarse en Alemania por *Rechenberg* y *Schwefel* [19], aplicándose sobre todo a problemas de optimización numérica. Son metaheurísticas eficientes, que utilizan una buena técnica de optimización para valores reales y tienen mucho desarrollo teórico. Su característica más importante es la autoadaptación de parámetros.

La primera variante fue llamada **EE-(1+1)** ya que se trabajaba con un padre y un descendiente por generación. Posteriormente se sustituyó por la **EE-(μ,λ)** con μ padres > 1 y λ descendientes > 1. La mutación se va a autoadaptar de acuerdo a la regla de *1/5 éxito*.

Posteriormente se introdujo la **EE-(μ+λ)**, cuyo único cambio con respecto a la **EE-(μ,λ)** estriba en la selección de la población final, ya que los μ individuos de la nueva generación se consiguen seleccionando los μ mejores entre los μ padres y los λ descendientes de la anterior generación.

Programación Evolutiva

Esta es la menos importante de todas las técnicas de **Computación Evolutiva** que estamos comentando, nace en EE.UU. en los 60 de la mano de *Fogel* [10]. Aplicada sobre todo a problemas de aprendizaje y de optimización, no utiliza operador de cruce y realizan una autoadaptación de los parámetros estándar.

Es virtualmente equivalente a las **Estrategias de Evolución**, aunque se diferencian en que esta técnica utiliza una selección determinística en vez de aleatoria, como lo hacían las

Estrategias de Evolución. Además, la **Programación Evolutiva** considera las estructuras abstractas de codificación como *especies* en vez de como *individuos*.

Programación Genética

Koza [12] ha sido el gran difusor de esta técnica evolutiva con cuna en Alemania. Se ha aplicado al campo del aprendizaje automático o *machine learning*, y a problemas de predicción y clasificación. Tienen capacidad para competir con las redes neuronales, necesitan poblaciones muy grandes y su proceso es lento.

La **Programación Genética** va a utilizar estructuras de representación no lineales como árboles, grafos etc... Además el tamaño de los cromosomas no va a ser fijo como en otros paradigmas evolutivos, pudiendo variar el árbol tanto en profundidad como en anchura.

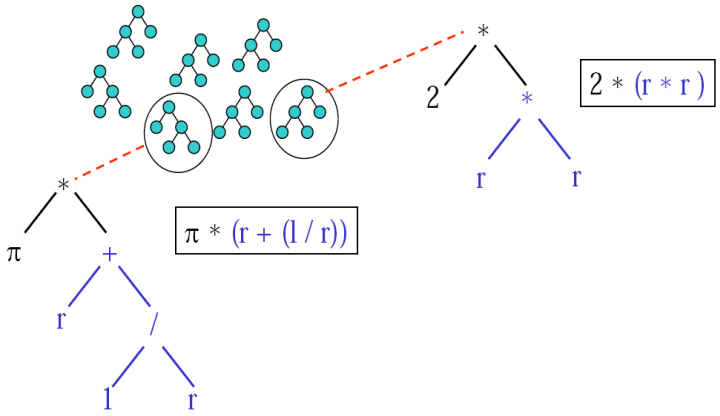

Figura 2.2: Dos representaciones de árbol en PG.

Un ejemplo de aplicación de la **Programación Genética** (PG) es la construcción de programas o expresiones (cada programa es un individuo de la población) que van mejorando con el paso de las generaciones.

El cruce se realiza intercambiando ramas de los árboles entre los dos padres implicados. El mismo proceso de cruce ya implica una introducción de diversidad, porque los descendientes suelen ser muy diferentes a los padres, luego en la mayoría de las ocasiones el operador de mutación no será necesario.

Una variación de la **Programación Genética** son los algoritmos **GA-P**, que son unos algoritmos híbridos entre los Algoritmos Genéticos y la Programación Genética. En estos GA-P, cada individuo va a constar de una expresión (programación genética) y una cadena de coeficientes (algoritmo genético), ambas partes evolucionarán simultáneamente. Los operadores de cruce y mutación se aplican de forma independiente en las componentes AAGG y PG.

2.2. Algoritmos Genéticos

En la última decada, los AAGG se han estado usando extensivamente para resolver problemas de búsqueda y de optimización en diversas áreas, como ciencia, entorno empresarial e ingeniería. La razón principal es su facilidad de comprensión, su gran aplicabilidad y su perspectiva global.

Desarrollados por *Goldberg* y sus alumnos [14], su descripción más importante fue dada por *Bäck* en su libro *Handbook on Evolutionary Computation* [1]. Tres revistas científicas internacionales se dedican hoy día a investigar sobre esta temática, destacando *Transactions on Evolutionary Computation* de la organización *IEEE*. Como ya sabemos, se basan en la Teoría de la Evolución y en la Genética.

Partiendo de una población normalmente aleatoria, los AAGG van a ir evolucionando dicha población, intercambiando información genética entre los individuos, mutando o cambiando cierta parte de ellos o seleccionando los más aptos para la siguiente generación (computacionalmente hablando, se seleccionarán las mejores soluciones para resolver el problema). Así, de cada generación van quedando los más aptos que pasan a formar la siguiente y a completarla con descendientes, repitiéndose el ciclo de vida hasta llegar a un criterio de parada elegido por el diseñador. La siguiente figura 2.3 muestra dicho ciclo:

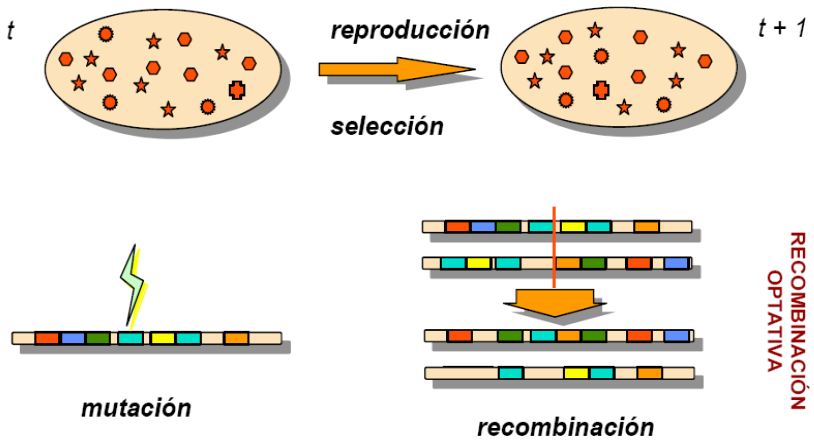

Figura 2.3: Elementos que forman un algoritmo genético.

En el siguiente apartado 2.2.1 iremos examinando uno a uno todos los elementos y operadores de los AAGG. La mayoría de estos conceptos también serán utilizados por los **Algoritmos Genéticos Multiobjetivo**.

Antes ilustraremos la estructura principal de cualquier Algoritmo Genético (algoritmo 2.1). Aunque esta estructura puede ser modificable dependiendo de si el algoritmo es o no estacionario, o de si se introducen nuevos conceptos evolutivos, la mayoría de los AAGG comparten dicha estructura.

Algoritmo 2.1 (Algoritmo Genético)

$t = 0$
Inicializar $P(t)$
Evaluar $P(t)$
mientras *no se cumpla condición de parada*
    $t = t + 1$
    Seleccionar $P(t)$ *desde* $P(t-1)$
    Recombinar $P(t)$
    Mutar $P(t)$
    Evaluar $P(t)$
fin mientras

2.2.1. Conceptos básicos

Representación

Ya sabemos que un individuo de la población va a ser una solución a nuestro problema. Deberemos representar la solución de tal forma que se ajuste a las restricciones del problema. Existen muchas formas de representar una misma solución. Así, notando N como el tamaño del cromosoma, los tres tipos de representación más importantes son:

- **Binaria**: es la más utilizada. Cada gen $i \in \{1, N\}$ del cromosoma podrá tener los valores discretos $x_i \in \{0, 1\}$.

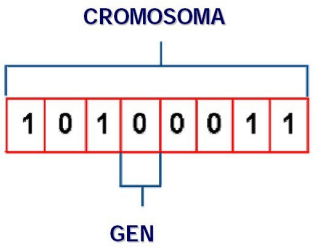

Figura 2.4: Genes de un cromosoma binario.

- **Entera**: Cada gen $i \in \{1, N\}$ del cromosoma podrá tomar los valores enteros $x_i \in \{0, n\}$, en dónde n es el máximo valor entero posible fijado por el diseñador del algoritmo.

  En algunos problemas como en el *Viajante de Comercio* los individuos van a representar permutaciones de sus elementos. Tendremos que realizar una representación de

la solución que garantice que al aplicar un operador, la permutación sigue existiendo en el cromosoma. Así aparece una representación entera particular a la que llamamos **representación de orden**.

- **Real**: en este tipo de representación los valores del cromosoma son números $x_i \in \Re$. Esta codificación hace que el campo de posibles soluciones sea mucho mayor al haber infinitos valores permitidos.

Independientemente del tipo de representación, la función de evaluación del cromosoma devolverá un valor $f \in \Re$.

Algunas veces la obtención del fenotipo (representación lógica del cromosoma) a partir del genotipo (solución real de nuestro problema) es un proceso obvio, otras veces será necesario un algoritmo de obtención del fenotipo a partir de los datos del problema.

Inicialización

El proceso de evolución natural es un proceso estocástico. De igual forma, los AAGG van a ser no determinísticos, cruzándose y mutándose de forma aleatoria. También suele ser una norma general inicializar la población de individuos de forma aleatoria, cumpliéndose siempre las restricciones de representación de la solución. Por ejemplo, en un problema con representación binaria, el proceso de inicialización de la población puede determinar si el gen toma el valor 0 o 1 con una probabilidad 0.5 en cada caso.

Otra opción es que la población inicial tenga una parte de los individuos creados aleatoriamente y otra parte a partir de heurísticas, para poder comenzar el proceso de evolución con individuos de una cierta calidad.

Evaluación de un individuo

En la evolución natural, los individuos más fuertes y más aptos para vivir en unas determinadas condiciones son los que sobreviven a lo largo de generaciones. En Computación Evolutiva ocurre igual: cada individuo va a tener unas habilidades o calidad que harán que tenga más opciones de permanecer en la población que otros individuos con menos calidad. A esta calidad se le llama ***fitness* de la solución**. Cuanto mejor *fitness* tenga una solución, mejor va a ser para nuestro problema.

En el ejemplo del *Viajante de Comercio* nos interesaría encontrar caminos lo más cortos posibles que pasen por todas las ciudades. Debido a esto, la calidad de la solución vendrá dada por la longitud del camino que recorre el viajante: a menor longitud de camino, más calidad.

Operador de cruce

Simula la reproducción que se da en las especies. A partir de varios padres se generan uno o varios descendientes, que van a heredar características genéticas de cada padre.

El cruce se debe diseñar de acuerdo a la representación utilizada en el cromosoma y al número de padres e hijos que entran en juego en el proceso. La recombinación de material genético de los padres debe generar hijos válidos dentro de las restricciones del problema.

Uno de los cruces más importantes es el **cruce OX** para representación binaria. A partir de dos padres obtenemos dos hijos de la siguiente forma: aleatoriamente seleccionamos un punto que esté comprendido entre 1 y el tamaño del cromosoma. Este punto de corte dividirá a los dos padres en dos partes, cada hijo estará formado por una parte del primer padre y por una parte del segundo padre (ver figura 2.5). De esta forma cada hijo va a tener información de los dos padres, y los hijos van a ser diferentes entre sí.

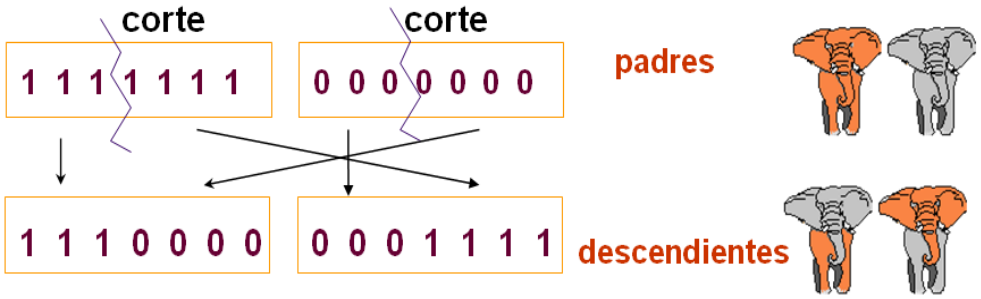

Figura 2.5: Cruce en un punto para representación binaria.

Para otras representaciones existen otras técnicas de cruce, pero todas ellas comparten una misma filosofía y propósito. Para cualquier cruce habrá que fijar también una probabilidad de cruce p_c que nos indicará cuantos elementos de la población tomarán parte en el proceso reproductivo.

Operador de mutación

Con el operador de mutación se consigue simular la modificación errónea de los genes de los individuos que se da en la Naturaleza. Se produce una pequeña modificación en los genes del individuo que hace que éste se diferencie de sus progenitores. Computacionalmente se suele cambiar el valor de un gen del vector que representa el cromosoma del individuo de forma aleatoria. Como es lógico, este cambio debe producir una nueva solución válida.

Se debe controlar el número de mutaciones que se realizan en cada generación de la población mediante una probabilidad de mutación p_m, similar a la probabilidad que existía para el cruce.

Como se ve en la tabla 2.2, la mutación para representaciones binarias es extremadamente trivial, aleatoriamente se elegirá un gen del cromosoma, en este caso el penúltimo gen, y se cambiará su valor inicial por su complementario.

0	1	0	0	**0**	1

0	1	0	0	**1**	1

Tabla 2.2: Ejemplo de mutación en un cromosoma binario.

Estrategia de selección

Se debe garantizar que los mejores individuos son los que tienen más posibilidades de permanecer en la población y de ser los padres de las nuevas generaciones frente a los individuos de peor calidad. Hay que ser cuidadoso con dar también la posibilidad de sobrevivir a individuos menos fuertes, ya que pueden introducir material genético valioso en el proceso de reproducción. Esta idea se conoce como **presión selectiva**, que determinará en qué grado la reproducción estará dirigida por los mejores individuos.

Vamos a comentar las dos estrategias de selección más utilizadas, que nos dirán qué individuos de la población serán los padres de las nuevas generaciones:

- **Selección proporcional**: dependiendo del fitness que tenga cada individuo $i \in \{1..N\}$ así va a ser su probabilidad de reproducirse. Cada individuo tendrá una probabilidad dada por $ps_i = \frac{f_i}{\sum_j f_j}$ siendo f_i el fitness del individuo i.
- **Selección por torneo**: escogemos aleatoriamente un número k de individuos, que tendrán que *luchar* entre ellos para ver cuál será el elegido. De entre los escogidos ganará aquel individuo que tenga una mejor calidad.

Estrategia de reemplazamiento

Otro aspecto que hay que tener en cuenta a la hora de diseñar un algoritmo genético es la estrategia de reemplazamiento a utilizar. Ya hemos visto que elegimos a los mejores individuos para reproducirse y mutarse y que éstos generan unos descendientes, pero a partir de estos padres y descendientes, ¿cómo formamos la nueva población? Tenemos varias opciones:

- Coger siempre a los descendientes para la nueva generación.
- Elegir de entre los padres y descendientes a los mejores.
- Realizar una especie de torneo entre los hijos y sus progenitores directos para ver cuáles son mejores, y con ellos formar la nueva población.

Aparte de estas opciones existe también un concepto que ya introducimos en secciones anteriores llamado **elitismo**, consistente en guardar siempre al mejor o mejores individuos

de la población. La razón es obvia: si hemos encontrado a un individuo muy bueno, !no hay que dejar que se pierda! Esto choca un poco contra la inspiración biólogica de los AAGG, ya que en la Naturaleza no se da ningún fenómeno parecido.

Modelo Generacional y de Estado Estacionario

A la hora de diseñar un AG podemos optar por dos enfoques distintos, el modelo **generacional** y el de **estado estacionario**.

En el modelo generacional, en cada iteración vamos a crear una población completa con nuevos individuos, bien a partir de los hijos de la anterior, de los padres e hijos etc... **La nueva población reemplazará directamente a la antigua**. Por contra, en el modelo de estado estacionario se escogen dos padres de la población en cada generación y se les aplican los operadores genéticos. **Los descendientes de esos dos padres reemplazarán a estos últimos**, formando la nueva población.

2.2.2. Diversidad y convergencia

Sin duda alguna son los factores más determinantes. Un AG, tanto monoobjetivo como multiobjetivo, que no tenga un *equilibrio entre diversidad y convergencia* está destinado al fracaso.

En cualquier algoritmo o metaheurística de búsqueda y optimización se debe establecer un equilibrio entre dos factores que aparentemente son contrapuestos: la **exploración del espacio de soluciones** y la **explotación del espacio de búsqueda**.

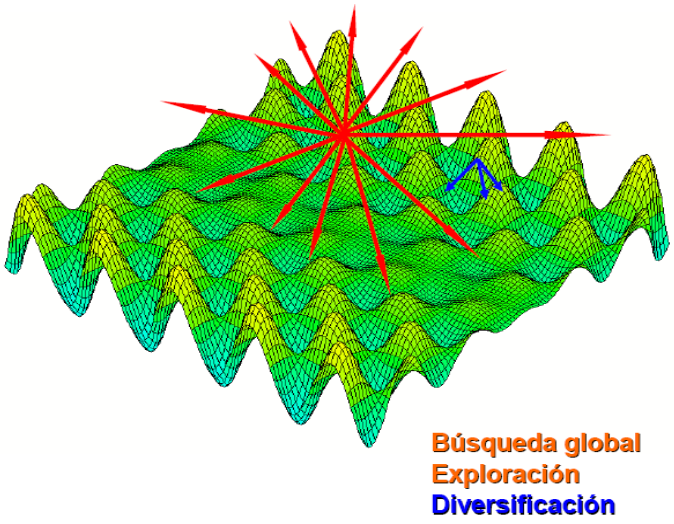

Figura 2.6: Ilustración de una búsqueda global.

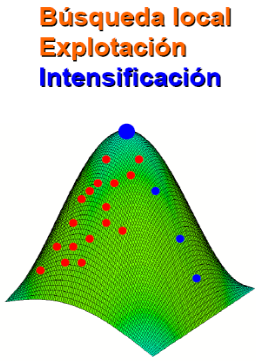

Figura 2.7: Ilustración de una búsqueda local.

La exploración del espacio de soluciones realiza una búsqueda global en amplitud (figura 2.6), encontrando de este modo las zonas más prometedoras. Por otro lado, la explotación del espacio de búsqueda hace una búsqueda en profundidad en zonas locales y prometedoras (figura 2.7), mejorando así las soluciones.

El modelo de estado estacionario produce una presión selectiva alta y una convergencia rápida ya que se están eliminando los peores individuos de la población por otros mejores. Sin embargo, genera una diversidad menor.

Con los AAGG podemos establecer un equilibrio adecuado entre esa exploración y explotación por medio de sus operadores. Trasladando este problema al marco evolutivo, nacen dos conceptos también contrapuestos:

- **Convergencia**: el algoritmo evolutivo centra la búsqueda en regiones prometedoras mediante la presión selectiva, seleccionando a los mejores individuos para reproducirse y olvidándose de los más diferentes. Demasiada convergencia en un AG puede ocasionar un grave problema como es la **convergencia prematura a óptimos locales**. Esto ocurre cuando sólo buscamos en una zona del espacio de soluciones y olvidamos otras, en dónde puede haber mejores soluciones a nuestro problema.

- **Diversidad**: introduciendo diversidad evitamos la convergencia prematura, ya que vamos a explorar otras zonas no visitadas por el algoritmo. La diversidad está asociada a las diferencias que hay entre los cromosomas de la población.

¿Cómo podemos evitar caer en óptimos locales sin dejar de lado la explotación de zonas prometedoras? Algunas propuestas dan solución a este problema:

- Diversidad en la mutación. Una probabilidad de mutación p_m alta no soluciona la convergencia prematura. La solución consiste en adaptar la probabilidad. Así para soluciones malas, la p_m deberá ser alta, mientras que para soluciones buenas la p_m deberá ser baja.

- Diversidad en el cruce. También se hace necesario introducir diversidad en el operador de cruce. Ésto se consigue prohibiendo el incesto, denegando el cruce entre individuos de la población que no superen una cierta distancia, o utilizando métodos de cruce que introducen gran diversidad como el **cruce HUX**.

- Separación espacial. Algoritmos que tienen una separación espacial como los **AAGG Celulares** o **AAGG Distribuidos**.

- Autoadaptación. Consiste en adaptar los parámetros necesarios en los AAGG a lo largo de la ejecución del algoritmo en función de su estado actual o información sobre el espacio de búsqueda. Se puede adaptar la función de evaluación, los operadores genéticos, representación etc...

- Estrategias de reemplazamiento. Cuando hablamos de estas estrategias ya comentamos que podrían aumentar o disminuir la **presión selectiva** del algoritmo. Diferentes métodos como los basados en similitud o en competición (p.ej. *Family Competition*) nos pueden ayudar en nuestra tarea.

Eshelman presentó un algoritmo llamado **CHC** [9] que obtiene un gran equilibrio entre diversidad y convergencia, cuyas características fundamentales son la introducción de un **nuevo cruce llamado HUX**, **prevención de incesto** y **reinicialización de la población** cuando se dan ciertas condiciones, todo ello para dar diversidad al algoritmo. Además utiliza una **selección elitista** para producir una convergencia elevada.

Utilizaremos este tipo de enfoque **CHC** en alguno de nuestros algoritmos, por lo que volverá a ser comentando más ampliamente en el Capítulo 3, en el cuál desarrollamos nuestra propia propuesta.

2.3. Optimización Evolutiva Multiobjetivo

Todos los AAGG anteriores tienen grandes limitaciones cuando hablamos de problemas reales. Casi todos los problemas a los que nos enfrentamos tienen más de una variable de decisión, por ejemplo problemas de climatización (coste, temperatura...), de control (fuerza motora, potencia, precisión del brazo del robot...), problemas financieros etc... Sólo hace falta pensar en un problema real y ver que no sólo va a depender de una variable, sino de varias más.

Problemas multiobjetivo. Frontera de Pareto

Matemáticamente, un problema que posee varias funciones objetivo a minimizar se representa de la siguiente forma:

$$minimizar[f_1(x), f_2(x), ..., f_k(x)] \text{ sujeto a ciertas } m \text{ condiciones } g_i(x) \qquad (2.1)$$

en dónde k será el número de funciones objetivo $f_i : \Re_n \longrightarrow \Re$. Llamaremos $x = [x_1, x_2, ...x_n]^T$ al vector formado por todas las variables de decisión.

Nuestro problema será determinar, de entre el conjunto F de todos los vectores de variables de decisión posibles que satisfacen las restricciones $g_i(x)$, los valores $x_1^*, x_2^*, ..., x_n^*$ que obtienen un valor óptimo para todas las funciones objetivo.

Es raro encontrar un único punto en el que se optimicen simultáneamente todas las funciones objetivo. Normalmente, en problemas multiobjetivo vamos a tener un conjunto de soluciones que van a ser relativamente buenas en todos los objetivos, y no una única solución. Por tanto, el término *optimalidad* va a cambiar de significado en este tipo de problemas. La teoría sobre esta noción de *optimalidad* que más se ha adoptado es la de *Vilfredo Pareto* en 1967, llamada ***Optimalidad de Pareto***.

Definición 6 *Diremos que un vector de variables de decisión $x^* \in F$ es una solución* ***pareto-optimal*** *o* ***no-dominada*** *si, y sólo si:*

$$\begin{array}{c} \forall i \in 1, 2, ..., k,\ f_i(x^*) \leq f_i(x) \\ \wedge \\ \exists j \in 1, 2, ..., k \ tal\ que\ f_j(x^*) < f_j(x) \end{array} \tag{2.2}$$

En otras palabras, **una solución domina a otra si es mejor o igual que ella en todos los objetivos, menos en uno, que sí es mejor. Todos los vectores de decisión que no son dominados por ninguna otra solución forman el llamado frente de Pareto** (ver figura 2.8).

Como ya hemos dicho, no suele existir una única solución optimal, sino un conjunto de soluciones óptimas, que son aquellas soluciones no-dominadas que forman el frente de Pareto.

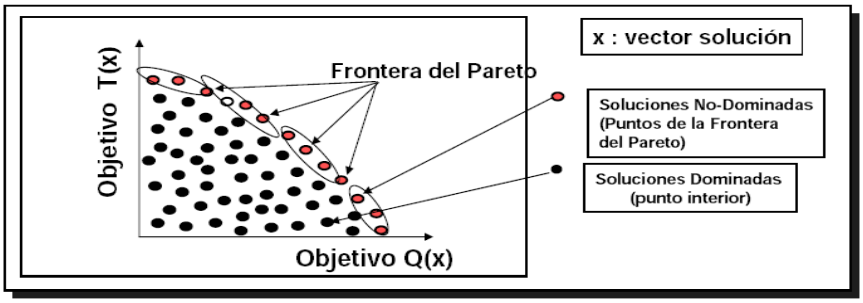

Figura 2.8: Composición del frente de Pareto.

Agregación de los objetivos mediante pesos

Todos estos problemas se pueden resolver utilizando AAGG tradicionales mono-objetivo. La *función fitness* asignaba a cada solución o individuo de la población un valor de calidad en

función de lo buena que era respecto a la variable de decisión. Luego la solución podría estar en agregar a la *función fitness* cada una de las funciones objetivo y ponderar la importancia de cada una. Así por ejemplo, si $T(x)$ y $S(x)$ son funciones objetivo del problema, la función *fitness* podría ser una función agregativa de ambas, ponderada por unos pesos de importancia tal que así: $F(x) = 0{,}75T(x) + 0{,}25S(x)$. La búsqueda de la solución se realizaría en una sóla dimensión (comprendería a las dos dimensiones reales) exactamente igual que en problemas mono-objetivo.

Este tipo de algoritmos se conocen como **modelos evolutivos que utilizan pesos para la agregación**, sin embargo, no son muy usados ya que su resultado es mejorado por los modelos que veremos a continuación y que utilizaremos en nuestro *Proyecto Fin de Carrera*, los **Algoritmos Genéticos Multiobjetivo** u **Optimización Evolutiva Multiobjetivo** (**EMO**), que son **modelos evolutivos que generan poblaciones de individuos no dominados**.

Optimización Evolutiva Multiobjetivo

Van a tener un único objetivo fundamental, alcanzar y abarcar todo el frente de Pareto como representa la figura 2.9, en dónde se hayan las buenas soluciones a un problema determinado.

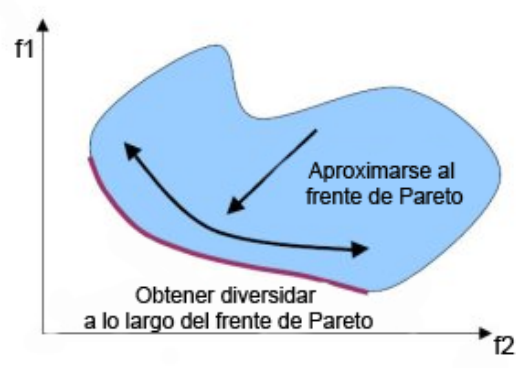

Figura 2.9: Objetivo de un algoritmo genético multiobjetivo.

Como muestra la figura, aparte de llegar a una solución del frente de Pareto hay que conseguir abarcarlo todo. En estos algoritmos también es importante un tema del que ya hemos disertado, la **convergencia** y la **diversidad**. Intentaremos ir convergiendo a soluciones cada vez más próximas al frente de Pareto, pero también tenemos que añadir diversidad al algoritmo para obtener soluciones de otra parte del frente. Para conseguir diversidad se suelen utilizar **funciones de densidad**, que premian a las soluciones que están en zonas menos exploradas con respecto a soluciones de zonas muy concurridas. Todos estos temas serán tratados hasta la finalización del capítulo.

2.4. Evolución y estado actual del arte

La primera implementación que fue reconocida como **EMO** fue la de *Schaffer*, llamada **VEGA**. Consistía básicamente en un AG simple con una modificación el mecanismo de selección. Se creaban un conjunto de subpoblaciones mediante una selección por torneo de acuerdo a cada uno de los objetivos. Para un problema de k objetivos se creaban k poblaciones de tamaño M/k. Finalmente se terminaba en una única población de tamaño M. Este algoritmo en realidad no producía valores buenos para una sola de las funciones objetivo, pero sí moderadamente buenos para todas ellas.

A mediados de los 80s y principios de los 90s algunos investigadores siguieron trabajando con AAGG multiobjetivo, pero la mayoría del trabajo realizado se centraba en modificar minimamente AAGG simples usando una función agregativa de objetivos.

Esta primera época de investigación no dejó buenos algoritmos y técnicas multiobjetivo, ya que se intentaba trabajar sobre los AAGG simples en vez de intentar crear una nueva técnica evolutiva. De todas maneras, es lógico que sucediera así ya que nos encontrábamos ante los primeros pasos de la Optimización Evolutiva Multiobjetivo.

2.4.1. Primera generación

El mayor avance en esta primera generación fue dado por *Goldberg*, que analizó el algoritmo VEGA [13] y propuso un esquema de selección basado en el concepto de optimalidad de Pareto. *Goldberg* sugirió el estándar en esta primera generación, que solía adoptar técnicas de *nichos* y de *fitness sharing*.

A continuación describimos brevemente los algoritmos más importantes de esta época.

Nondominated Sorting Genetic Algorithm

También conocido como NSGA, propuesto por *Srinivas* y *Deb* [20]. La población era clasificada en función del ranking de no-dominancia en el que se encontraban sus individuos. Para mantener la diversidad se utilizaban técnicas poco útiles de *fitness sharing*. El algoritmo permitía encontrar regiones no-dominadas, con cierta diversidad gracias al *sharing*.

Niched-Pareto Genetic Algorithm (NPGA)

Propuesto por *Horn* [15], utilizaba un esquema de selección también basado la dominancia de Pareto. La idea básica del algoritmo era elegir a dos individuos entre la población, y ver quién ganaba mediante el concepto de no-dominancia. Si ninguno de los dos dominaba al otro, el torneo se decidía por *fitness sharing*.

Multi-Objective Genetic Algorithm (MOGA)

Fue diseñado por los investigadores *Fonseca* y *Fleming* [11]. En este algoritmo, el puesto de un elemento de la población lo decide el número de individuos a los que domine. Todos los individuos no dominados tendrán un *ranking* de 1.

En conclusión, esta primera generación estuvo caracterizada por el uso de un mecanismo de selección basado en la teoría de Pareto y en el *fitness sharing*. Los primeros pasos sólidos en **EMO** se acababan de dar.

2.4.2. Segunda generación

Esta generación nació con el concepto de **elitismo**. En el área de los evolutivos multiobjetivo, elitismo se suele referir a una población externa en dónde se van almacenando a lo largo de las distintas generaciones las soluciones no-dominadas.

Strength Pareto Evolutionary Algorithm (SPEA)

Fue concebido como una agregación de diversos algoritmos multiobjetivo. Sus autores, *Zitzler* y *Thiele* [24], consideraron oportuno dotar al algoritmo de un archivo externo en dónde se fueran guardando las soluciones no-dominadas. Para cada individuo de la población se calculaba una medida llamada *strength*, similar al *ranking* del NSGA. Se utilizaba una técnica de *clustering* para asegurar la diversidad.

Pareto Archived Evolution Strategy (PAES)

Introducido por *Knowles* y *Corne* [16], consiste en una estrategia de evolución (1+1) junto a un archivo externo para las mejores soluciones encontradas. Otro aspecto interesante del PAES es la utilización de una *función de crowding* para mantener la diversidad, dividiendo recursivamente el espacio de objetivos. Cada solución es situada en una celda de la malla global.

Pareto Envelope-based Selection Algorithm (PESA)

Propuesto también por *Corne* [4], utiliza una división en celdas del espacio parecido al PAES, con otro archivo de población externo. También tenemos una medida de *crowding* que decide que soluciones van a pasar a formar parte de la población externa. Esta población externa juega un papel muy importante en este algoritmo ya que no sólo determina la diversidad, sino que también el esquema de selección. Existe una versión más reciente del algoritmo llamado PESA-II.

Micro Genetic Algorithm

Enfoque dado por *Coello* y *Toscano* [2] que utiliza una pequeña población y un proceso de reinicialización. La memoria de la población es dividida en dos trozos, la reemplazable, que cambia en cada iteración del algoritmo, y la no-reemplazable que no lo hace en toda la ejecución del mismo.

Hoy día nos encontramos en esta segunda generación que se ha basado en el énfasis de la eficiencia y en el elitismo. También se han desarrollado varias métricas y funciones de test para estos evolutivos. Los dos algoritmos más usados en las actualidad, pertenecientes a esta segunda generación, son el **NSGA-II** y **SPEA-2**, que merecen un estudio aparte.

2.5. NSGA-II y SPEA2 a fondo

2.5.1. NSGA-II de Deb

El algoritmo **NSGA-II** fue propuesto por *K. Deb* en el año 2000 y publicado en el *IEEE Transactions* en abril de 2002 [6]. Realmente, aunque su génesis esté en el algoritmo NSGA del mismo autor, son importantes las diferencias que existen entre ellos.

El algoritmo NSGA (*Nondominated Sorting Genetic Algorithm*) fue criticado por diversas razones, y es ahí en dónde el NSGA-II se diferencia de su progenitor. Las mayores críticas al NSGA se centraron en los siguientes aspectos:

- **Tiempo de computación demasiado alto en la ordenación de nodominancia**: para dicha ordenación por fronteras de nodominancia, que explicaremos más tarde, se obtenía una complejidad computacional de $O(MN^3)$ dónde M representa el número de objetivos del algoritmo y N el tamaño de la población.
- **Ausencia de elitismo**: como explicamos anteriormente, la 2ª generación de EMOs se basaron en el elitismo, demostrando su eficacia. NSGA no disponía de elitismo y se hizo necesario incluirlo para el NSGA-II.
- **Necesidad de especificar un parámetro** σ_{share}: este parámetro era necesario para utilizar el concepto de *sharing*, que proporcionaba al NSGA la herramienta para conseguir diversidad en la población. Con la nueva estrategia de diversidad utilizada en el NSGA-II no necesitamos esa especificación y se consigue mayor transparencia en el algoritmo.

Ya hemos adelantado ciertos conceptos en los que se basa el algoritmo multiobjetivo, podemos resumir brevemente las características del algoritmo para después profundizar más en él.

El NSGA-II es un algoritmo genético multiobjetivo que se basa en la **ordenación por fronteras de no-dominancia**. Ya sabemos cuándo una solución es no-dominada, y qué es

una frontera de Pareto. Así, esta ordenación consiste en ir asignando a cada solución un *ranking* que será igual al número de frontera en la que se encuentra. Por ejemplo, una solución que sólo es dominada por otra solución estará en la segunda frontera de no-dominancia, y tendrá un *ranking* igual a 1. Este *ranking* será el valor que tenga el *fitness* de cada solución. Como es lógico, estaremos ante un problema de minimización ya que se intentan conseguir soluciones que no estén dominadas por ninguna otra, es decir, minimizar ese *ranking* o *fitness*. Con este enfoque se consigue introducir elitismo, ya que en cada generación del algoritmo vamos a ir guardando siempre los mejores individuos, aquellos que están en la frontera de Pareto (*ranking* de no-dominancia = 0), o en las mejores fronteras posibles.

Otro concepto en el que se basa el NSGA-II es el de **distancia de *crowding*** (en español, distancia de apiñamiento). Sustituye al *sharing* que utilizaba el NSGA y va a conseguir introducir diversidad en las soluciones del problema. Cuando haya que decidir entre varias soluciones con parecidos valores de *fitness* o no-dominancia, se preferirá coger una solución que esté menos apiñada en el espacio de soluciones, esto es una solución más diferente a las demás y que permitirá explorar mejor el campo de soluciones.

Lazo principal del algoritmo

Como suele ocurrir en los AAGG, la primera población de individuos P_0 se obtiene aleatoriamente. A partir de ella podemos obtener la primera población de descendientes, notada por Q_0 con un tamaño N. Los descendientes se consiguen mediante una selección por torneo de *crowding* (que veremos unas secciones más adelante), cruzando y mutando convenientemente.

A partir de estas dos poblaciones iniciales podemos dar marcha al algoritmo, que tiene los siguientes pasos:

PASO 1 Juntamos las poblaciones de padres y descendientes, P_t y Q_t, en una población global llamada R_t. Utilizando el concepto de frentes o *ranking* de nodominancia vamos a ir agrupando los diferentes individuos en dichos frentes: F_1 para los individuos que forman el frente de Pareto (*ranking*=0), F_2 para aquellos con *ranking*=1, etc...

PASO 2 Comenzando con una población $P_{t+1} = \oslash$ también de tamaño N, vamos a ir completándola a partir de la población global R_t, frente a frente, hasta que uno de estos frentes no coja completamente en la población (mientras que $\mid P_{t+1} \mid + \mid F_i \mid <= N$, hacemos que $P_{t+1} = P_{t+1} \cup F_i$ y $i = i + 1$) (ver figura 2.10).

PASO 3 Si no hemos conseguido completar totalmente la nueva población P_{t+1} con frentes exactos, deberemos seleccionar los $N - \mid P_{t+1} \mid$ individuos necesarios del frente F_i. Para ello ordenaremos el frente utilizando el operador de *crowding* $<_c$.

PASO 4 En este paso tendremos ya completada la población P_{t+1}. Obtendremos la población de descendientes Q_{t+1} aplicando selección por torneo de *crowding*, mutación y cruce a P_{t+1}. Avanzamos una generación: $t = t + 1$.

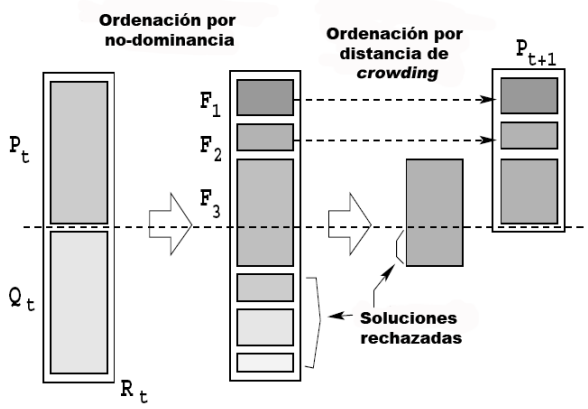

Figura 2.10: Proceso de selección por frentes de no-dominancia ([7]).

Ordenación rápida por no-dominancia

Al principio de la sección hablamos de que uno de los mayores problemas del NSGA era el alto orden de complejidad de su ordenación por no-dominancia ($O(MN_3)$). Este algoritmo mejora ese problema.

En una primera aproximación a la ordenación por no-dominancia podemos considerar un método válido el buscar, en una primera etapa, los individuos que son no-dominados (ninguna solución les domina y por tanto tendrán un *ranking*=0). En una segunda etapa descartamos los individuos anteriores y volvemos a buscar aquellos que no sean dominados (*ranking*=1) y así hasta que no queden más elementos que ordenar. Sin embargo, este método hasta cierto punto ingenuo tendrá una complejidad de $O(MN_3)$ donde M representa el número de objetivos y N el número de individuos de la población.

El NSGA-II destaca por su rápido proceso de ordenación que supera claramente a esta ordenación ingenua. Este se basa en que cada individuo p de la población tiene dos valores:

1. una cuenta de dominancia (n_p) que almacenará el número de soluciones que dominan a la solución p.

2. un conjunto de individuos (S_p) que serán aquellos dominados por la solución p.

El proceso de ordenación consistirá en ver los individuos cuya cuenta de dominancia n_p sea 0. Estos individuos formarán el frente de Pareto y tendrán un *ranking* de 0. Disminuiremos en una unidad las cuentas de dominancia de cada uno de los individuos pertenecientes a los conjuntos S_p dominados por estas soluciones de *ranking* 0, que separaremos en una nueva lista de individuos que serán aquellos de *ranking* 1. Con los individuos de los conjuntos dominados por esta nueva lista de *ranking* 1 haremos lo mismo y formaremos una nueva lista (*ranking* 2). Así hasta identificar todas los frentes. La complejidad final de este algoritmo es de $O(MN_2)$.

Distancia de *crowding*

Es el operador de densidad del algoritmo NSGA-II. Va a conseguir obtener las soluciones que están en zonas menos densas del campo de soluciones. Una distancia de *crowding* alta nos va a indicar que una solución está en una zona poco poblada, y una baja distancia nos dirá que hay alrededor otras soluciones parecidas a ella.

La distancia de *crowding* de una solución i en su frente va a estar definida por la media de los lados del cuboide que forman la solución de la derecha, $i+1$, y la solución más la izquierda, $i-1$ (ver figura 2.11). Si las soluciones $i-1$ e $i+1$ están muy juntas con respecto a la solución i, el perímetro del cuboide será muy pequeño y también lo será la distancia de *crowding*, denotando la densidad de la zona.

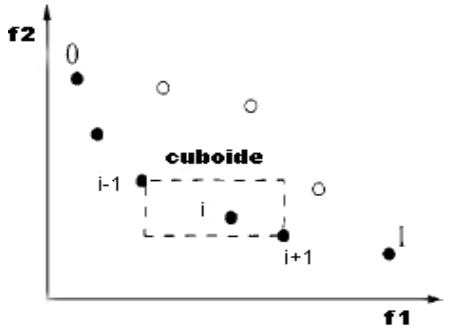

Figura 2.11: Cuboide necesario para el cálculo de la distancia de *crowding* ([7]).

El algoritmo que obtiene la distancia de *crowding* se detalla a continuación en tres pasos:

PASO 1 Partimos de un frente F con un número de individuos $\mid F \mid$. Para cada individuo i del frente inicializamos su distancia de *crowding* ($d_i = 0$).

PASO 2 Para cada objetivo $m = 1, 2..., K$ ordenamos el frente F en orden ascendente según su función objetivo f_m. Podemos notarlo como el vector de soluciones $I^m = ordenar(f_m, >)$.

PASO 3 Para cada objetivo $m = 1, 2..., K$ asignamos el valor ∞ a la distancia de las soluciones frontera (la solución 0 y $\mid F \mid -1$ del conjunto I^m). Para las demás soluciones no frontera, el valor de la distancia de *crowding* estará dado por:

$$d_{I_j^m} = d_{I_j^m} + \frac{f_m^{(I_{j+1}^m)} - f_m^{(I_{j-1}^m)}}{f_m^{max} - f_m^{min}}$$

Operador de *crowding*

La forma que tiene el algoritmo NSGA-II de preservar la diversidad es mediante el operador de *crowding* (apiñamiento) que ya hemos introducido líneas antes. Tiene la ventaja respecto a las funciones de *sharing* utilizadas en el NSGA de que no posee un parámetro σ_{share}.

Partimos de que cada individuo o solución i debe tener dos valores asociados: uno de ellos es el *ranking* de dominancia r_i y el otro es la distancia de *crowding* d_i. Podemos definir el operador de *crowding* $<_c$ de la siguiente forma:

Definición 7 *Una solución i gana el torneo de crowding a otra solución j, o simplemente es mejor solución que la otra solución, si se cumple al menos una de las siguientes condiciones:*

1. *La solución i tiene un menor ranking de dominancia r_i que la solución j.*
2. *Las soluciones i y j tienen igual ranking y además la solución i tiene mayor distancia de crowding que la solución j. Matemáticamente: $r_i = r_j \ \wedge \ d_i > d_j$.*

La justificación del operador de *crowding* se basa en que cuando tenemos que elegir entre dos soluciones, elegiremos primero a la que tenga menor *ranking* de dominancia. Pero en el caso de que estén en el mismo *ranking* o frente, elegiremos aquella solución que esté en una zona menos densa. De esta forma estamos introduciendo mayor diversidad en el algoritmo.

2.5.2. SPEA2 de Zitzler

Diseñado por *Zitzler*, *Laummans* y *Thiele* [22] un poco más tarde que su gran competidor el NSGA-II, es uno de los EMO de referencia actualmente. Nace para corregir ciertas debilidades de su predecesor SPEA, pareciéndose más a él que el algoritmo NSGA-II al suyo.

Los dos SPEA tienen iguales fundamentos. Hacen uso del elitismo mediante un archivo o población externa, en donde se van guardando las mejores soluciones, es decir, aquellas que están en el frente de Pareto.

La asignación del *fitness* se basa en los individuos dominados por cada solución. Recordemos que el NSGA-II utilizaba una ordenación por frentes de todos sus individuos.

Dos conceptos de selección se usan a lo largo de todo el algoritmo: la **selección de ambiente** (*environmental selection*) y la **selección para la reproducción** (*mating selection*). La primera de ellas se basa en seleccionar los individuos de la población principal que por pertenecer al frente de Pareto o ser los mejores individuos van a ser incluidos en la población externa para que no se pierdan en la siguiente generación.

La segunda selección se realiza sobre las dos poblaciones, tanto la principal como la externa. Sus individuos van a competir para reproducirse.

Las mayores diferencias que existen con su predecesor se muestran a continuación. Son rectificaciones que subsanan debilidades del SPEA original:

- Utiliza una nueva asignación del *fitness*, en dónde para cada individuo de la población se estima el número de soluciones que domina y que le dominan.
- Para intentar guiar de una forma más conveniente el proceso de búsqueda se ha incorporado una estimación de densidad del vecino más cercano (con objetivos similares al operador de *crowding* del NSGA-II).
- Se sustituye el algoritmo de *clustering* del SPEA por un truncado del archivo externo.
- Sólo los miembros de la población externa participan en la selección para la reproducción o *mating selection.*

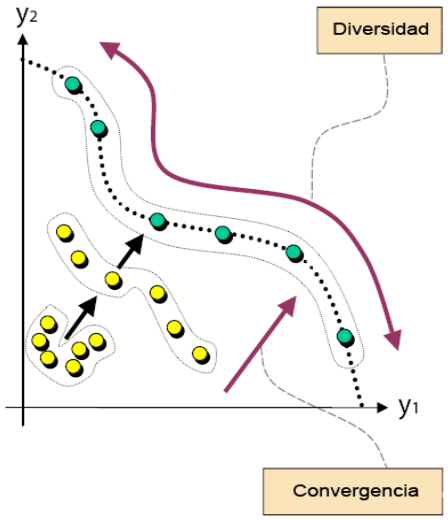

Figura 2.12: Objetivos que intenta cumplir el SPEA2 ([23]).

Lazo principal del algoritmo

También en este EMO la primera población de individuos P_0 se obtiene aleatoriamente. El archivo o población externa $\overline{P}_0$ empieza vacía. Los tamaños de ambas poblaciones son parámetros de entrada del algoritmo que notaremos como N y $\overline{N}$.

PASO 1 Generamos la población aleatoria inicial P_0. El archivo externo empieza vacío $\overline{P_0} = \oslash$.

PASO 2 Se asigna el fitness a los individuos de ambas poblaciones, P_t y $\overline{P}_t$.

PASO 3 Realizamos la **selección de entorno** descrita anteriormente. Copiaremos todos los elementos no-dominados (*fitness*=0) de ambas poblaciones P_t y $\overline{P}_t$ en la nueva población externa $\overline{P}_{t+1}$. Tenemos tres posibilidades:

- Si el número de individuos no-dominados de ambas poblaciones es igual al tamaño de la población externa $\overline{N}$, pasamos al siguiente paso.
- Si se excede el tamaño de la población externa habrá que eliminar elementos no-dominados utilizando un **operador de truncado** (ver sección posterior).
- Y si los individuos seleccionados no llegan a completar la población externa tendremos que **rellenar con los mejores individuos dominados** de P_t y $\overline{P}_t$.

PASO 4 Si hemos cumplido el criterio de parada, hemos terminado el algoritmo, devolviendo $\overline{P}_{t+1}$ como el conjunto de soluciones no-dominados de salida.

PASO 5 En este paso realizamos la **selección para la reproducción** utilizando un algoritmo de torneo binario con reemplazamiento sobre $\overline{P}_{t+1}$ para escoger a los padres.

PASO 6 Aplicamos recombinación y mutación clásica sobre los padres seleccionados en el **PASO 5**. La población descendiente formará la población general de la siguiente generación: P_{t+1}. Incrementamos el contador de generaciones y volvemos al **PASO 2**.

Asignación del *fitness*

Se introduce en SPEA2 un nuevo enfoque de asignación del *fitness* en el que cada individuo va a tener asociado tanto el número de soluciones que domina como el número de soluciones que le dominan. Así, para cada individuo i de la población P_t y del archivo externo $\overline{P}_t$ obtenemos un valor de fortaleza (*strength*) notado por $S(i)$ que representa las soluciones que domina:

$$S(i) = |\{i \mid j \in P_t + \overline{P}_t \wedge i \succ j\}| \tag{2.3}$$

$\succ$ representa la relación de no-dominancia entre dos soluciones. Basándonos en esta fortaleza se asocia a cada individuo un valor *raw* $R(i)$, calculado como sigue:

$$R(i) = \sum_{j \in P_t + \overline{P}_t, j \succ i} S(j) \tag{2.4}$$

El valor *raw* de un individuo i es igual a la suma de las fortalezas $S(j)$ de todas las soluciones que dominan a i. Si $R(i) = 0$ entonces no hay ninguna solución que domine a i, luego será una solución del frente de Pareto (ver figura 2.13).

Además del *raw* existe otra medida de la que depende el *fitness*, esta es la **información de densidad**, que sirve para poder discriminar entre elementos de la población con igual valor *raw* $R(i)$. Este valor nos recuerda a la distancia de *crowding* del NSGA-II. El cálculo del estimador de densidad es una variación del **kNN**, en dónde la densidad en un punto cualquiera es igual a la distancia a su k-esimo vecino más cercano. Se suele utilizar la inversa de la distancia al k-esimo vecino y un valor de $k = \sqrt{N + \overline{N}}$. Por tanto, la densidad de un individuo i vendrá dada por:

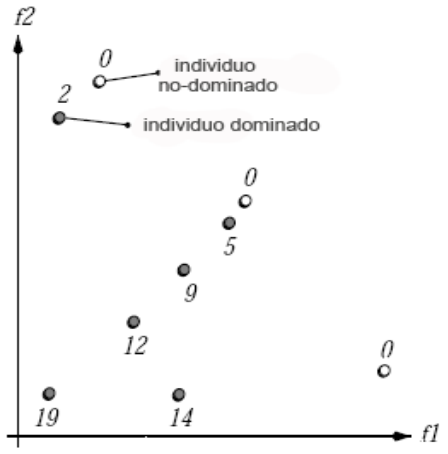

Figura 2.13: Valores *raw* de las soluciones en SPEA2 ([22]).

$$D(i) = \frac{1}{\sigma_i^k + 2} \tag{2.5}$$

Finalmente, el valor de *fitness* global de cada individuo i será una suma aritmética del valor *raw* y de la densidad:

$$F(i) = R(i) + D(i) \tag{2.6}$$

Seguimos teniendo un valor $F(i) < 1$ para soluciones no-dominadas pertenecientes al frente de Pareto. La asignación del *fitness* va a tener una complejidad de $O(M^2 log(M))$, donde $M = N + \overline{N}$.

Selección de ambiente

El concepto fundamental se basa en que en el archivo externo $\overline{P}_{t+1}$ se deben guardar las soluciones no-dominadas que encontremos tanto en P_t como en $\overline{P}_t$. Luego $\overline{P}_{t+1}$ estará formado por:

$$\overline{P}_{t+1} = \{i \mid i \in P_t + \overline{P}_t \ \wedge \ F(i) < 1\} \tag{2.7}$$

Habremos terminado la selección de ambiente si el tamaño de la población externa $\overline{N}$ es igual al número de individuos no-dominados. Pero esto casi nunca sucede y hay que, o bien eliminar elementos no-dominados, o bien añadir nuevos elementos dominados hasta completar $\overline{P}_{t+1}$. En el último caso, la solución consiste en coger los $\overline{N} - |\overline{P}_{t+1}|$ mejores elementos dominados del archivo externo y la población, y copiarlos al nuevo archivo. Lo podríamos implementar ordenando las poblaciones por *fitness* descendentemente cogiendo las mejores soluciones no-dominadas.

El otro caso es más complejo ya que hay que utilizar el operador de truncado del archivo para eliminar las soluciones que sobran. El proceso de truncado se realiza iterativamente

hasta que se consigue obtener un número de individuos igual al que necesitamos ($\overline{N}$). En cada iteración, un individuo i es eliminado de la población si $i \leq_d j \forall j \in \overline{P}_{t+1}$. Un individuo i es $\leq_d$ que j si se cumple que:

$$\forall\ 0 < k < |\overline{P}_{t+1}|\ :\ \sigma_i^k = \sigma_j^k$$
$$\vee$$
$$\exists\ 0 < k < |\overline{P}_{t+1}| : [(\forall\ 0 < l < k : \sigma_i^l = \sigma_j^l)\ \wedge\ \sigma_i^k < \sigma_j^k]$$

donde σ_i^k representa la distancia de i a su k-esimo vecino más cercano en $\overline{P}_{t+1}$.

El operador de truncado elige al individuo que tiene la mínima distancia a otro individuo de la población. Tiene un tiempo de computación, en el peor de los casos, de $O(M^3)$, con una media de $O(M^2 log M)$.

La figura 2.14 muestra un proceso simple de truncado en el que las soluciones 1, 2 y 3 son eliminadas del conjunto de soluciones por estar muy cerca de otras.

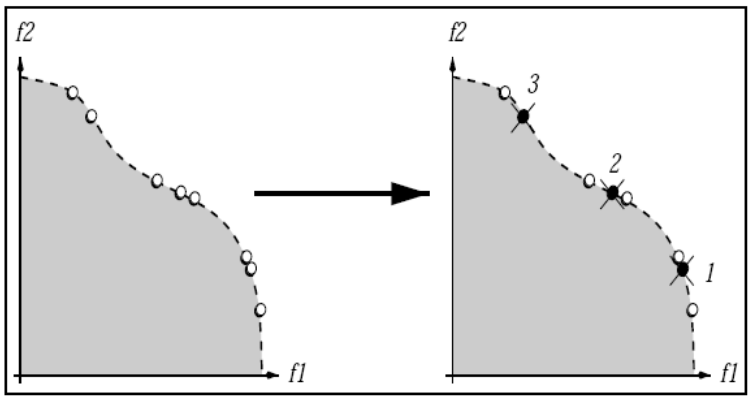

Figura 2.14: Ejemplo de proceso de truncado en SPEA2 ([22]).

2.5.3. Ventajas y desventajas de ambos algoritmos

Ya hemos descrito ampliamente el funcionamiento del algoritmo NSGA-II y del SPEA2. De todas formas recomendamos el libro *Multi-objective Optimization using Evolutionary Algorithm* [7] para un estudio aún mayor.

Aunque sin sobresalir demasiado, podemos afirmar que el algoritmo NSGA-II consigue mejores resultados, como media, que el SPEA2. Sus puntos fuertes son:

- Utiliza un buen método para generar diversidad en la población como es el operador de *crowding*.

- Es un algoritmo mucho más sencillo (a la hora de implementarlo y de comprenderlo) que su gran rival.

- A pesar de que tanto el SPEA2 como el NSGA-II tienen una complejidad $O(MN^2)$ en cada generación, el NSGA-II es más rápido empíricamente.

Por contra presenta algunas debilidades:

- La eficacia del operador de *crowding* va a depender del tamaño de la población. Si el número de soluciones del frente de Pareto no es muy grande, el operador de *crowding* no va a tener efectos sobre él. Además, puede que la distancia de *crowding* falle cuando hay más de 3 funciones objetivo.

- La ordenación por no-dominancia actúa sobre una población de tamaño $2N(Q_t + P_t)$, mientras que otros algoritmos semejantes sólo lo hacen sobre una población N.

Los puntos más importantes que hay que valorar en el diseño de un EMO son: la forma de conseguir avanzar hacia el frente de Pareto, y la obtención de diversidad mediante un estimador de densidades.

En el primer punto, tanto el NSGA-II como el SPEA2 tienen funcionamientos parecidos ya que ambos intentan ver, de acuerdo a las soluciones que dominan a otra o mediante *ranking* de frentes, qué soluciones serían las más aptas. Sin embargo, el operador de densidad del NSGA-II (operador de *crowded*) es realmente el punto más fuerte del algoritmo y se comporta mejor que el SPEA2 para un número pequeño de objetivos.

En la figura 2.15 mostramos tres tipos distintos de operadores de densidad. Tanto SPEA2 como NSGA-II utilizan una función de densidad dependiente de la esfera (segunda ilustración). Los otros dos enfoques pertenecen a otros dos algoritmos multiobjetivo, MOGA y PAES respectivamente.

La sencillez del algoritmo es algo que también presenta una clara ventaja respecto al SPEA2. Basta con intentar implementar el truncado de archivo externo del SPEA2 para darse cuenta de la complejidad que posee.

De todas maneras, el SPEA2 suele conseguir mejores resultados en cuanto a exploración del frente de Pareto que el NSGA-II en problemas complejos debido a su costoso operador de densidades. También presenta ventajas como la facilidad de cálculo de las medidas de *fitness*, o la rapidez con la que se consiguen soluciones del frente.

Una última desventaja del SPEA2 respecto al NSGA-II es el parámetro adicional que representa el tamaño de la poblacion externa y que puede afectar, según sea su valor, al desarrollo del algoritmo. Para finalizar comentar que el SPEA2 es más lento que su adversario, sobre todo por su operador de truncado, que tiene una alta complejidad de $O(M^3)$ siendo $M = N + \overline{N}$.

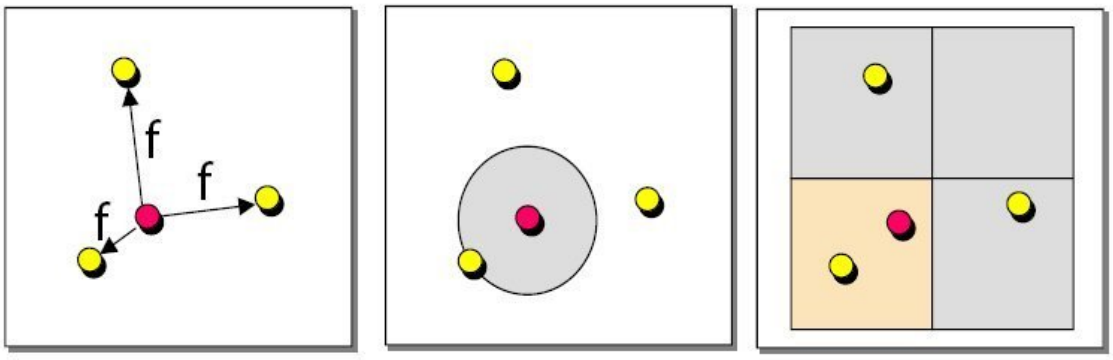

Figura 2.15: Distintos tipos de funciones de densidad ([23]).

2.6. Perspectivas futuras y aplicaciones

Sin duda alguna, los mejores algoritmos que hay en la actualidad son el **NSGA-II** y el **SPEA2** que acabamos de analizar. La popularidad del primero se basa sobre todo en su eficiencia en comparación con otros multiobjetivo. Consigue obtener un buen frente con sólo 5000 evaluaciones. La mayoría de algoritmos que intentan compararse con él tienen que intentar superarle con menos de 5000 evaluaciones y con codificación binaria (el NSGA-II obtiene mejores resultados con codificación real).

La Optimización Evolutiva Multiobjetivo lleva 20 años de estudio e investigación y se ha convertido en un área famosa en la que todavía se sigue investigando con asiduidad. Sin embargo parece que hay un cierto *parón* en cuanto a la realización de nuevos algoritmos. Los estudios se basan sobre todo en modificaciones al esquema de selección de los individuos mediante medidas estadísticas que discriminan entre unos y otros. También se han intentado publicar, no algoritmos nuevos, sino pequeñas mejoras a los dos EMOs más importantes. Sin embargo, la mejora suele ser mínima, y el algoritmo suele ganar mucha complejidad, por lo que no se utilizan.

La comunidad investigadora en EMO tiene como sueño construir un algoritmo que encuentre y defina totalmente el frente de Pareto sin un costo computacional tan alto. Pero por ahora no se ha conseguido. Las investigaciones se están centrando en mejorar ciertos aspectos tales como [3]:

- **Incorporación de preferencias** a los algoritmos para que no den un conjunto de soluciones del Pareto, sino la solución que más se amolda a esas preferencias.
- **Paralelismo** en EMO. Actualmente se está trabajando poco en este tema y se espera que su importancia vaya en aumento.
- **Restricciones en el campo de búsqueda** que ayuden a acotar el problema.
- **Uso de estructuras de datos más eficientes**.

2.6.1. Formas relajadas de dominancia: ϵ-dominancia

Un camino de estudio e investigación ya abierto, y que cada vez se está volviendo más popular es el tema que ocupa este epígrafe. En las formas relajadas de dominancia la idea básica es considerar **una solución x como mejor que una solución y aún si x es peor que y en algún (o algunos) objetivo(s)**. Tal deterioro debe compensarse con una mejora en el valor de los otros objetivos. Es una forma de aumentar diversidad en el algoritmo.

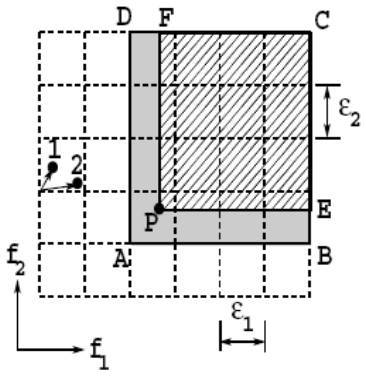

Figura 2.16: Esquema de funcionamiento de la *epsilon*-dominancia ([8]).

La forma relajada de dominancia más popular es la denominada ***dominancia*** ϵ [18]. El concepto de ***ϵ-dominancia*** requiere que el usuario defina la precisión que desea utilizar para evaluar cada función objetivo. Esta precisión o tolerancia ϵ define una rejilla en el espacio de las funciones objetivo del problema. Se diseña de tal forma que sólo se admite una solución por cada hipercubo (por ejemplo, si el problema es de minimización, sólo se acepta la solución no dominada más cercana a la esquina inferior izquierda de cada caja). Si se define un valor alto de ϵ se generarán pocas soluciones no dominadas, pero se logrará una rápida convergencia. Con un valor pequeño (el hipercubo será más pequeño) se consigue una mejor definición del frente pero con un mayor coste computacional.

La figura 2.16 muestra como funciona la ϵ-dominancia: con la dominancia tradicional, P domina al área PFCEP, mientras que con la ϵ-dominancia se domina a ADCBA. También en la figura 2.17 hacemos una comparación entre los dos tipos de dominancia.

Se define un nuevo concepto de dominancia basado en ese valor ϵ, y de frente de ϵ-dominancia:

Definición 8 *Sean f y g dos soluciones $\in \Re^{+m}$. Entonces diremos que cuando estemos en un problema de minimización, f ϵ-domina a g para algún $\epsilon > 0$, notado por $f \succ g$, si y solo si $\forall i \in 1, ..., m$:*

$$(1 + \epsilon) f_i \leq g_i$$

Definición 9 *Sea $F \in \Re^{+m}$ un conjunto de vectores de decisión del problema, y $\epsilon > 0$. Entonces diremos que F_ϵ es un frente de Pareto ϵ-aproximado de F, si cualquier vector*

$g \in F$ es ϵ-dominado por al menos un vector de $f \in F_\epsilon$:

$$\forall g \in F : \exists f \in F_\epsilon \text{ tal que } f \succeq_\epsilon g$$

El conjunto de todos los frentes ϵ-aproximados de f se denota por $P_\epsilon(F)$.

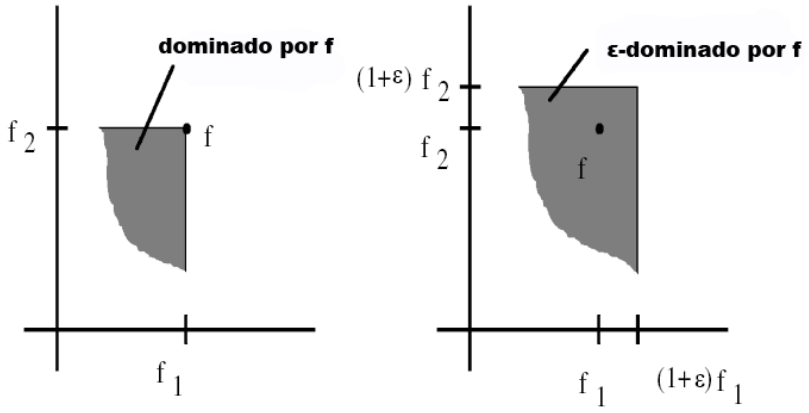

Figura 2.17: Comparación entre dominancia tradicional y ϵ-dominancia ([18]).

El autor **Coello** cree que quizás sea en esta técnica en dónde resida el futuro o 3º generación de Algoritmos Genéticos Multiobjetivo.

2.6.2. Algoritmo ϵ-MOEA de Deb

Debido a la importancia que está tomando este nuevo enfoque teníamos que entrar en profundidad y valorar la posibilidad de añadir un EMO de este tipo al sistema de Minería de Datos que vamos a desarrollar. En la bibliografía existen distintos algoritmos importantes que hacen uso del concepto de ϵ-dominancia, como el ϵ-NSGA-II [17] o el que vamos a examinar, el ϵ-MOEA propuesto por el indio *Deb* [8]. Hemos decidido utilizar este último, y no el primero, porque el ϵ-NSGA-II es una pequeña variación del ya estudiado NSGA-II. Además, utiliza una adaptación de los paramétros, técnica que se escaparía del objetivo de estudio de la ϵ-dominancia.

El algoritmo ϵ-MOEA tiene un esquema bastante simple en comparación con otros algoritmos EMO que hemos examinado. Va a utilizar dos poblaciones, $P(t)$ y $E(t)$ (para preservar el elitismo), y va a ir eligiendo un individuo de cada población para obtener los descendientes. Como es de preveer, utiliza la ϵ-dominancia para mantener la diversidad. El lazo principal es el siguiente:

PASO 1 Inicialmente se genera la población ordinaria $P(0)$ aleatoriamente. En el archivo $E(0)$ se volcarán las soluciones ϵ-no-dominadas que haya en $P(0)$.

PASO 2 Dos soluciones, p y e, una de cada población, se escogerán para generar a uno o más descendientes.

a) Para obtener el individuo de la población $P(t)$ podemos seguir varios esquemas. Vamos a optar por coger aleatoriamente dos miembros de la misma y ver quién domina a quién (en el caso de que nadie domine a nadie lo elegimos aletoriamente). Este individuo será nuestro primer padre, p.

b) El elemento de la población no-dominada $E(t)$ se elegirá aleatoriamente. Será el segundo padre, e.

PASO 3 Obtendremos λ descendientes a partir de estos dos padres p y e. En el artículo de *Deb* [8] se obtiene sólo un descendiente ($\lambda = 1$). Utilizaremos un cruce típico según la representación de las soluciones. En este algoritmo no hay mutación.

PASO 4 Para incluir a los nuevos individuos en las poblaciones habrá que ver si son aptos para ingresar tanto en la población ordinaria $P(t)$ como en el archivo $E(t)$:

a) **Inserción en el archivo no-dominado:** el descendiente c_i será comparado con los miembros del archivo según su relación de ϵ-dominancia (ver proceso en detalle más adelante).

b) **Inserción en la población ordinaria:** la decisión de incluir o no al descendiente en esta población se realiza por dominancia tradicional. Si el nuevo individuo domina a uno o más elementos de la población, este reemplazará a uno de ellos aleatoriamente. Mientras que si no domina a ninguno, el nuevo elemento no será incluido.

PASO 5 Si no se ha cumplido la condición de parada deseada en el algoritmo volvemos al **PASO 2**. Si ya hemos llegado al criterio de parada, los elementos del archivo $E(t)$ serán las soluciones de salida del algoritmo.

El tamaño de la población ordinaria no variará a lo largo de las generaciones, siendo un parámetro definido por el usuario. Por el contrario, no es necesario definir un límite para el archivo de población ya que su valor dependerá del ϵ elegido, a mayor epsilon, menor tamaño va a tener el frente y viceversa.

El proceso de inserción del descendiente en el archivo $E(t)$ es la parte más compleja del algoritmo. Cada solución del archivo vendrá definida por un vector $B = (B_1, B_2, ..., B_M)^T$, en dónde M es el número de objetivos, y cada valor del vector se obtiene como sigue:

$$\begin{cases} B_j(f) = & \lfloor (f_j - f_j^{min})/\epsilon_j \rfloor, \text{ si minimizamos } f_j, \\ & \lceil (f_j - f_j^{min})/\epsilon_j \rceil, \text{ si maximizamos } f_j, \end{cases}$$

dónde f_j^{min} es el mínimo valor posible del j-esimo objetivo, y ϵ_j es la tolerancia permitida en el j-esimo objetivo (igual al parámetro ϵ). El array va a dividir el espacio de objetivos en hipercajas cada una de tamaño ϵ_j en el j-esimo objetivo. El vector de identificación será construido tanto para cada elemento a del archivo como para el descendiente c_i.

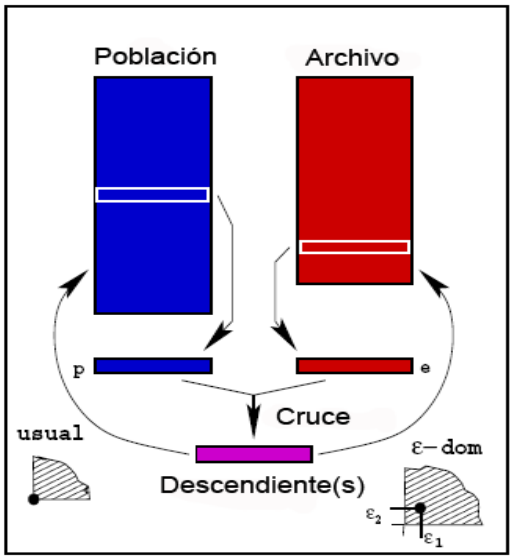

Figura 2.18: Estructura del algoritmo ϵ-MOEA ([8]).

- Si el vector B_a de algún individuo del archivo domina al vector del descendiente c_i será porque el descendiente es ϵ-dominado por ese miembro del archivo, por lo que no será incluido.
- Por contra, si el vector B_{c_i} del descendiente domina al vector B_a de algún elemento a del archivo, el elemento del archivo será eliminado y en su lugar incluiremos al descendiente.
- Si no se cumple ninguna de las dos premisas anteriores habrá que tener en cuenta si el descendiente comparte vector B con algún miembro del archivo:
  - El compartir vector con un miembro del archivo nos da a entender que están en el mismo hipercubo del espacio de búsqueda. Llegados a este punto habrá que quedarse con un sólo elemento en la caja. Nos quedaremos con el que domine por el método tradicional al otro. Si ninguno domina al otro, elegiremos como representante de la caja al que esté más cerca del vector B (menos distancia euclídea).
  - En el caso de que el descendiente c_i no comparta vector B con ningún miembro del archivo, dicho c_i será incluido en el archivo.

Todo el proceso anterior garantiza que sólo va a haber una solución por hipercubo, y sólo van a estar en el archivo los hipercubos pertenecientes al frente. El algoritmo se irá repitiendo hasta que se cumpla el criterio de parada. Los miembros del archivo serán los devueltos por el algoritmo como soluciones óptimas al problema. Las propiedades más importantes del ϵ-MOEA son [8]:

1. Es un algoritmo que presenta un gran equilibrio convergencia-diversidad.
2. Pone especial énfasis en las soluciones no-dominadas.
3. Mantiene la diversidad en el archivo permitiendo que sólo una solución esté en cada caja del frente de Pareto.
4. Utiliza un enfoque elitista.
5. Es mucho más rápido que otros EMO como NSGA-II o SPEA2.

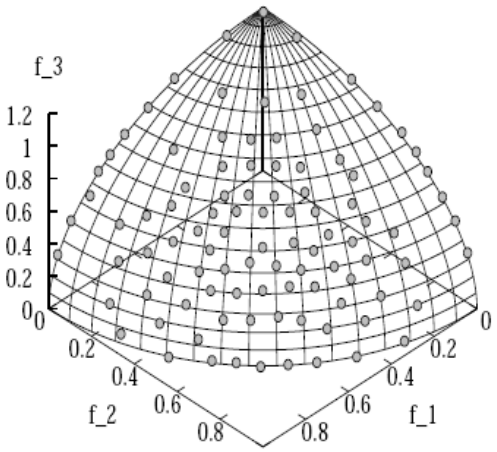

Figura 2.19: Frente resultante de una ejecución del ϵMOEA ([8]).

2.6.3. Aplicaciones

Las aplicaciones actuales que están teniendo los EMO se pueden dividir en tres grandes grupos principales: ingenieriles, industriales y científicas [3].

La literatura más interesante está dentro del primer grupo, las aplicaciones para ingeniería. No es de extrañar, ya que estas aplicaciones necesitan muy a menudo modelos matemáticos muy complejos con muchas variables que tener en cuenta.

- **Aplicaciones ingenieriles:** Ingeniería eléctrica, hidráulica y estructural, ingeniería aeronáutica, robótica y control, telecomunicaciones, e ingeniería civil y de transportes.
- **Aplicaciones industriales:** Diseño y manufacturado, monitorización y gestión.
- **Aplicaciones científicas:** Química, Física, Medicina y Ciencias de la Computación.

Capítulo 3

AAEE para Minería de Datos en problemas de alta dimensionalidad

3.1. Introducción

Los dos objetivos fundamentales que nos marcamos para nuestro Sistema de Minería de Datos fueron:

- El diseño de un sistema de clasificación que lograra mejorar la precisión de esta tarea predictiva con problemas complejos y de gran dimensionalidad.
- La aplicación de técnicas de selección de características que consigan facilitar la labor de dicho sistema.

En cuanto a técnicas de selección de características existen multitud de propuestas de algoritmos que a partir de un conjunto inicial de variables obtienen un conjunto más reducido de las mismas. Algunos de ellos ya los hemos visto detenidamente en el Capítulo 1, como **Las Vegas Filter** y **Las Vegas Wrapper**, **FOCUS**, **Relief** o el algoritmo **GRASP** de **Battiti**. Además, se pueden diseñar propuestas de algoritmos de selección de características basados en AAGG con codificación binaria o entera.

La limitación de todos estos algoritmos es que proporcionan un único conjunto reducido de variables a partir del conjunto inicial. Para un problema de clasificación concreto puede ser útil obtener no sólo un subconjunto de variables, sino los mejores subconjuntos de variables, de forma que el usuario, si así lo desea, pueda elegir entre uno u otro para diseñar el clasificador, o que con todos ellos se pueda diseñar un multiclasificador que mejore los resultados de los clasificadores individuales y que posibilite el uso del mismo independientemente de las variables que tengamos para una determinada instancia.

Para conseguir este fin aplicaremos los EMO más apropiados, obteniendo los mejores conjuntos de características que existan para el problema dado, es decir, el frente de Pareto del problema.

Teniendo en cuenta que vamos a tener diferentes subconjuntos de características nos vemos en la necesidad de desarrollar un clasificador por subconjunto y unirlos formando un **multiclasificador**, que será el modelo que utilizará nuestro sistema de clasificación. Se diseñarán varios clasificadores, y estudiaremos la mejor forma de mezclar o fusionar esos clasificadores en uno final, en un multiclasificador.

El aprendizaje del multiclasificador lo realizará un AG que detallaremos en este mismo capítulo. El esquema de todo el sistema final se representa en la figura 3.1.

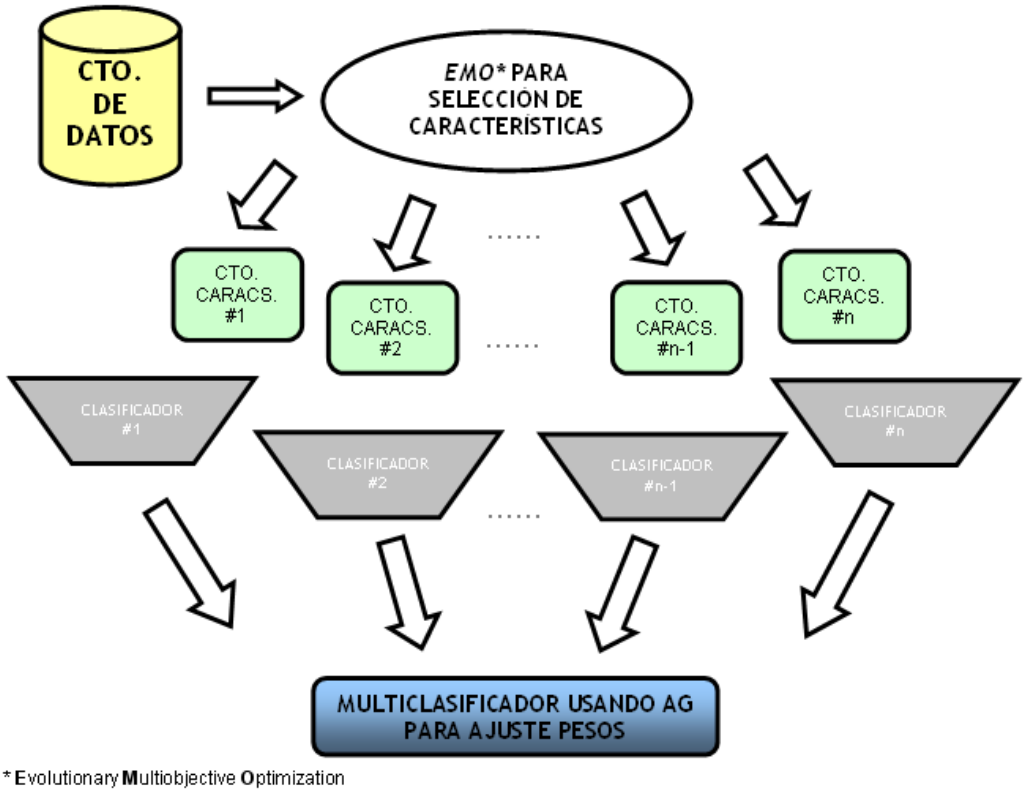

Figura 3.1: Esquema general del sistema de Minería de Datos.

La organización del capítulo es la siguiente:

- En la sección 3.2 describiremos la aplicación de la Optimización Evolutiva Multiobjetivo al problema de la selección de características. Para ello, en la subsección 3.2.1 describiremos el diseño de una versión de los mejores algoritmos multiobjetivo existentes para el problema de la selección de características. Y en la sección 3.2.2 se explicará una hibridación de los algoritmos multiobjetivo anteriores para intentar una mejor adaptación al problema de la selección de características, mejorando el comportamiento de los mismos.

- Por último, se explicará el desarrollo y funcionamiento del multiclasificador y de las características de los AAGG implementados que aprenden su configuración y parámetros (sección 3.3).

3.2. EMO aplicada a la Selección de Características

En la bibliografía especializada hemos encontrado ciertas referencias y trabajos realizados sobre la aplicación de AAGG al problema de la selección de características, sin embargo, es algo totalmente novedoso la aplicación de AAGG Multiobjetivo a tal fin.

La principal bondad de esta aplicación estriba en que estos algoritmos devuelven varios subconjuntos de características tal y como muestra la ilustración 3.2. Pero además de esto deberemos buscar un algoritmo de selección de características que genere subconjuntos que aumenten la precisión del clasificador, y que sean subconjuntos con una dimensionalidad baja.

Esta parte del sistema es fundamental ya que todo el proceso de clasificación dependerá de los subconjuntos de características que se hayan seleccionado.

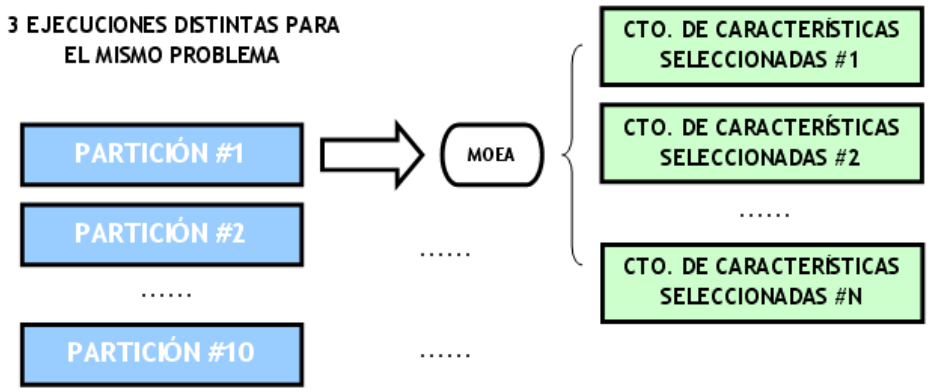

Figura 3.2: Aplicación de los EMO a la selección de características.

De acuerdo a lo estudiado en las secciones 5 y 6 del capítulo 2, los mejores algoritmos multiobjetivo que podemos aplicar a nuestro problema eran el NSGA-II, SPEA2 y ϵ-MOEA. En la subsección 3.2.1 se describen las adaptaciones de dichos algoritmos para resolver el problema de la selección de características, y en la 3.2.2 mejoras sobre los mismos.

3.2.1. Algoritmos Genéticos Multiobjetivo estándar

El primer paso que se debe analizar en el diseño de un EMO es el de establecer el número de objetivos y ver cuáles van a ser cada uno de ellos. Para que el problema sea realmente un

problema multiobjetivo, dichos objetivos tienen que ser independientes entre sí.

En la primera aproximación que hicimos al problema consideramos tres objetivos, todos ellos para ser minimizados, que eran los siguientes:

- **Objetivo 1:** Error de clasificación usando un clasificador. Hemos elegido el **kNN** por su escaso tiempo de aprendizaje y buenos resultados, aunque se podrían haber utilizado otros modelos como *reglas difusas* o *nítidas*, *árboles de decisión* etc...
- **Objetivo 2:** El porcentaje de características seleccionadas en la solución.
- **Objetivo 3:** Una medida de separabilidad de clases, en particular el ratio de inconsistencias.

La representación sería binaria y el tamaño de cada individuo sería igual al número de características del problema. Cada gen indicaría si la i-esima característica se había seleccionado mediante un 1. En caso de no haber sido seleccionada el valor del gen del cromosoma sería de 0.

Sin embargo, pronto descubrimos que estos tres objetivos no serían adecuados ya que al establecer como objetivo el número de características, ciertas soluciones poco válidas serían soluciones no dominadas. Esto es, cualquier solución con 1 característica seleccionada sería la mejor para el Objetivo 2, y por lo tanto, aunque su precisión y ratio de inconsistencia fuesen pésimos, sería seleccionada como una solución del frente de Pareto.

Consideramos que sería más conveniente tomar únicamente como objetivos el **error de clasificación** y el **ratio de inconsistencias**, utilizando el número de características seleccionadas como valor de ayuda en los mecanismos de diversidad de los algoritmos multiobjetivo.

De esta manera, para el algoritmo NSGA-II, que como ya sabemos utiliza la *distancia de crowding* como función de densidad para otorgar diversidad al frente de Pareto podríamos ponderar dicha distancia y darle más preferencia a aquellas soluciones que tuvieran menos características.

El SPEA2, en vez de utilizar la *distancia de crowding*, hacía uso de una *función de densidad* basada en el concepto de vecinos más cercanos. De igual forma, podríamos modificar dicha función de densidad para premiar a las soluciones con menos características.

El algoritmo ϵMOEA no tiene ningún mecanismo de diversidad parecido ya que sólo se basa en el concepto de ϵ-dominancia. Perfilando un modelo estacionario genera dos descendientes por generación, reemplazando a aquellos peores elementos de cada población. Por lo que optamos únicamente por dejar los dos objetivos indicados sin hacer uso del número de características.

Luego finalmente, nuestros **EMO estándar** tendrán 2 objetivos, con una representación binaria y con modificaciones a los mecanismos de diversidad del frente de Pareto.

Tras analizar los algoritmos y realizar unas primeras ejecuciones de prueba nos dimos cuenta de sus posibles limitaciones:

- **Falta de diversidad**, muy importante en el problema de la selección de características. Además, uno de nuestros objetivos era el de devolver subconjuntos de características que fueran distintos entre sí, y si no existe mucha diversidad en el algoritmo, los subconjuntos generados van a ser muy parecidos.

- **Conjuntos con demasiadas características**. Tenemos que intentar favorecer la creación de subconjuntos con menos características.

La solución podría estar en diseñar algoritmos multiobjetivo que incorporen componentes que potencien, sobre todo, la diversidad. Nuestras propuestas en este sentido se describen en la siguiente subsección.

3.2.2. Algoritmos Genéticos Multiobjetivo híbridos

En estos EMO híbridos (también referidos como **NSGA-II híbrido**, **SPEA2 híbrido** y **ϵMOEA híbrido**) incluimos ciertas modificaciones basadas sobre todo en el algoritmo CHC para otorgar diversidad al algoritmo y en la modificación de los dos objetivos del problema.

- Introducción de diversidad. Cuando estudiamos el problema pensamos que uno de los mayores inconvenientes que podrían tener los algoritmos **EMO estándar** era la diversidad, y vimos oportuno utilizar el esquema del algoritmo CHC de *Eshelman* [9]. Dicho algoritmo tiene los siguientes puntos importantes:

  - **Cruce HUX**: utiliza un cruce que crea descendientes bastante distintos a los padres. Su origen viene del *uniform crossover* (UX), que cambiaba los bits de los padres con una probabilidad fija p. Este cruce HUX es una versión modificada que cruza exactamente la mitad de los alelos distintos en los padres, en donde los bits intercambiados son elegidos aleatoriamente sin reemplazamiento. Garantiza que los descendientes siempre tengan la máxima distancia de *Hamming* con sus padres.
  - **Prevención de incesto**: cuando los padres seleccionados para reproducirse no son lo suficientemente distintos entre ellos se desechan y no se producen los hijos, volviéndose a elegir nuevos padres.
  - ***Restart***: si la población converge prematuramente en óptimos locales se le aplica un proceso de reinicialización. Se dejan a los mejores elementos encontrados hasta el momento y se crean aleatoriamente otros individuos a partir de los mejores.
  - **Ratio de divergencia:** Para la creación de los individuos aleatorios se usa un valor llamado **ratio de divergencia** que determina hasta qué punto deben tener más parte aleatoria o más parte proveniente de los mejores individuos. Este ratio también es utilizado en el proceso de *restart*.

Pensamos también que uno de los problemas que podría causar cierta convergencia prematura era el que en los archivos externos o poblaciones de individuos no-dominados los individuos eran muy parecidos desde generaciones tempranas, lo que hacía que siempre se estuviera explorando el mismo espacio de soluciones. La decisión que adoptamos fue la de **no permitir soluciones idénticas respecto al genotipo tanto en los archivos externos de los algoritmos, como en la población general de aquellos que carecieran de estos**. Así, aunque dieramos más oportunidades a los individuos menos buenos, estos nos podrían dar información de otra zona del espacio de soluciones y conseguir explorar más convenientemente. Este cambio no afecta a los mejores individuos porque seguirán en la población y no serán desplazados. Los tres EMO elegidos usan una metodología elitista, por lo que no existe el riesgo de que las mejores soluciones queden desplazadas por las peores.

- Cambios en los objetivos. La modificación que realizamos a los esquemas de diversidad de los algoritmos como *distancia de crowding* y *estimadores de densidad* podrían no afectar demasiado a las soluciones obtenidas ni al devenir del algoritmo, por lo que hemos considerado que para esta otra versión evolutiva sería interesante cambiar el esquema. De esta manera eliminaremos la ponderación en *distancia de crowding* y *estimadores de densidad* y cambiaremos los objetivos para que estos tengan en cuenta el número de características seleccionadas: **en vez de considerar únicamente el error de precisión como objetivo, podríamos ponderar con un 70 % y 30 % tanto el error como el número de características**:

$$objetivo_1 = 0{,}7 \times error_{precision} + 0{,}3 \times ratio_{caracteristicas}$$

Así conseguiremos soluciones no-dominadas que no sólo estén dadas por la precisión, sino obtener también soluciones con pocas características.

Como cada uno de los tres algoritmos multiobjetivo tienen unas peculiaridades determinadas, vamos a analizar uno por uno los cambios introducidos en ellos.

NSGA-II híbrido

Los cambios respecto a la versión estándar son los siguientes:

- Consideramos dos objetivos, uno de ellos una ponderación entre precisión y número de características. El otro objetivo, el ratio de inconsistencias.
- Se elimina la participación del número de características en la *distancia de crowding* por tener poca repercusión.
- No se permiten individuos iguales en la población, así tenemos más diversidad (también hay individuos menos buenos que pueden empobrecer la solución final) y podemos generar individuos más distintos.

- *Cruce HUX*, prevención de incesto y *restart* de la población tomando a los mejores individuos como plantilla.

Cuando no se reemplaza ningún descendiente en la población (en cada generación habrá un número variable de ellos), el umbral va decrementándose hasta que llegue a ser menor que cero, momento en el cuál se produce una reinicialización tal como sigue:

- Se copian en la nueva población los elementos no-dominados.
- El resto de elementos hasta completarla se crean aleatoriamente usando como plantilla a los elementos no-dominados. Existirá un parámetro llamado *ratio de divergencia*, especificado por el usuario, el cuál determinará la cantidad de genes totalmente aleatorios o copiados del elemento usado como plantilla.
- El valor de umbral se restaura a:

$$umbral = ratio_{divergencia} \times (1 - ratio_{divergencia}) \times tam_{cromosoma}$$

El esquema del algoritmo se muestra en la figura 3.3.

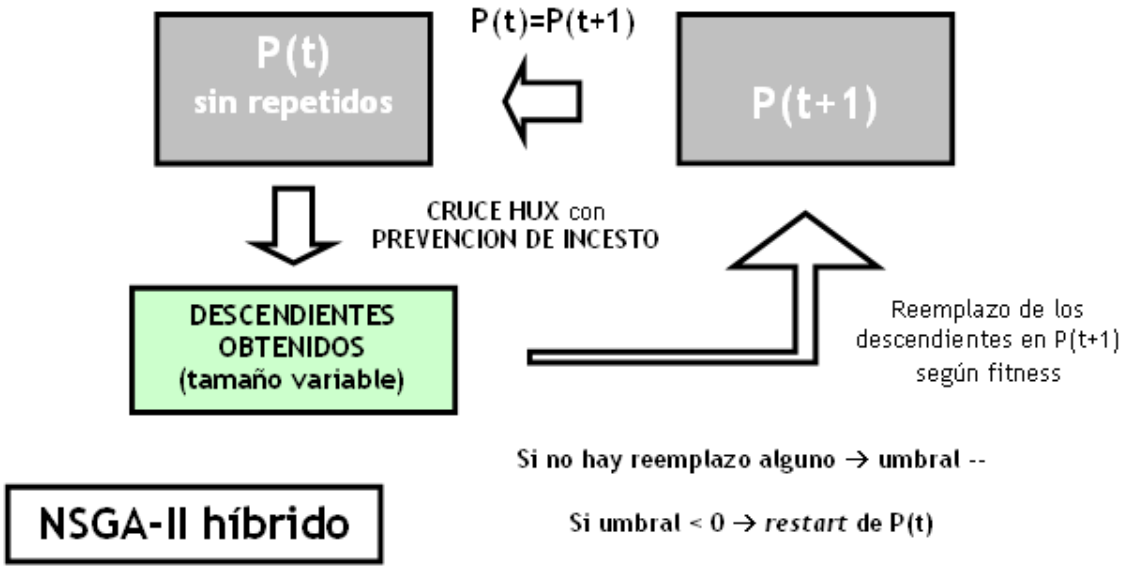

Figura 3.3: Estructura del algoritmo NSGA-II híbrido.

SPEA2 híbrido

La solución es bastante parecida a la anterior (figura 3.4) al ser algoritmos elitistas bastante parecidos. Los cambios introducidos son los siguientes:

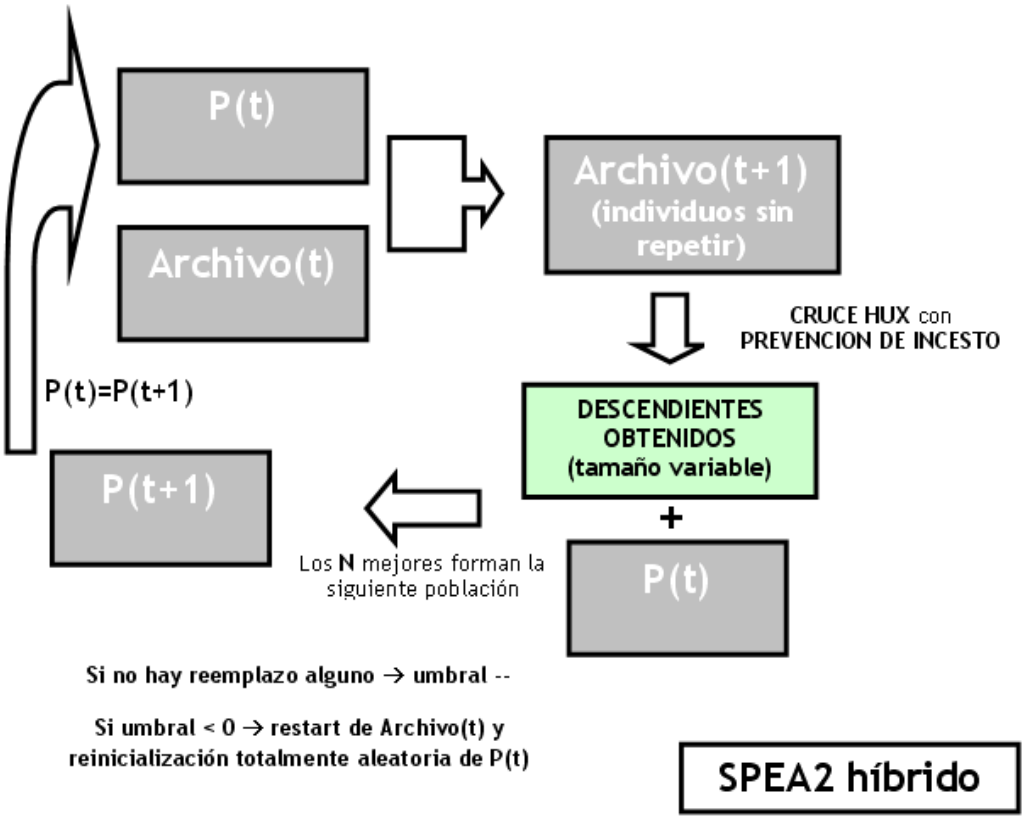

Figura 3.4: Estructura del algoritmo SPEA2 híbrido.

- Los objetivos son los mismos que en el NSGA-II, los dos ya mencionados.
- También se elimina la función de densidad ponderada por el número de características.
- No se permiten individuos idénticos en el archivo externo. Sí se permite en la población general.
- Al igual que en el NSGA-II se produce reinicialización y se utiliza el operador de cruce HUX con prevención de incesto.

Se crea un número determinado de descendientes a partir del archivo externo, si no se inserta ninguno en la población ordinaria se decrementa el umbral. Cuando este tenga un valor por debajo de cero se aplica una reinicialización en el archivo externo con un proceso parecido al que ocurría en el NSGA-II. La población ordinaria se reinicializa toda a ella a nuevos individuos aleatoriamente y el archivo externo (no tiene elementos iguales) se deja igual.

ϵMOEA híbrido

El funcionamiento de este algoritmo varía más respecto al de los dos anteriores (figura 3.5) por lo que hemos tenido que cambiar el enfoque híbrido que utiliza.

Introducimos aquí un nuevo concepto de cruce que nos vendrá bien por las características estacionarias del modelo del algoritmo.

Definición 10 *Un **cruce HUX intensivo** consiste en intentar forzar el cruce a toda costa. Así, si debido a la prevención de incesto no se puede producir el cruce, se eligen a otros padres hasta que se pueda realizar, o se rebase un límite de esperanza para ello. En el caso de que finalmente no se puedan generar los descendientes se considerará que los elementos son muy parecidos, aplicándose algún mecanismo de introducción de diversidad.*

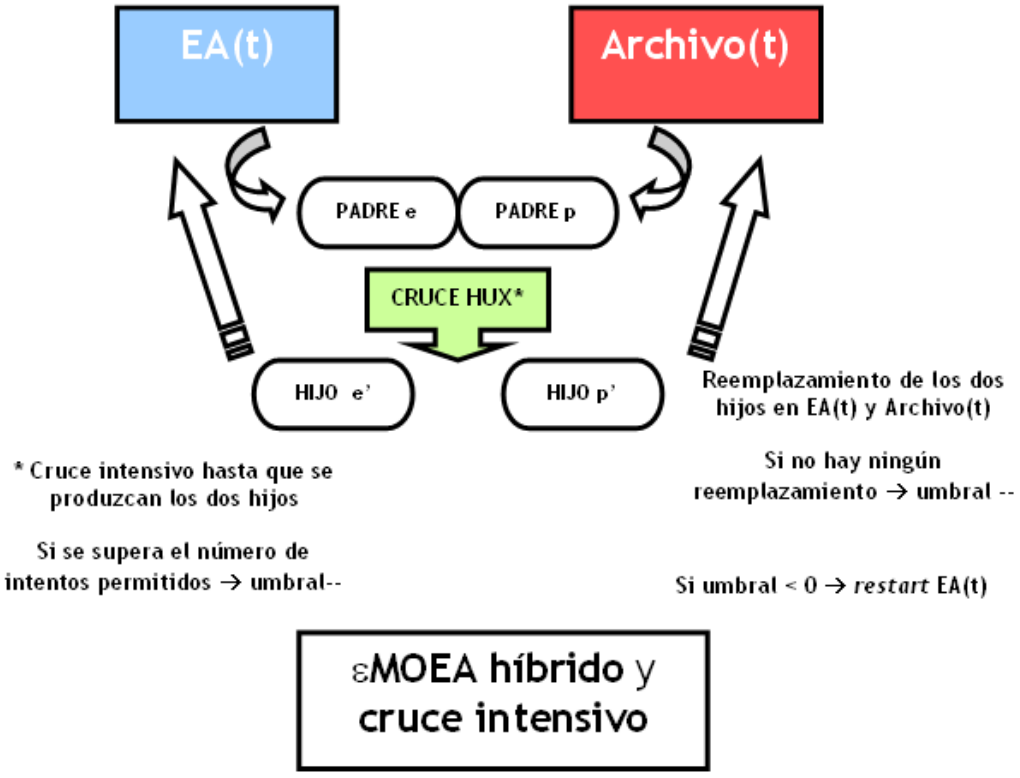

Figura 3.5: Estructura del algoritmo ϵ-MOEA híbrido.

Al cruzar sólo una vez por generación y al incluir prevención de incesto y cruce HUX se deberá probar varias veces para que se puedan generar los descendientes necesarios. Hemos fijado como valor de esperanza para que se produzca el cruce el valor $tam_{poblacion}/2$. Si aún así no se consigue cruzar se decrementará el umbral y se pasará a la siguiente generación.

La otra condición para que se decremente el umbral será si, aún habiéndose generado descendientes, ninguno de estos reemplaza a ningún individuo tanto de la población ordinaria como del archivo de ϵ-no-dominados.

Cuando el umbral llega a estar por debajo de cero se produce la reinicialización. El archivo no se toca ya que sólo contiene una copia de elementos ϵ-no-dominados y tienen que permanecer en él. Por contra, la población general se reinicializa por completo de la siguiente forma:

- El 30 % de los individuos se crean aleatoriamente por completo.
- El 70 % restante se crea también aleatoriamente pero usando como plantilla a los elementos ϵ-no-dominados del archivo, que se eligen al azar.

3.3. Diseño del multiclasificador evolutivo

En la sección 1.4 del capítulo 1 vimos como existían dos grupos fundamentales de multiclasificadores: **fusión de clasificadores** y **selección de clasificadores**. También existían varias técnicas o algoritmos para fusionar las salidas de los distintos clasificadores en una única, como eran el **voto de la mayoría**, **BKS** o **Naive-Bayes**.

Nosotros, basándonos en el trasfondo evolutivo y bioinspirado de nuestro sistema vamos a realizar una fusión de clasificadores en la que cada clasificador de la solución final se tenga en cuenta. El multiclasificador final vendrá dado por una combinación lineal de todos ellos:

$$multiclasificador = \omega_1 \times clasif_1 + \omega_2 \times clasif_2 + ... + \omega_n \times clasif_n$$

Donde $\omega_1, ..., \omega_n \in [0, 1] \wedge \omega_1 + ... + \omega_n = 1$ serán pesos de importancia de cada clasificador aprendidos mediante un algoritmo genético. La figura 3.6 muestra el proceso de formación del multiclasificador a partir de los pesos aprendidos por el AG.

Se podrá dar la opción al usuario de elegir qué clasificadores (en esta versión sólo el **kNN**, pero es fácilmente ampliable a los que se desen) y parámetros utilizar para los conjuntos de características devueltos por cada partición del problema (trabajaremos siempre con *10 validación cruzada*). Pero nuestra propuesta está muy basada en *computación evolutiva* por lo que se dará también la posiblidad de que un AG aprenda para cada conjunto de características qué clasificador con sus parámetros por defecto sería el más adecuado.

Es por esto por lo que tendremos dos tipos de AAGG:

- AG que aprende sólo los pesos de cada clasificador. La configuración de cada clasificador ha sido elegida por el usuario. La representación de cada solución vendrá dada por un valor entero por clasificador que representa el peso del mismo.

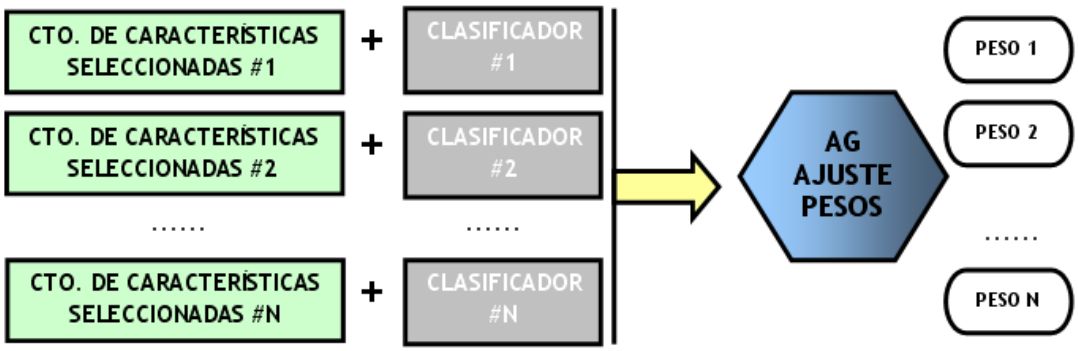

Figura 3.6: Proceso de ajuste de pesos del multiclasificador.

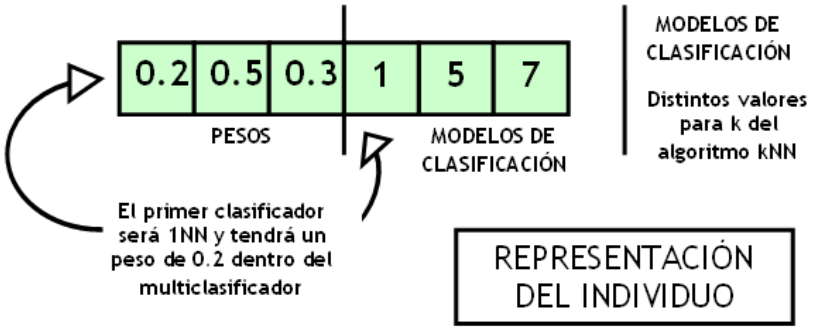

Figura 3.7: Esquema de representación del AG que aprende tanto los métodos de clasificación a utilizar como sus pesos.

- AG que aprende tanto los pesos como la configuración más adecuada de cada clasificador (la figura 3.7 muestra la representación para este tipo de AG).

Los dos AAGG tienen características comunes, por lo que su esquema será idéntico salvo por la representación del individuo. Sus características más importantes son las que se enumeran a continuación:

- **Representación del individuo.** La ilustración 3.7 mostraba que la codificación será real, con una parte destinada a los pesos y otra a los valores k. La longitud será el doble del número de clasificadores. En el caso de que sólo se aprendan los pesos el tamaño será igual al número de clasificadores. Los valores de $k \in \{1, 3, 5, 7, 9\}$.

- **Fitness.** La función *fitness* vendrá dada por el error en *training* del multiclasificador. La clasificación se produce como sigue: cada clasificador votará a una clase según su

peso. Se sumarán los pesos obtenidos por cada clase y la clase con mayor valor será la elegida.

- **Cruce HUX.** Tras realizar el cruce es necesario reajustar los pesos de los clasificadores para que estos sumen 1.

- **Mutación.** La mutación se producirá en un 1 % de los genes que indican el peso de cada clasificador. No es necesario mutar la información del valor del kNN ya que el espacio de soluciones es mucho menor que el de los pesos. La mutación consiste en cambiar un peso por un valor aleatorio entre 0 y 1. También hay que reajustar los pesos tras la mutación.

- **Enfoque CHC.** Se utiliza prevención de incesto con un ratio de divergencia de 0.35, con una reinicialización utilizando al mejor individuo como plantilla. En cada generación, los padres e hijos se ordenan según su *fitness*, cogiéndose los N mejores para formar la siguiente generación. El tamaño de la población se ha fijado a 100 individuos.

Tras el aprendizaje automático de los pesos de cada clasificador puede que alguno de ellos tenga un peso igual a 0, o un peso muy bajo que no influya en la decisión final del multiclasificador. Para no tener que guardar clasificadores innecesarios hemos establecido una función de borrado de clasificadores. Dicha función eliminará un clasificador cuando éste tenga **un peso menor o igual** que un umbral de pesado que se calcula como sigue:

$$\begin{cases} umbral_{peso} = 0, & \text{si } num_{clasif} <= 5 \\ umbral_{peso} = \frac{1}{num_{clasif}+3}, & \text{si } num_{clasif} > 5 \end{cases}$$

Esta función da un valor de umbral (la figura 3.8 muestra la gráfica de la función) que va decreciendo conforme aumenta el número de clasificadores. Esto es lógico porque cuantos más clasificadores tengamos en el multiclasificador, menos peso va a tener cada uno de ellos. Así, para 12 clasificadores, todos aquellos que tengan un peso menor o igual a 0.07 serán eliminados. Si por el contrario tenemos 100 clasificadores se eliminarán los que no superen el umbral de 0.0097. Si tenemos 5 o menos clasificadores sólo se borrarán aquellos cuyo peso sea nulo. La razón estriba en que con tan pocos clasificadores el resultado puede estar muy igualado y cualquier clasificador con poco peso puede tomar parte.

Una vez que para cada partición y ejecución del problema a tratar tengamos unos conjuntos de características seleccionados con sus clasificadores asociados y sus pesos de importancia, podremos ver qué multiclasificador de los 30 creados (10 particiones por ejecución, realizando 3 ejecuciones con distintas semillas) obtiene un mejor valor de precisión. Aquel que consiga los mejores resultados será el multiclasificador final del problema, y el que utilizaremos para clasificar nuevas instancias.

Figura 3.8: Función umbral para el peso de los clasificadores.

Clasificador por defecto

En el momento en que estudiamos y elaboramos el multiclasificador nos dimos cuenta de que era necesaria la inclusión de un clasificador por defecto que se activaría en ciertas ocasiones a la hora de clasificar nuevas instancias.

Supongamos que para un problema determinado hemos conseguido aprender y construir los 30 multiclasificadores de la forma anteriormente vista, y que hemos elegido el mejor multiclasificador de entre esos 30. Dicho clasificador estará compuesto por un conjunto de clasificadores, cada uno de los cuáles con un conjunto de características distinto y con un peso de importancia. Imaginemos también que estamos clasificando nuevas instancias a partir del multiclasificador creado, y que el usuario introduce valores para variables que no están en ningún clasificador del multiclasificador elegido, ¿cómo conseguimos clasificar la instancia? Pensamos que lo más correcto sería **introducir un clasificador por defecto** que entre en juego en el momento en que no se active ningún clasificador del multiclasificador (ver figura 3.9 para entender todo el proceso).

Un clasificador se activará cuando haya datos de todas las variables que utiliza. Como es lógico, puede que unos clasificadores se activen y otros no. En ese caso habrá que reajustar los valores de los pesos de los clasificadores activos, para que la suma de estos sea igual a la unidad.

Si no se activa ninguno de ellos habrá que utilizar el clasificador por defecto. Dicho clasificador por defecto será un clasificador **1NN** que utilice los datos de entrenamiento de la partición a la que pertenece el multiclasificador, y sólo las características que aporta el usuario.

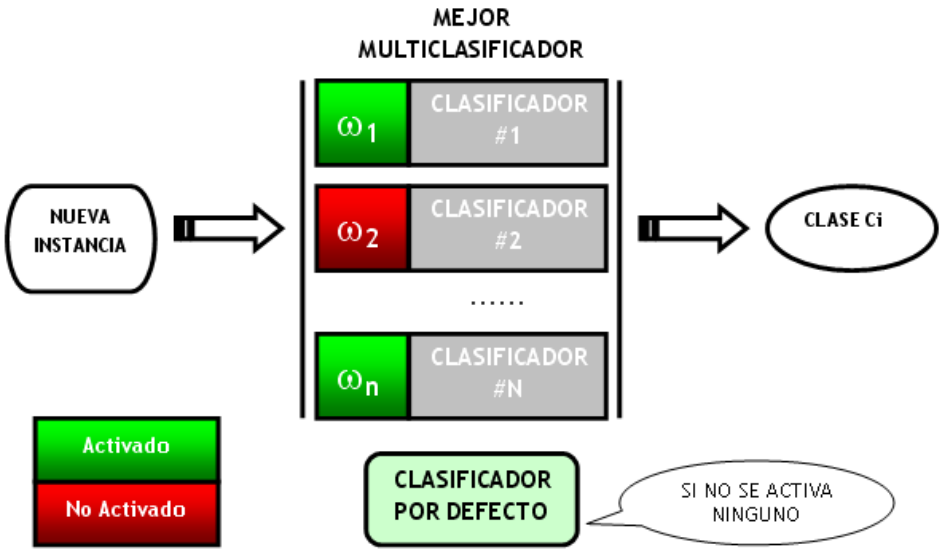

Figura 3.9: Proceso global de clasificación mediante el multiclasificador.

Tras mostrar en el siguiente capítulo todo el proceso de Ingeniería del Software seguido en el desarrollo del *Proyecto*, se presenta y analiza en el capítulo 5 toda la experimentación realizada tanto con los modelos evolutivos multiobjetivo implementados como con el multiclasificador.

Capítulo 4

Análisis y diseño de la aplicación

4.1. Requerimientos del sistema

4.1.1. Funcionales

La aplicación de Minería de Datos se centra en el campo de la clasificación y de la selección de características. Debe permitir lo siguiente:

- El sistema debe permitir **crear y entrenar un multiclasificador personalizable para cualquier problema que el usuario desee** siempre que los archivos de la base de datos del problema estén en el formato que acepta el sistema. Además, **tendrá por defecto ciertas bases de datos de problemas de clasificación populares** para que el usuario no tenga que especificar los archivos de los mismos.

  - El multiclasificador, al ser personalizable, debe dar la opción de poder elegir entre diversas técnicas de selección de características y clasificación a la hora de ser creado.
  - Para la selección de características, se podrá elegir entre los tres algoritmos genéticos multiobjetivo elegidos: NSGA-II, ϵ-MOEA y SPEA2.
  - Para la clasificación también se ofertará como mínimo un método, pudiendo actualizar fácilmente el sistema si se deseara incluir más posteriormente.
  - Se deberá utilizar un AG para que dé pesos de importancia a los distintos clasificadores que forman el multiclasificador.
  - Si el usuario lo desea, en vez de elegir manualmente los métodos de clasificación y sus parámetros, **el sistema puede aprender de forma totalmente automática los mejores clasificadores a utilizar con cada conjunto de características así como sus pesos** por medio de un AG que obtenga la mejor configuración posible para cada multiclasificador.

- El sistema deberá permitir la introducción cómoda e interactiva por parte del usuario de los parámetros de configuración de todos los métodos y algoritmos utilizados. Por tanto, **el proceso de aprendizaje será interactivo con el usuario**.
- También se ofrecerá al usuario una visualización paso a paso de todo el proceso de minería de datos, mostrando estadísticas básicas y resultados intermedios.
- En la misma interfaz del usuario deberá aparecer una breve ayuda que indique qué hacer en cada paso del proceso de aprendizaje.
- El sistema también deberá albergar una pequeña base de datos con resultados de clasificación con otras técnicas ajenas al *Proyecto* para poder comparar en la misma aplicación si el multiclasificador creado supera a dichas técnicas o no. Estos datos sólo se ofrecerán para el reducido número de problemas que el sistema tendrá por defecto.
- El sistema creará, al finalizar el aprendizaje del multiclasificador, un archivo para poder restaurar el multiclasificador posteriormente.
- Se deberá permitir la **clasificación de nuevas instancias a partir de un multiclasificador creado con antelación**.
  - El usuario podrá cargar dicho multiclasificador a partir de un fichero.
  - Se podrá dar al sistema de Minería de Datos una nueva instancia para ser clasificada o un fichero que contenga una batería de instancias. Además, esas nuevas instancias podrán tener valores perdidos o nulos. El formato del archivo de instancias será el mismo que el de la BD.
  - Al finalizar el proceso de clasificación se mostrará la clase a la que pertenece cada una de las instancias según el multiclasificador.

4.1.2. No funcionales

El sistema tendrá dos partes que deben estar totalmente separadas, la **interfaz** y la **lógica de la aplicación**.

- La interfaz deberá ser amigable y totalmente accesible para personas no familiarizadas con la Minería de Datos. La ejecución tanto del aprendizaje del multiclasificador como de la clasificación debe ser lo menos engorrosa posible.
- Se intentará que toda la lógica de la aplicación o alguno de sus elementos puedan ser reutilizados en otros proyectos o marcos de trabajo. Por lo que sí es necesario que haya una total separación entre interfaz y lógica de negocio.
- Los archivos de las BBDD deberán seguir una estructura determinada, a poder ser, la misma que utilizan otros programas de Minería con el objetivo de compartir fuentes de datos.

- Todos los ficheros que se generen se realizarán utilizando la tecnología **XML**.

4.1.3. Modelado mediante casos de uso

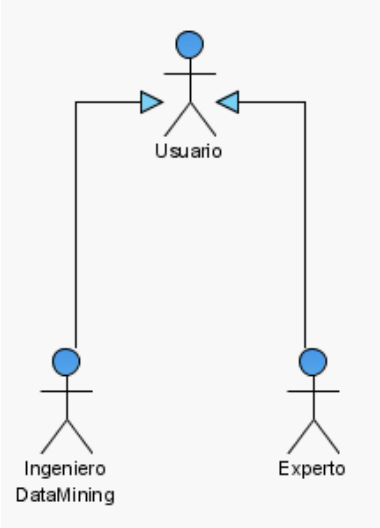

Figura 4.1: Perfiles de usuario del sistema.

La aplicación de Minería de Datos estará enfocada a dos tipos de usuarios distintos:

Ingeniero o investigador en Minería de Datos

- **Descripción y características del usuario**
  Va ser una persona totalmente preparada en Computación y Minería de Datos que utilizará el módulo de aprendizaje del multiclasificador, por lo que conocerá todos los pasos del proceso, parámetros de los diferentes algoritmos y métodos. Podrá configurar el multiclasificador de la manera más adecuada a las necesidades del problema o del usuario final o experto.

- **Requisitos de usabilidad**
  Ya que es una persona acostumbrada al uso de estos sistemas, no precisará de una interfaz demasiado *ingenua* o simple. Se deberán centrar los esfuerzos en que el usuario pueda configurar el proceso de aprendizaje, y en mostrar estadísticas y otros datos técnicos que puedan serle de ayuda a la hora de comprender los resultados que está dando el multiclasificador.

Usuario experto

- **Descripción y características del usuario**
  Este usuario es la persona que, en teoría, utilizará el multiclasificador una vez que este esté configurado y formado para que pueda utilizarlo como una herramienta de apoyo a la toma de decisiones en su área de trabajo. No tiene porqué tener conocimientos de Minería de Datos ni de Computación, sólo debe estar familiarizado con entornos Web y ordenadores personales. Un ejemplo de este usuario puede ser un determinado **médico** que necesita diagnosticar una enfermedad a sus pacientes y que utilizará este sistema para que le ayude en su tarea.

- **Requisitos de usabilidad**
  Aunque tiene que tener formación en Informática como cualquier otro usuario de una aplicación, la interfaz deberá ser más amigable que para el otro perfil de usuario, con más elementos gráficos e imágenes para que el usuario no sienta rechazo a su utilización. Se deberán omitir de esta interfaz todas las alusiones o conceptos propios del campo de la Minería de Datos y de la Computación, ya que va a ser un usuario ajeno a ella el que la utilice.

Intentando reflejar los requisitos que tiene el sistema, vamos a modelar mediante *UML*, y más concretamente, utilizando *casos de uso*, las características que tendrá que cubrir el prototipo final que implementaremos.

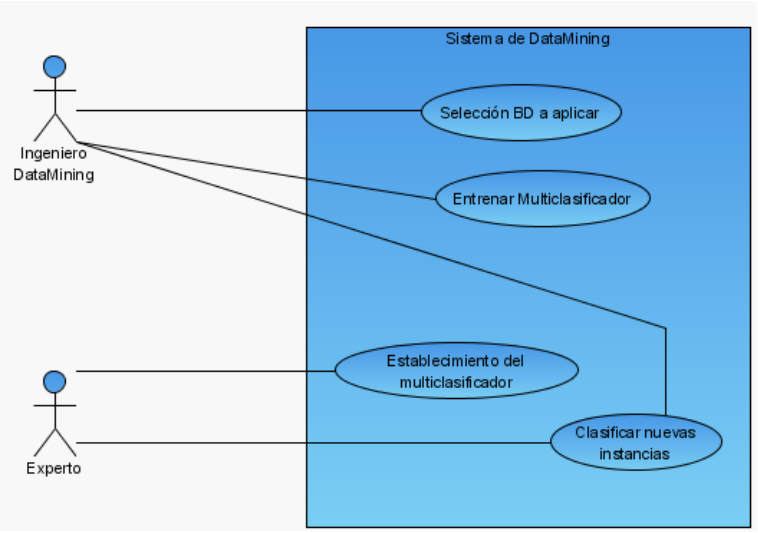

Figura 4.2: Diagrama frontera del sistema.

El caso de uso fundamental es el *diagrama frontera*, que refleja las relaciones entre los distintos actores y el sistema. Los casos de uso más importantes del sistema se muestran dentro de la caja que lo delimita y son los que modelan las funcionalidades principales. El sistema se relacionará con dos tipos de usuarios como hemos visto anteriormente.

Debido a que el caso de uso que posee un mayor número de requisitos es el llamado *Entrenar Multiclasificador*, que comprende tanto todo el proceso de selección de características, clasificación, multiclasificador con ajuste de pesos e incluso la interacción con el usuario en los distintos pasos del aprendizaje, hemos decidido detallar más el caso de uso, obteniendo cuatro casos de uso más dentro del mismo, como se aprecia en la figura 4.3.

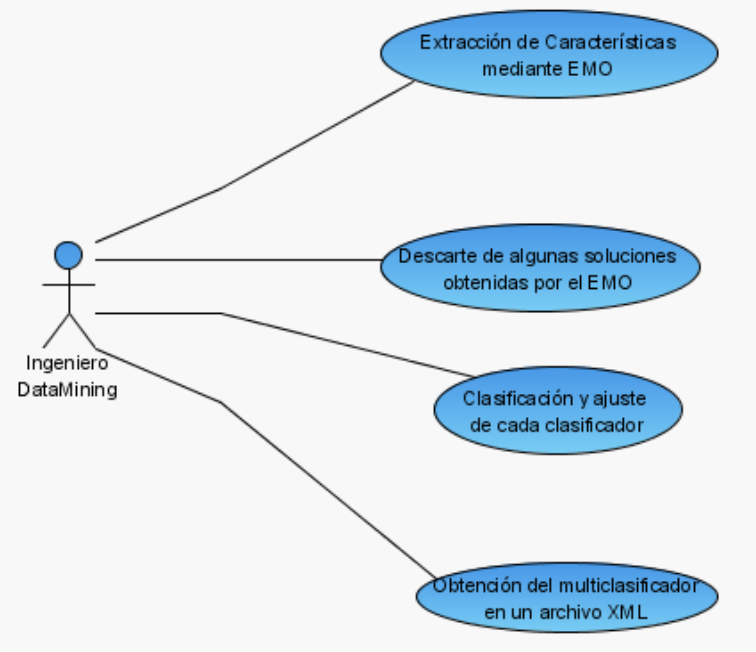

Figura 4.3: Detalle del caso de uso *Entrenar Multiclasificador*.

Además, hemos considerado que sería necesario expresar mediante casos de uso dos excepciones importantes del sistema. Eso no quiere decir que no existan más, sino que las excepciones *ErrorFormatoArchivoMulticlasificador* y *ErrorFormatoArchivoBD* son las más interesantes de especificar en la etapa de análisis del sistema.

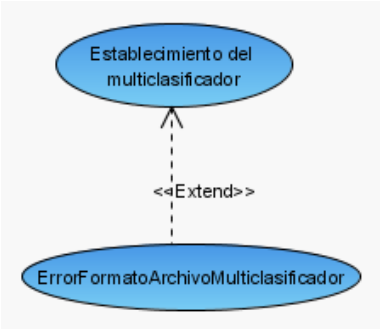

Figura 4.4: Excepción *ErrorFormatoArchivoMulticlasificador*.

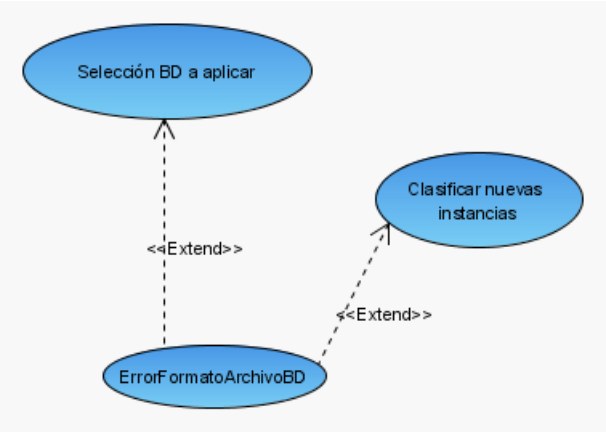

Figura 4.5: Excepción *ErrorFormatoArchivoBD*.

También existen en el modelado que hemos realizado ciertos casos de uso que tienen una complejidad mayor que los otros, por lo que optamos por documentarlos ampliamente en los **tablas 4.1-4.5** que presentamos a continuación.

Los casos de uso que documentamos son los llamados *Selección BD a aplicar*, *Establecimiento del clasificador* y *Clasificar nuevas instancias* del diagrama frontera, y el caso de uso *Clasificación y ajuste de cada clasificador* perteneciente al caso de uso *Entrenar Multiclasificador*.

Nombre caso de uso: Selección BD a aplicar	
Actores: Ingeniero *DataMining*	
Descripción: Se especifica sobre qué base de datos se quiere entrenar el multiclasificador. Si la base de datos es muy común no tendrá que proporcionar al sistema ningún fichero, ya que éste lo tendrá por defecto.	
Precondiciones: El actor inicia la subaplicación de entrenamiento del multiclasificador. Se deberán tener los archivos en el formato que acepta el sistema.	
Postcondiciones: Se carga correctamente la base de datos.	
Flujo de eventos:	
Acciones usuario	**Acciones sistema**
1. Inicia subaplicación de aprendizaje.	
	2. Se da la opción de carga de BD propia o las que tiene el sistema por defecto.
3. Se elije BD por defecto o no.	
	4. El sistema pide al usuario que se le proporcionen los archivos.
5. El usuario sube los archivos de la BD.	
	6. Se carga correctamente la BD.
Flujo alternativo: Si se elije una BD por defecto en el paso 3 se salta directamente al paso 6.	
Excepciones: ErrorFormatoArchivoBD.	

Tabla 4.1: Documentación del caso de uso *Selección BD a aplicar*.

Nombre caso de uso: Clasificación y ajuste de cada clasificador	
Actores: Ingeniero *DataMining*	
Descripción: Para cada solución dada por el algoritmo genético multiobjetivo se va a crear un clasificador, en este caso de uso se elegirá qué tipo de clasificador asociar a cada solución junto con sus parámetros (si se desea, el proceso se realizará automáticamente).	
Precondiciones: La base de datos con la que se entrenará está correctamente cargada, se han seleccionado las características mediante un algoritmo multiobjetivo y el actor, en su caso, ha descartado las soluciones que no se ajustan a sus preferencias.	
Postcondiciones: El sistema multiclasificador quedará perfectamente definido y creado, ajustándose los pesos de cada clasificador y con una vista del porcentaje de error y ciertas estadísticas sobre él.	
Flujo de eventos:	
Acciones usuario	**Acciones sistema**
1. El actor se encuentra en la pantalla de configuración de los distintos clasificadores.	
	2. El sistema da la posibilidad de que el usuario ajuste los clasificadores o de que se aprenda automáticamente por el sistema.
3. En su caso, el usuario elige los clasificadores y sus parámetros.	
	4. Filtrado de esos parámetros. El sistema tiene todos los datos, creando los distintos multiclasificadores.
	5. De entre todos los multiclasificadores se elige el mejor atendiendo a su precisión, mostrándose información sobre él y estadísticas.
Flujo alternativo: Si se elije un aprendizaje automático de los clasificadores se pasa del paso 2 al 5 final.	
Excepciones: Ninguna	

Tabla 4.2: Documentación del caso de uso *Clasificación y ajuste de cada clasificador*.

Nombre caso de uso: Establecimiento del multiclasificador	
Actores: Experto	
Descripción: El experto necesita clasificar nuevas instancias para apoyarse en sus decisiones futuras. Cargará un multiclasificador, anteriormente creado por el Ingeniero de *Datamining*, para que pueda clasificar correctamente.	
Precondiciones: El Experto se encuentra en la subaplicación de clasificación de nuevas instancias, y posee el archivo XML que define al multiclasificador. En su caso, también dispondrá de los ficheros de entrenamiento usados por el multiclasificador en el entrenamiento.	
Postcondiciones: El multiclasificador se recupera correctamente y está listo para recibir y clasificar nuevas instancias del problema.	
Flujo de eventos:	
Acciones usuario	**Acciones sistema**
1. Inicia subaplicación de clasificación de nuevas instancias.	
	2. El sistema ofrece un diálogo para obtener el multiclasificador.
3. El usuario experto ofrece el fichero XML que representa al multiclasificador.	
	4. Se procesa el fichero. Si algún clasificador del multiclasificador está basado en KNN, y además el problema no se encuentra en el sistema, se pedirá al usuario que proporcione el fichero de entrenamiento de la partición sobre la que aprendió el multiclasificador.
5. El experto proporciona al sistema el fichero entrenamiento de la partición pedida.	
	6. Termina de crearse el multiclasificador.
Flujo alternativo: Si el multiclasificador no se basa en KNN, o bien, la BD que utiliza está en el sistema, el flujo pasa al paso 5.	
Excepciones: ErrorFormatoMulticlasificador	

Tabla 4.3: Documentación del caso de uso *Establecimiento del multiclasificador*.

Nombre caso de uso: Clasificar nuevas instancias	
Actores: Ingeniero DataMining y Experto.	
Descripción: Con el multiclasificador cargado, el actor puede suministrar al sistema una nueva instancia para que el sistema la clasifique.	
Precondiciones: El multiclasificador está cargado correctamente y el actor está en la pantalla de clasificación de nuevas instancias.	
Postcondiciones: La instancia se clasifica por el multiclasificador, mostrándose el resultado.	
Flujo de eventos:	
Acciones usuario	**Acciones sistema**
1. El usuario desea y se encuentra en la pantalla de clasificación de nuevas instancias.	
	2. Se ofrece la opción de suministrar un fichero con una batería de instancias.
3. El ingeniero o el experto le ofrecen dicho fichero con un número de instancias variable en el formato aceptado por el sistema	
	4. Se carga correctamente el fichero de instancias, y se ofrece al usuario la posibilidad de elegir una instancia de entre todas las que el fichero incluye.
5. El actor elige una instancia de entre las del fichero.	
	6. A partir de la instancia elegida, cada clasificador del multiclasificador general se activa si tiene los datos suficientes, ponderándose esos clasificadores activos.

Tabla 4.4: Documentación del caso de uso *Clasificar nuevas instancias.*

Continuación del flujo de evento del caso de uso...	
	9. Si no se activa ningún clasificador del sistema, se utiliza el clasificador por defecto, consistente en un KNN basado en la partición de entrenamiento del multiclasificador.
	10. Si fuera necesario, se pide al usuario que suministre el fichero de la partición necesaria.
11. El experto proporciona al sistema el fichero de entrenamiento de la partición pedida.	
	12. El sistema devuelve la clase de la instancia, y da la posibilidad al usuario de elegir más instancias para volver a clasificarlas.
Flujo alternativo: Si entra en juego el clasificador por defecto, pero el usuario ya ha suministrado el fichero de la partición, o es un problema que ya tiene el sistema, entonces se pasa del paso 9 al 12.	
Excepciones: ErrorFormatoMulticlasificador	

Tabla 4.5: Continuación del caso de uso *Clasificar nuevas instancias.*

4.2. Alternativas del sistema

Alternativa #1: Aplicación en C con interfaz de consola

Con esta alternativa se pretende desarrollar el sistema utilizando el lenguaje de programación **C** y una interfaz simple mediante consola **UNIX**.

La principal ventaja de esta alternativa es la eficiencia del lenguaje **C** muy útil cuando trabajamos con algoritmos lentos como los **EMO** que estamos desarrollando. Además, el trabajo mediante consola puede hacer que para un experto sea bastante rápida la utilización de la herramienta. Para ejecutar la aplicación sólo será necesario disponer de un ordenador personal, no harán falta servidores ni equipos auxiliares.

Por contra, existen bastantes inconvenientes para esta alternativa. El lenguaje de programación **C** es más arcaico que otros lenguajes, no permitiendo una cómoda conexión con tecnologías que debemos utilizar como **XML**.

La interfaz mediante comandos puede retraer a usuarios poco dóciles y con espíritu menos espartano, asumiendo el riesgo de que la aplicación no llegue a ser utilizada.

Por último, utilizando esta alternativa va a ser más difícil llevar a cabo técnicas modernas de Ingeniería del Software como modelado **UML** o patrones de diseño **MVC**.

Alternativa #2: Aplicación en JAVA con tecnología Cliente/Servidor

Todo el sistema de Minería de Datos será construido siguiendo el paradigma **cliente-servidor** y utilizando **JAVA** como tecnología tanto para el servidor como para la interfaz Web de la aplicación. La arquitectura básica se muestra en la figura 4.6.

La lógica de la aplicación deberá residir en un servidor para que sea accesible desde la interfaz Web del sistema. Todas las operaciones de aprendizaje, al menos, deberán ejecutarse en el servidor y no en el cliente.

Como ventajas de utilizar la tecnología **JAVA** tenemos la facilidad de programación con tecnologías nuevas, la posibilidad de intercambiar la interfaz Web por otra distinta sin necesidad de tocar el núcleo de la aplicación, o la posibilidad de aplicar fácilmente técnicas de Ingeniería del Software.

La arquitectura **cliente-servidor** nos permitirá realizar los cálculos más complejos en máquinas ajenas y más potentes. Mientras que la interfaz Web puede dar un carácter más sencillo y divertido a la aplicación.

Como desventaja más importante tenemos que la ineficiencia de **JAVA** puede ser un obstáculo a la hora de ejecutar algoritmos evolutivos. Incluso este alternativa puede planter problemas de tipo más económico, al necesitar una máquina potente y cara como servidor Web.

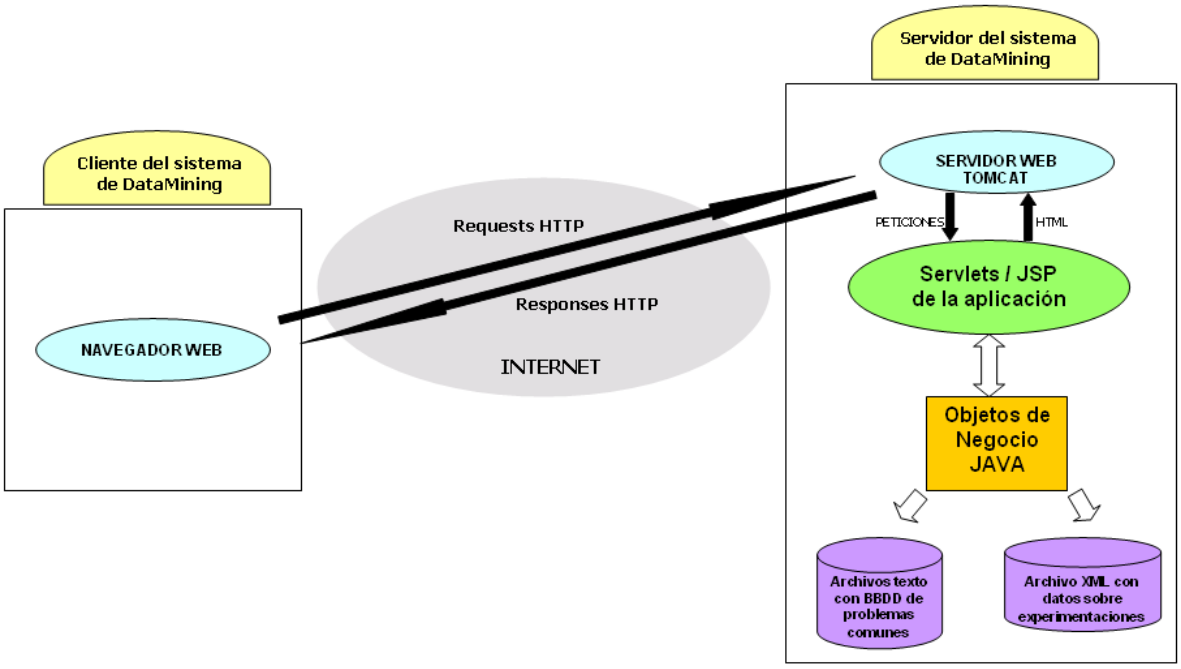

Figura 4.6: Arquitectura del sistema de Minería de Datos.

Alternativa #3: Aplicación de escritorio en JAVA

Utiliza el mismo lenguaje y tecnología para el cuerpo de la aplicación que la alternativa anterior. Sin embargo, la diferencia está en que no se adopta un enfoque **cliente-servidor**, sino que la aplicación es local al igual que la alternativa #1.

La interfaz será tipo *windows* con metáfora de escritorio. Será necesaria la descarga o instalación en el ordenador del usuario de toda la aplicación, así como su total ejecución.

Como ventaja más clara respecto a las demás alternativas tenemos que la interfaz es amigable y más cómoda que por consola, sin depender de un servidor como lo hacía la alternativa #2.

Sobresale como desventaja la gran ineficiencia que tendría la aplicación si el ordenador del usuario, como suele ser normal, no es una máquina muy potente.

Elección de la alternativa

La alternativa elegida es la **alternativa #2** que utiliza tanto tecnología **JAVA** como arquitectura **cliente-servidor** a través de la Web.

Las razones fundamentales de su elección han sido:

- La posibilidad de ser una aplicación multiplataforma.

- La potencia del lenguaje **JAVA**.
- La conveniencia de que la aplicación se ejecutase en un servidor potente de forma transparente al usuario.

También ha influido el atractivo de utilizar una página Web como interfaz para la aplicación y la comodidad de conexión con otras tecnologías.

La posibilidad de reutilizar parte de la aplicación en otros sistemas de Minería de Datos como **KEEL**[1], y que estos sistemas se encuentren implementados en **JAVA** han hecho que utilizar este lenguaje de programación sea prácticamente un requisito a cumplir.

4.3. Planificación del desarrollo software

Vamos a fijar la fecha de comienzo del desarrollo *software* a 10 de mayo de 2006, fecha para la cuál ya tendrían que estar terminadas las tareas predecesoras necesarias (ver el preámbulo Introducción y objetivos del *Proyecto* para la planificación global del mismo).

Según la planificación de todo el *Proyecto*, el desarrollo de la aplicación tendría una duración de 17 semanas.

Las tareas en las que hemos descompuesto el proceso de Ingeniería del *Software* son las siguientes:

ID	ACTIVIDAD	TIEMPO	COMIENZO	FIN
A	Manual preliminar del usuario	2	10/5/06	23/5/06
B	Análisis del sistema	2	24/5/06	6/6/06
C	Diseño a nivel de paquetes	1	7/6/06	13/6/06
D	Diseño clases para SC	3	14/6/06	4/7/06
E	Diseño clases Multiclasificador	2	14/6/06	27/6/06
F	Diseño de la Interfaz Web	2	14/6/06	27/6/06
G	Implementación prototipo SC	2	5/7/06	18/7/06
H	Implementación prototipo Multi	1	28/6/06	4/7/06
I	Implementación prototipo de la Interfaz	2	28/6/06	11/7/06
	TOTAL	**17**	**10/5/06**	**18/7/06**

Mostramos en las figuras 4.7 y 4.8 los diagramas de *Gantt* y de *Pert* de la planificación. En el diagrama de *Gantt* especificamos la duración de las tareas y fechas a cumplir, así como la dependencia de unas actividades con otras.

El diagrama de tareas o de *Pert* (figura 4.8) muestra de color naranja las tareas que forman el camino crítico entre los hitos de inicio y fin del desarrollo *software*.

[1]KEEL: `http://www.keel.es`

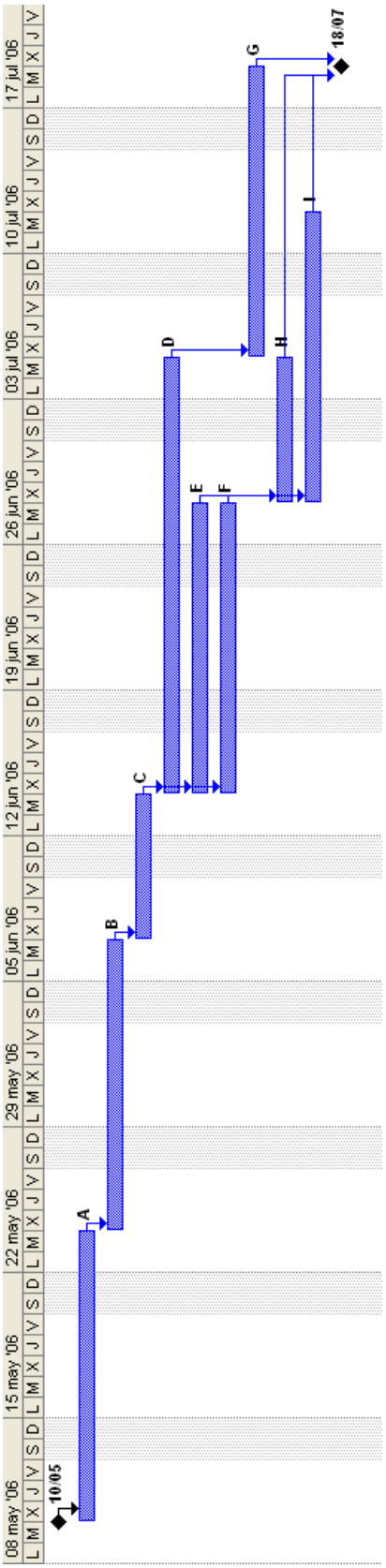

Figura 4.7: Diagrama de Gantt de las tareas de desarrollo *software*.

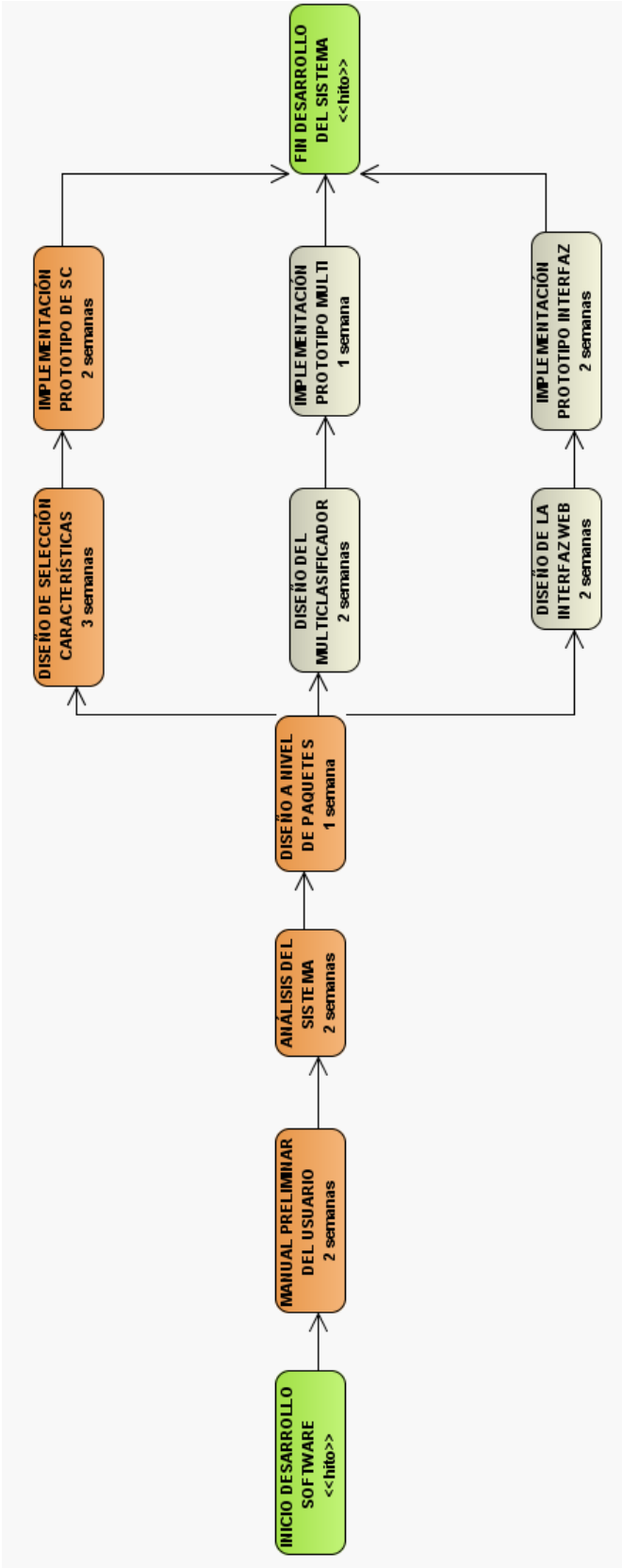

Figura 4.8: Diagrama de red de las tareas de desarrollo *software*.

4.4. Análisis del Sistema

4.4.1. Manual preliminar del usuario

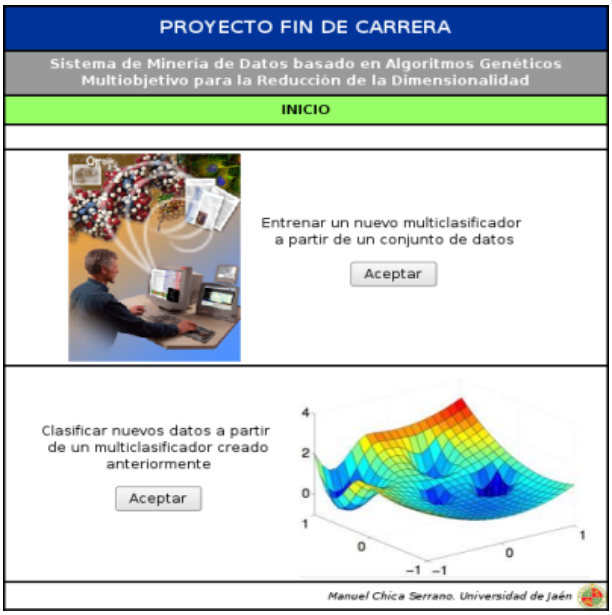

Figura 4.9: Pantalla inicial del Sistema Web.

Inicio del aprendizaje. Selección de la base de datos

La aplicación Web se inicia mostrando dos opciones que son las principales del sistema que estamos definiendo. La primera opción entrena y crea un multiclasificador a partir del conjunto de ficheros que forman la base de datos del problema y de los parámetros que vamos a ir suministrándole. La segunda opción da la posibilidad de clasificar instancias a partir de un multiclasificador creado con anterioridad.

El primer paso, por tanto, sería crear un multiclasificador. Para ello se hará click en el botón *Aceptar* de la subaplicación de entrenamiento, apareciendo el primer paso de entrenamiento del multiclasificador (figura 4.10).

En este paso será necesario indicar los **valores enteros** que pasarán a ser las tres semillas para las tres ejecuciones que se realizarán a lo largo del proceso. Estas semillas nos servirán para inicializar los generadores de números aleatorios que utilizan algoritmos probabilísticos como los AAGG. Las tres semillas tienen que ser distintas, ya que si establecemos el mismo valor para cada una de ellas nos encontraremos con los mismos resultados para cada ejecución.

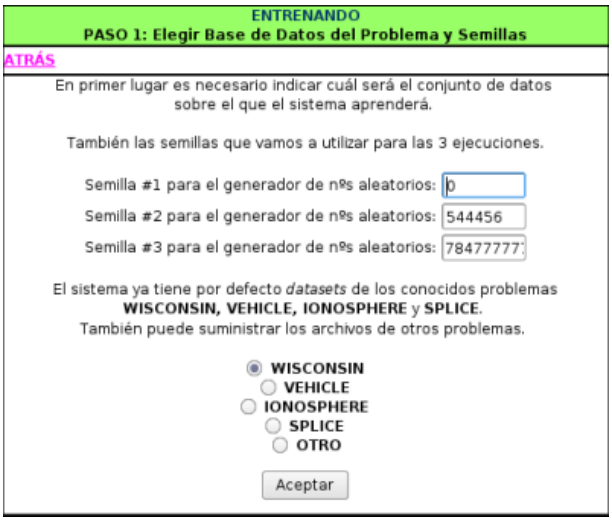

Figura 4.10: Pantalla de elección de BBDD y semillas para generador aleatorio.

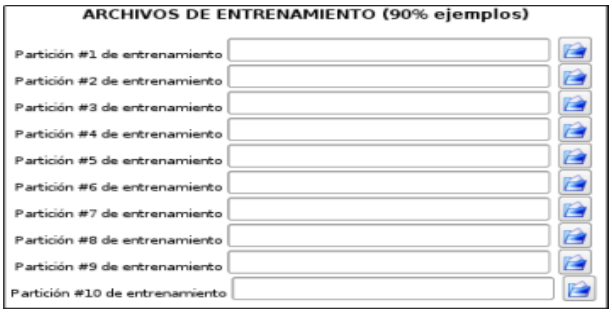

Figura 4.11: Pantalla de selección de archivos de una BD que no es por defecto.

Aparte del establecimiento de las semillas también tenemos que indicar la base de datos del problema sobre la que vamos a aprender el multiclasificador. Existen cuatro por defecto que podremos seleccionar, no habiendo necesidad de suministrar los ficheros de datos en este caso.

Sin embargo, si deseamos otra base de datos la aplicación nos dará la posibilidad de suministrarle los ficheros de datos. Para ello habrá que seleccionar la opción *Otro* y pulsar *Aceptar*. Seguidamente nos aparecerá una pantalla con 20 *inputs* (figura 4.11) para que especifiquemos mediante un cuadro de diálogo la ubicación de cada fichero de *training* y de *test*.

Tenemos que tener en cuenta que la base de datos debe haber sido particionada mediante **10 validación cruzada**, por lo que tendremos en nuestro poder 10 ficheros de entrenamiento y 10 de prueba. Pulsaremos *Aceptar*, y si los ficheros proporcionados son correctos llegaremos al segundo paso de configuración del multiclasificador (figura 4.12).

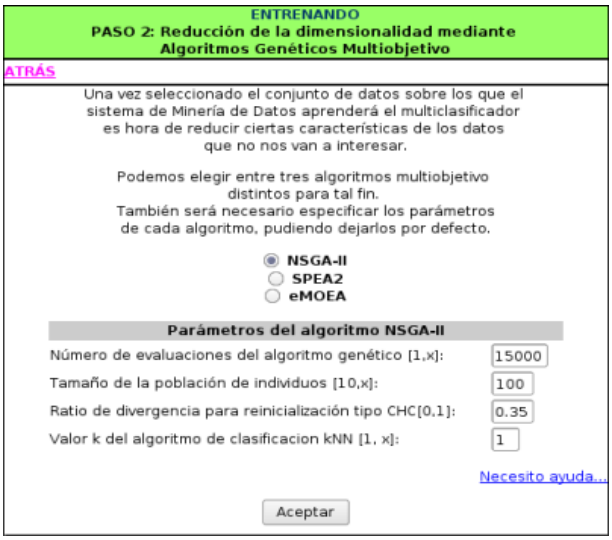

Figura 4.12: Pantalla de selección del EMO a utilizar junto a sus parámetros.

Aplicación de algoritmos de reducción de la dimensionalidad

Una vez indicada la base de datos y las semillas que se utilizarán para las tres ejecuciones podemos reducir la dimensionalidad de la base de datos aplicando Algoritmos Genéticos Multiobjetivo.

Nos encontramos en el paso 2 y podemos ver cómo la página Web nos muestra tres posibles algoritmos: NSGA-II, SPEA2 y ϵMOEA. Pulsando sobre el elegido se nos cambiará automáticamente la zona de parámetros, ya que cada uno de ellos tiene unos parámetros determinados (figura 4.12).

Estos parámetros son típicos de cualquier EMO y pulsando sobre el enlace *Necesito ayuda...* se puede ver información más detallada sobre cada uno de ellos. A pesar de ello podremos dejar los parámetros por defecto que nos aparecen directamente en la página Web si no queremos, o no sabemos, qué valores dar a dichos parámetros.

Tras haber elegido el algoritmo y los parámetros, y haber pulsado el botón *Aceptar* se ejecutarán los algoritmos para cada par partición-ejecución del problema, por lo que puede que el sistema tarde varios minutos en mostrar la siguiente página Web, apareciendo una ventana emergente de espera.

La página Web siguiente mostrará las soluciones que ha obtenido el algoritmo de reducción de la dimensionalidad (ver figura 4.13), indicando el error en *test* de cada solución y el número de características seleccionadas. Podremos descartar las soluciones que queramos pulsando el botón *Descartar* que se encuentra al lado de cada solución.

Como ya hemos indicado, existen un número variable de soluciones por cada ejecución y cada partición. Debido a ello el sistema nos da la posibilidad de navegar por cada ejecución-partición, seleccionando en la lista desplegable del apartado *Visionar soluciones* la ejecución

Figura 4.13: Pantalla de las soluciones devueltas por el EMO.

y partición que deseamos ver, pulsando seguidamente sobre el botón *Ver* para refrescar la pantalla.

Si deseamos ver las estadísticas de cada partición-ejecución (tendremos que haberla seleccionado anteriormente en el apartado *Visionar soluciones*), y la media de todas las particiones y ejecuciones, necesitamos hacer click en el enlace inferior derecho que posee el texto *Ver Estadísticas*.

Consecuentemente nos aparece una ventana emergente con dos partes bien diferenciadas. La parte superior nos muestra las medias y desviaciones estándar de la partición-ejecución que teníamos seleccionada en la página principal respecto al número de características seleccionadas y error en *test*. La parte inferior de la ventana emergente nos indica las mismas medidas pero para todas las soluciones en general. Pulsaremos el botón *Cerrar* para ocultar la ventana emergente de estadísticas.

Configuración y creación del multiclasificador

Ya hemos podido descartar las soluciones que no deseábamos, y visionar las estadísticas de los algoritmos de reducción de la dimensionalidad, por lo que podemos pasar a la última fase de creación del multiclasificador pulsando el botón *Aceptar*.

En este paso 3 se nos da la posibilidad de elegir entre un **aprendizaje automático** (no tendremos que elegir los parámetros para cada clasificador formado a partir de cada solución de los algoritmos de selección de características) o una *configuración manual* en la que el usuario es quién decide qué parámetros serán los más oportunos para cada clasificador y solución.

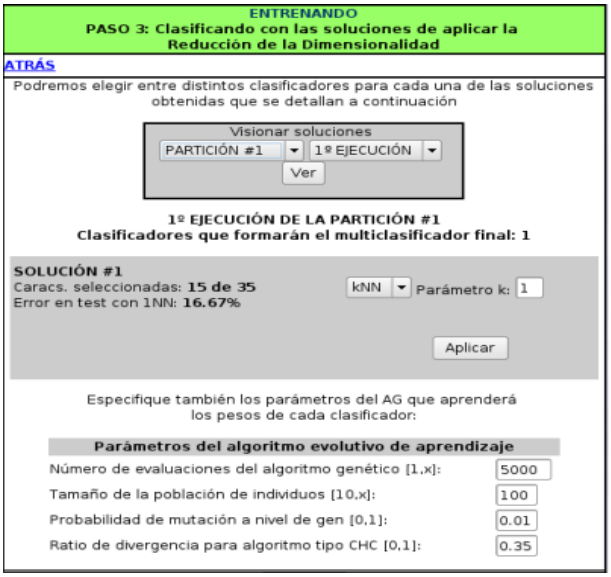

Figura 4.14: Pantalla de establecimiento de los parámetros de cada clasificador.

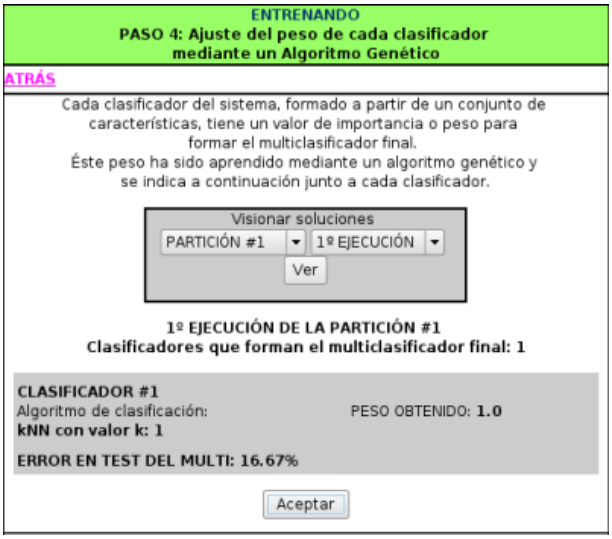

Figura 4.15: Pantalla que muestra los pesos aprendidos.

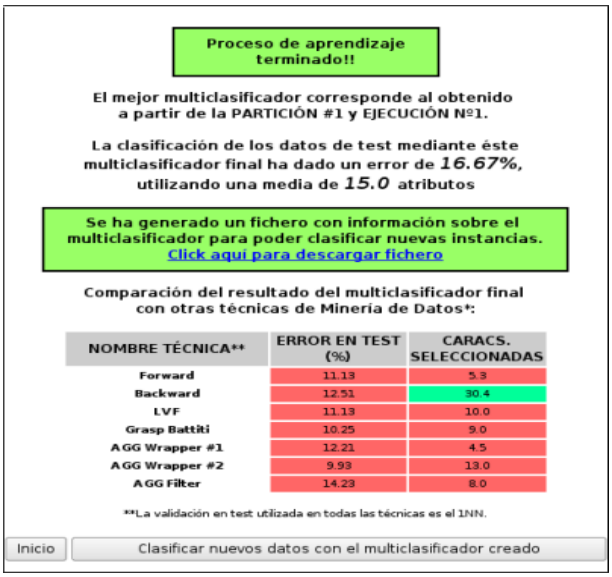

Figura 4.16: Pantalla final del entrenamiento del multiclasificador.

Si tenemos alguna duda podemos pulsar el enlace *Necesito Ayuda...* que nos explica más detenidamente la diferencia entre elegir una u otra opción. Pulsaremos el botón de la opción que más se ajuste a nuestras necesidades.

Si deseamos una configuración manual nos encontramos con una página (ver figura 4.14) que nos permite, para cada solución, cambiar el parámetro k del algoritmo **kNN** que es el único que se encuentra por ahora disponible en el sistema. Los valores permitidos para el parámetro k de cada clasificador son el **1,3,5,7** o **9**. Pulsando el botón *Aplicar* guardaremos los cambios realizados.

Al igual que ocurría con las soluciones del NSGA-II, SPEA2 o ϵMOEA, también podemos visionar los distintos clasificadores que estamos formando a través del apartado *Visionar Soluciones* y pulsando el botón *Ver*.

Por último, antes de formar los multiclasificadores tendremos que, tanto para la configuración manual como para la automática, especificar los parámetros del AG que genera los pesos de cada clasificador. La mayoría de ellos son los mismos que teníamos que definir para los EMO que reducian la dimensionalidad. De nuevo pulsaremos el botón *Aceptar* para terminar el paso en el que nos encontramos.

Ya hemos creado los 30 multiclasificadores (10 particiones $\times$ 3 ejecuciones) y podemos ver los pesos de cada clasificador dentro de su multiclasificador, así como el error en *test* del multiclasificador. Navegaremos por los multiclasificadores de cada partición-ejecución utilizando las listas desplegables y el botón *Ver*.

Y por fin hemos terminado de crear el multiclasificador (que es el que menos error tenía de entre los 30 que visionábamos en la página anterior), llegando a la página final (figura 4.16). En dicha página se nos muestra tanto el error en *test* del multiclasificador final como

Figura 4.17: Diálogo del suministro del archivo del multiclasificador al sistema.

el número medio de características que posee. Además, si hemos elegido una base de datos por defecto del sistema podremos comparar con otras técnicas de selección de características mediante una tabla tricolor con *metáfora de semáforo*.

Lo más importante que tenemos en esta página, y que no podemos olvidar, es el archivo de información del multiclasificador para descargarlo en nuestro ordenador mediante el enlace *Click aquí para descargar fichero*. Este fichero es de tipo XML y posee un nombre aleatorio que podremos cambiar a nuestro antojo una vez lo tengamos en nuestro PC. Habrá que almacenarlo en algún lugar seguro ya que no podremos clasificar nuevas instancias en otra sesión si no suministramos este archivo al sistema.

Clasificación de nuevas instancias

Desde la página *Inicio* o desde la página final de creación del multiclasificador, si lo acabamos de crear, podemos pasar a la clasificación de nuevas instancias.

Desde la página de resultados del multiclasificador creado podremos hacer click en el botón *Clasificar nuevos datos con el multiclasificador creado* que nos redireccionará a una página que nos permite especificar un fichero con las nuevas instancias a clasificar. Dicho fichero tendrá que tener el mismo formato que los ficheros de la base de datos.

En el caso de que accedamos a la zona de clasificación desde *Inicio* tendremos que suministrar al sistema el fichero XML que definía el multiclasificador, más el fichero de entrenamiento de la partición sobre la que se creó el multiclasificador [2].

Tras haber especificado el fichero de nuevas instancias, el multiclasificador etc... el sistema estará en condiciones de clasificar las nuevas instancias, redireccionándonos a la página de resultados de clasificación (figura 4.18).

[2]Podemos ver el número de partición abriendo el fichero XML y consultando el campo *training*

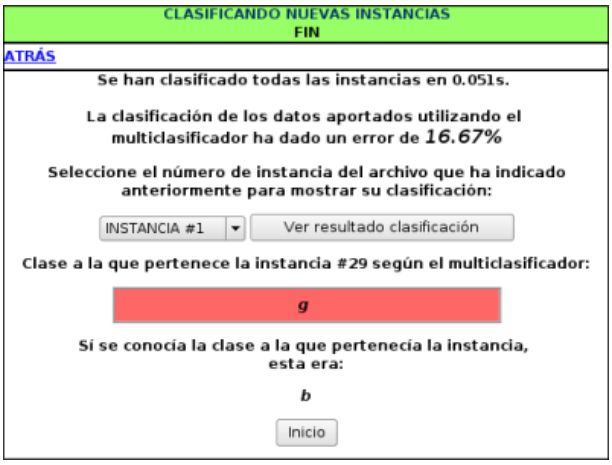

Figura 4.18: Pantalla del resultado de clasificación de nuevas instancias.

Dicha página muestra el error total al clasificar todas las instancias suministradas así como la clase de cada instancia (comparando con la clase real que tenía ésta, si se conocía). De nuevo una *metáfora de semáforo* nos indica si el multiclasificador ha acertado o no con la predicción.

4.4.2. Diagrama de clases conceptuales

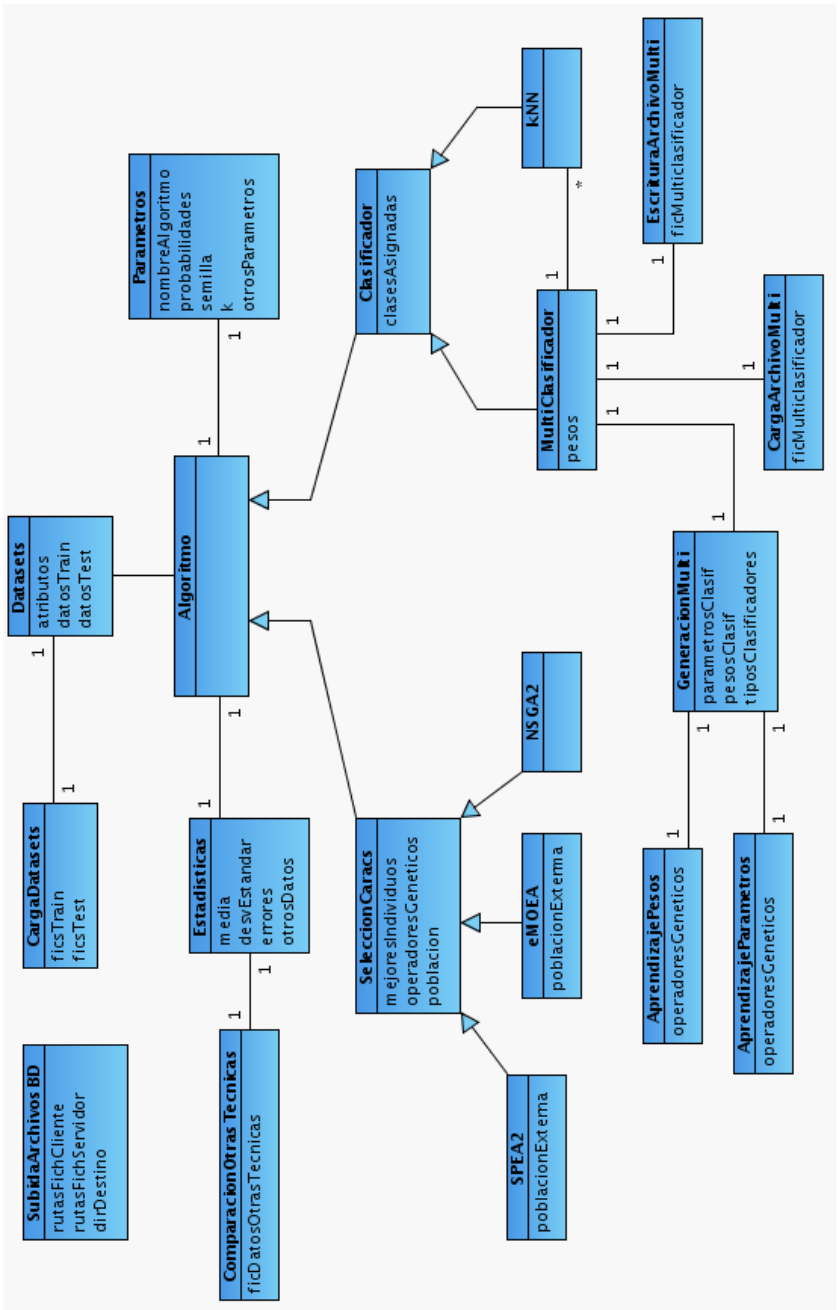

Figura 4.19: Diagrama de clases conceptuales.

4.4.3. Diagrama de colaboración

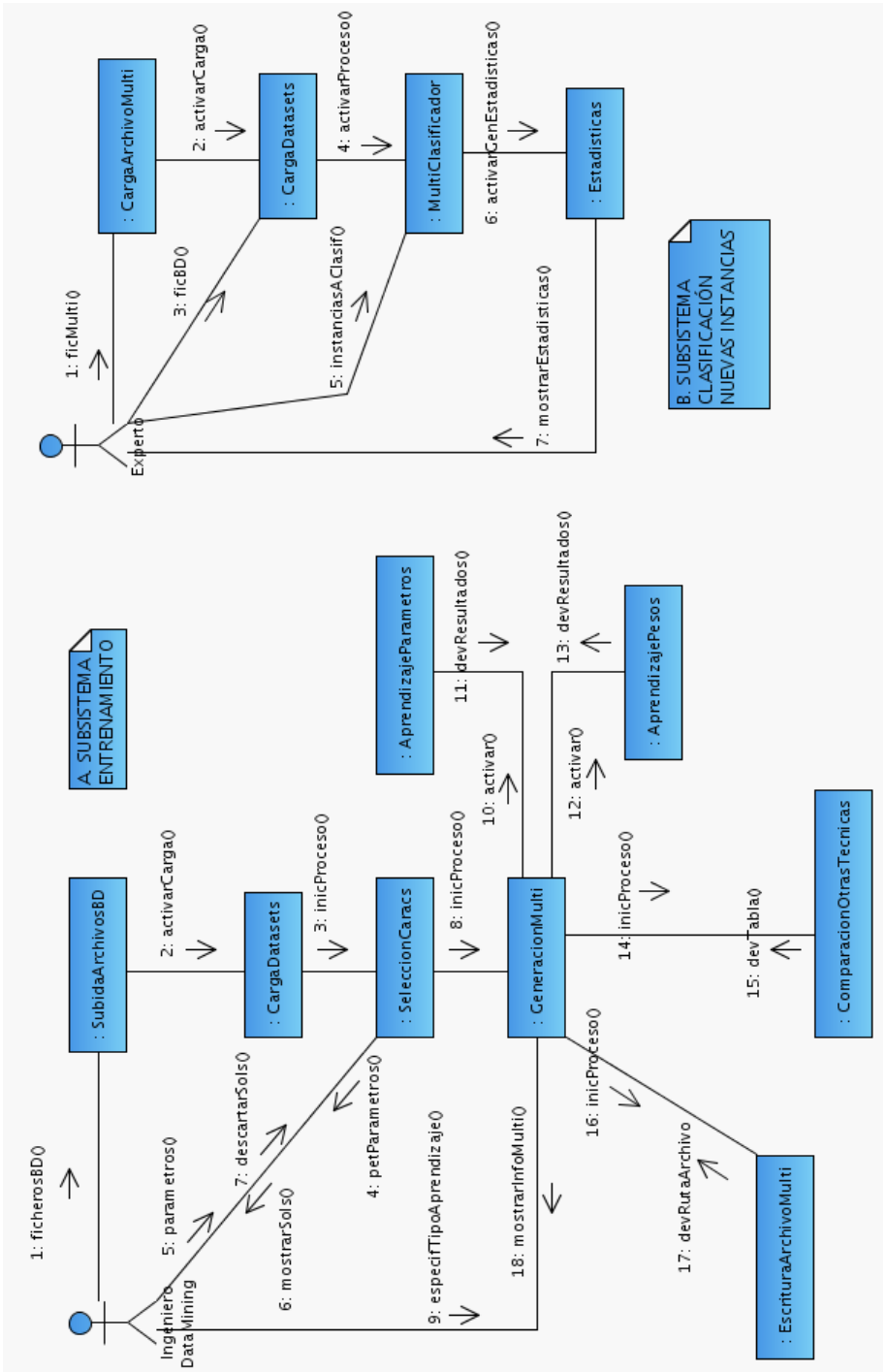

Figura 4.20: Diagrama de colaboración de los dos subsistemas.

4.5. Diseño de la Aplicación

4.5.1. Diagramas de paquetes y de clases

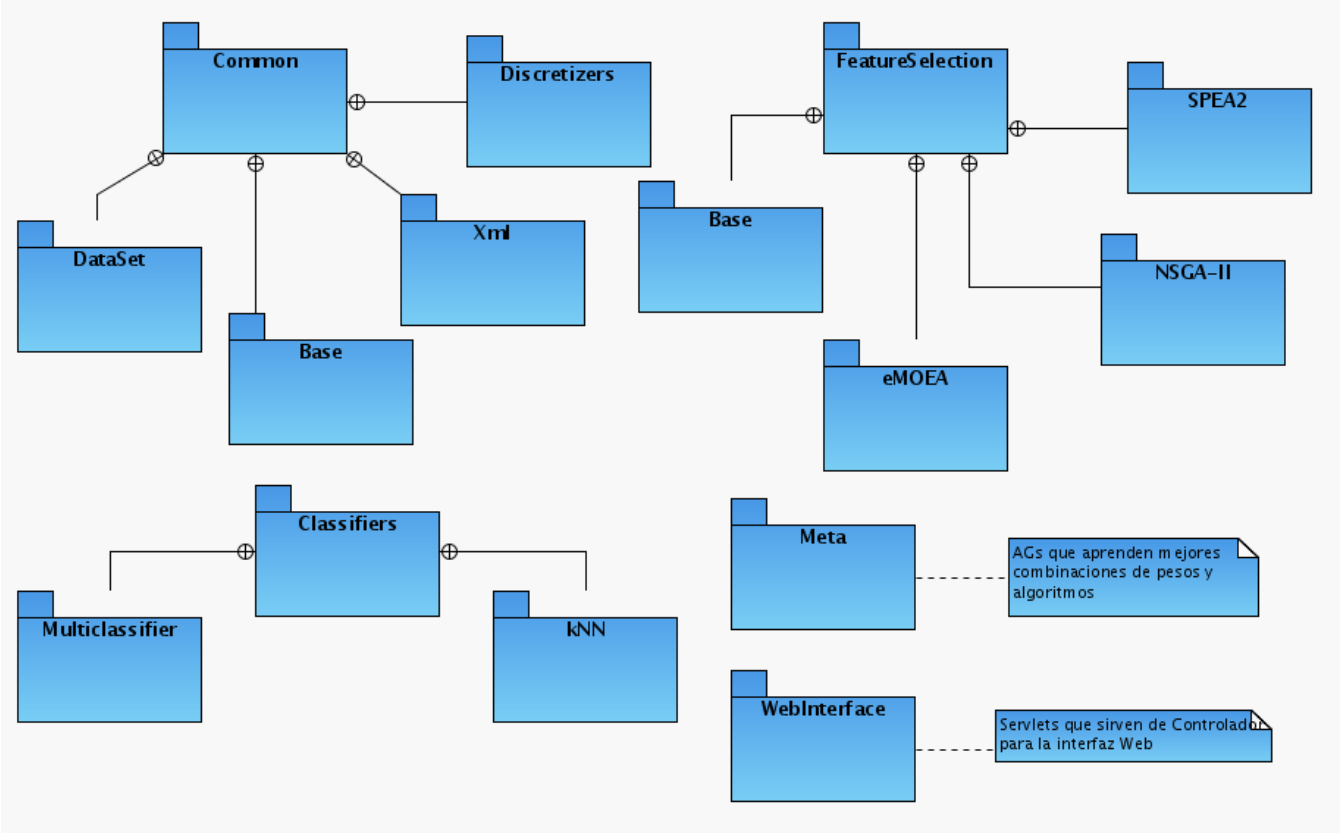

Figura 4.21: Diagrama de paquetes del sistema.

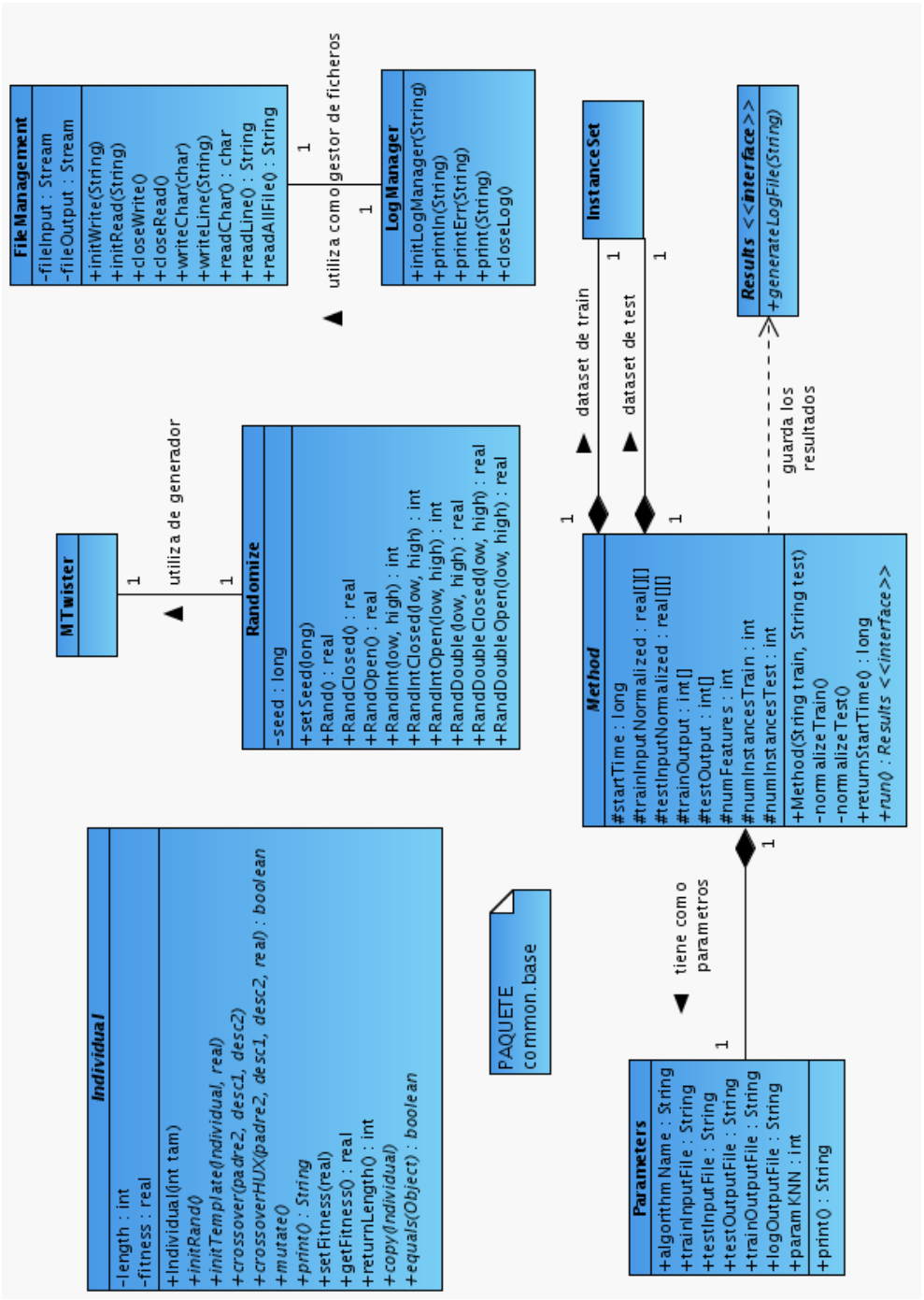

Figura 4.22: Diagrama de clases del paquete *common.base*.

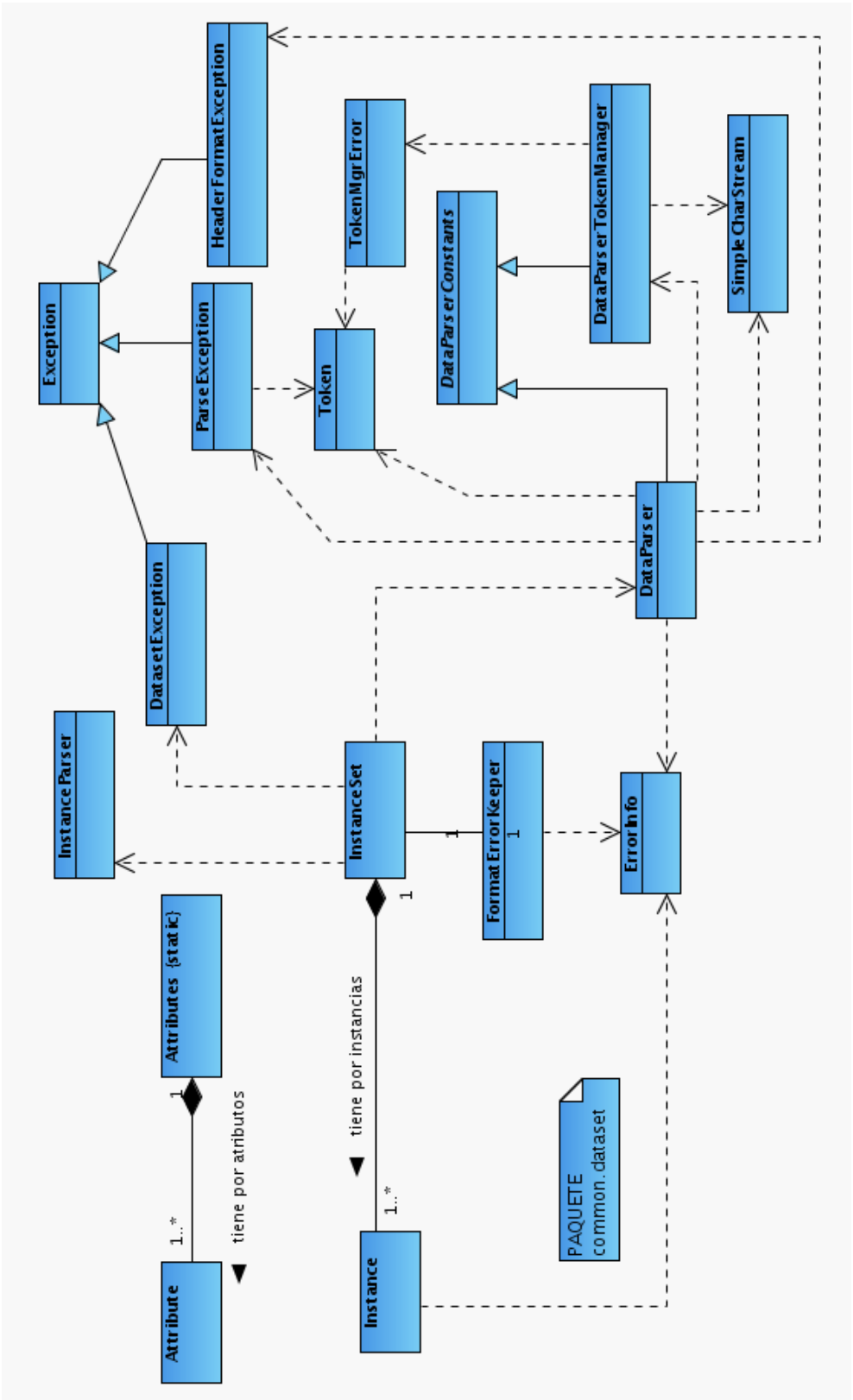

Figura 4.23: Diagrama de clases del paquete *common.dataset*.

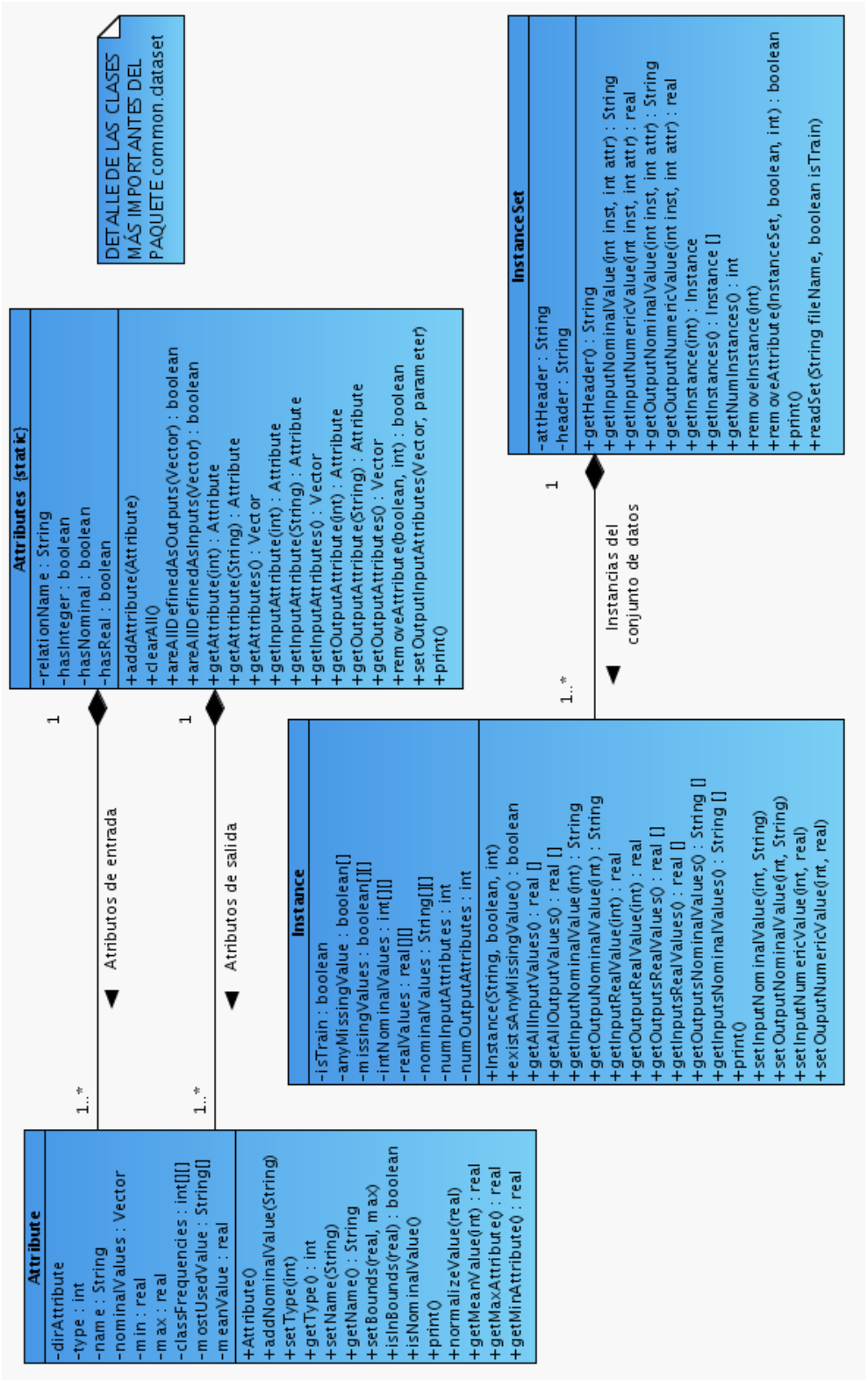

Figura 4.24: Detalle del diagrama de clases *common.dataset*.

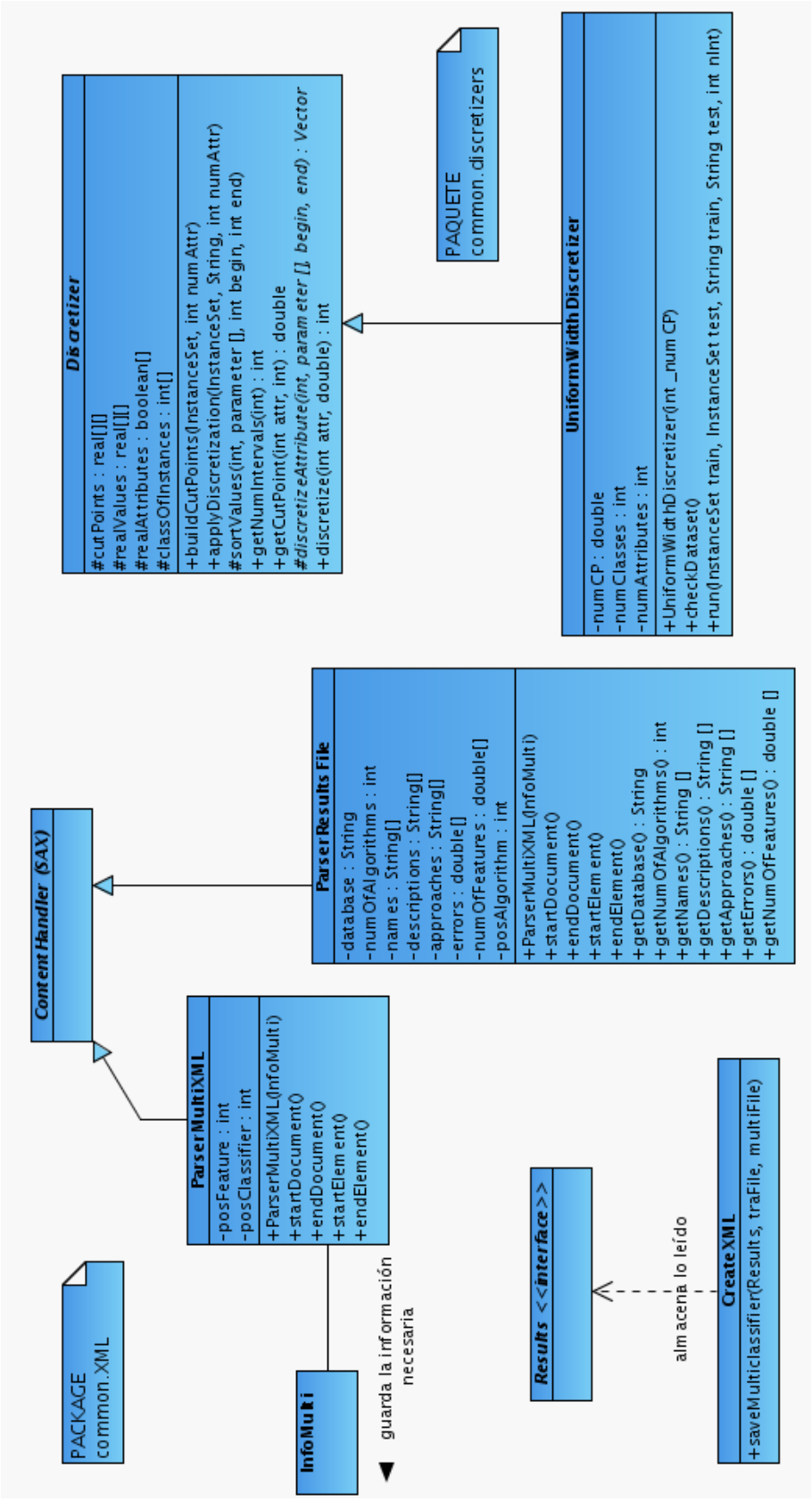

Figura 4.25: Diagramas de clases de *common.xml* y *common.discretizers*.

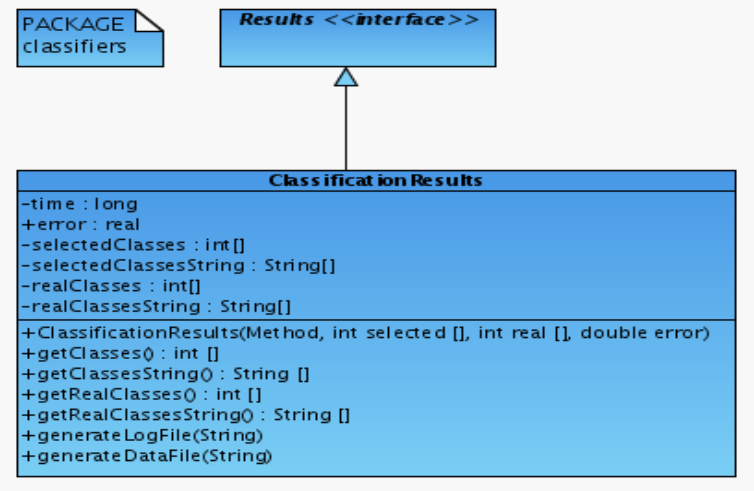

Figura 4.26: Diagramas de clases del paquete *classifiers*.

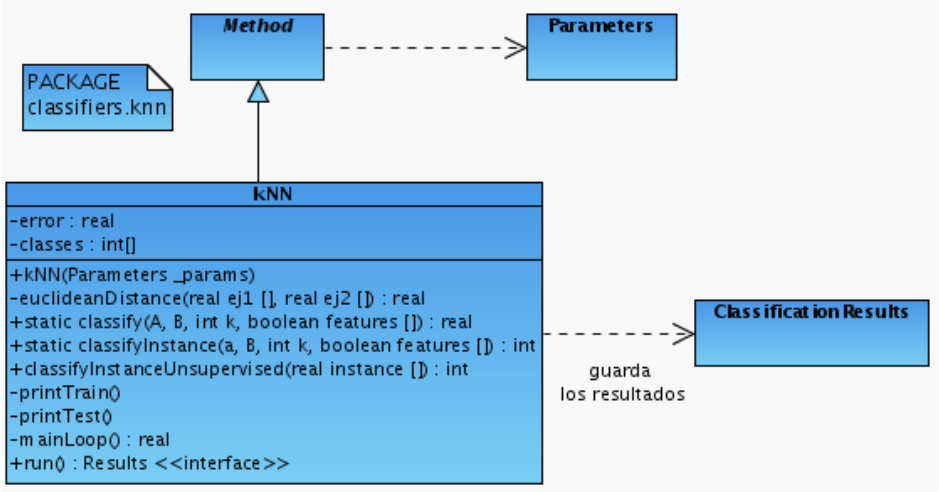

Figura 4.27: Diagramas de clases del paquete *classifiers.knn*.

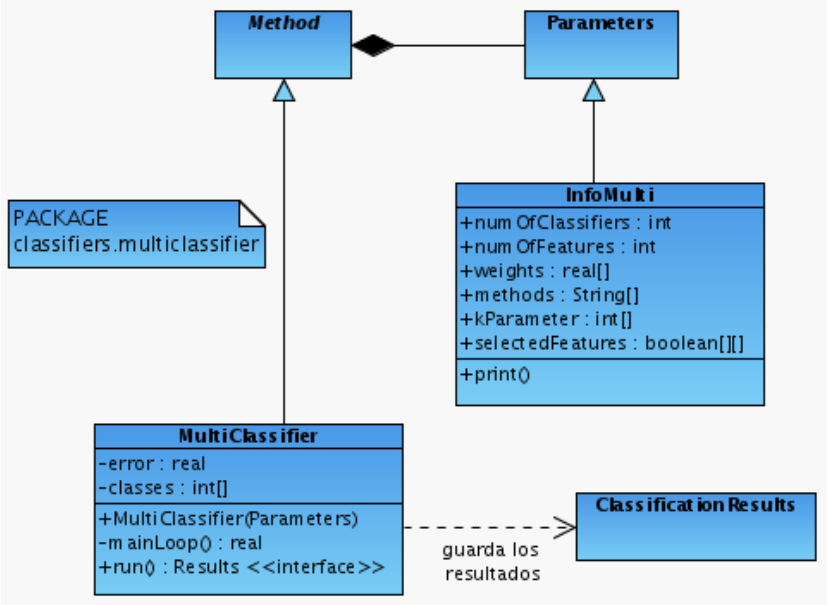

Figura 4.28: Diagramas de clases del paquete *classifiers.multiclassifiers*.

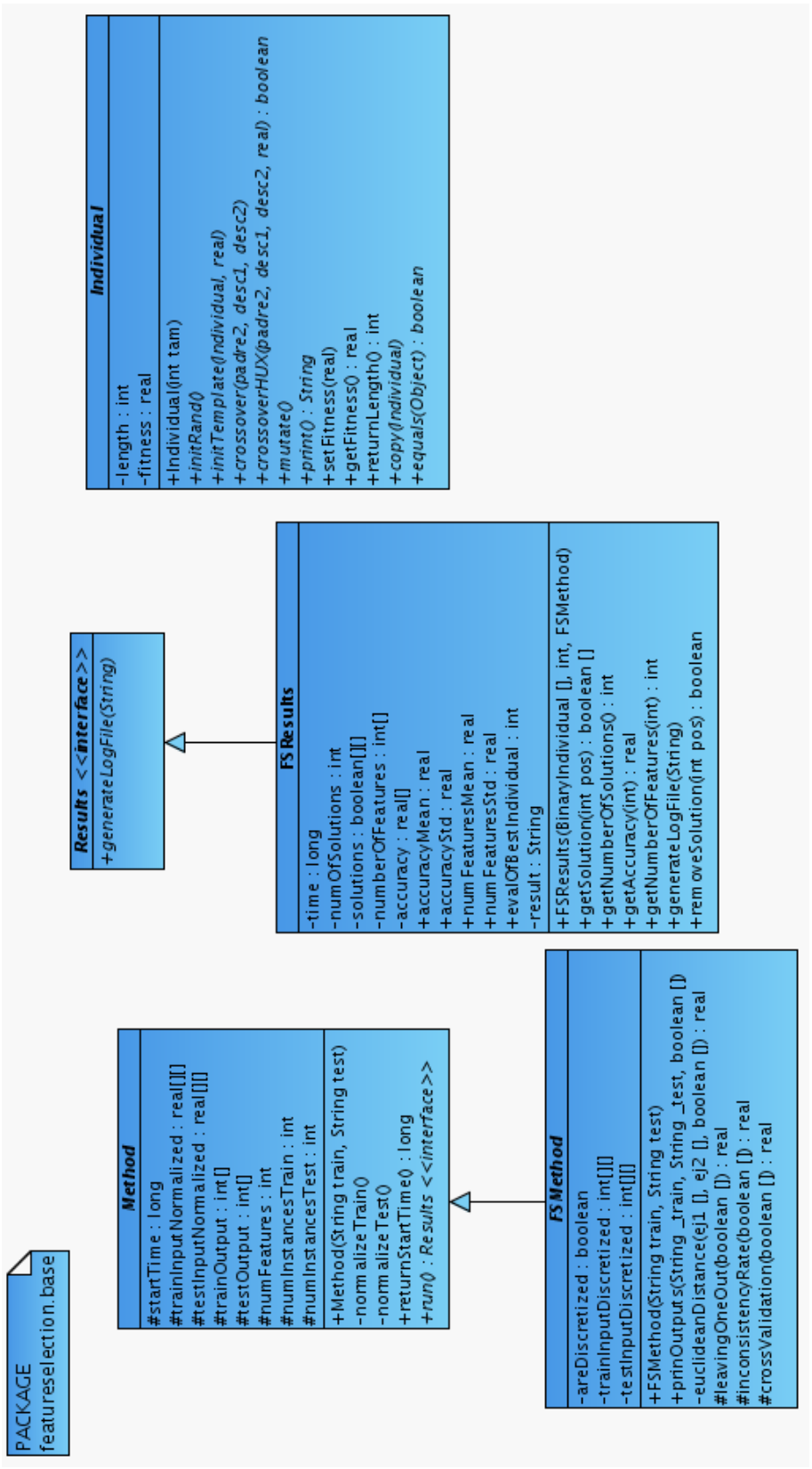

Figura 4.29: Diagramas de clases del paquete *featureselection.base*.

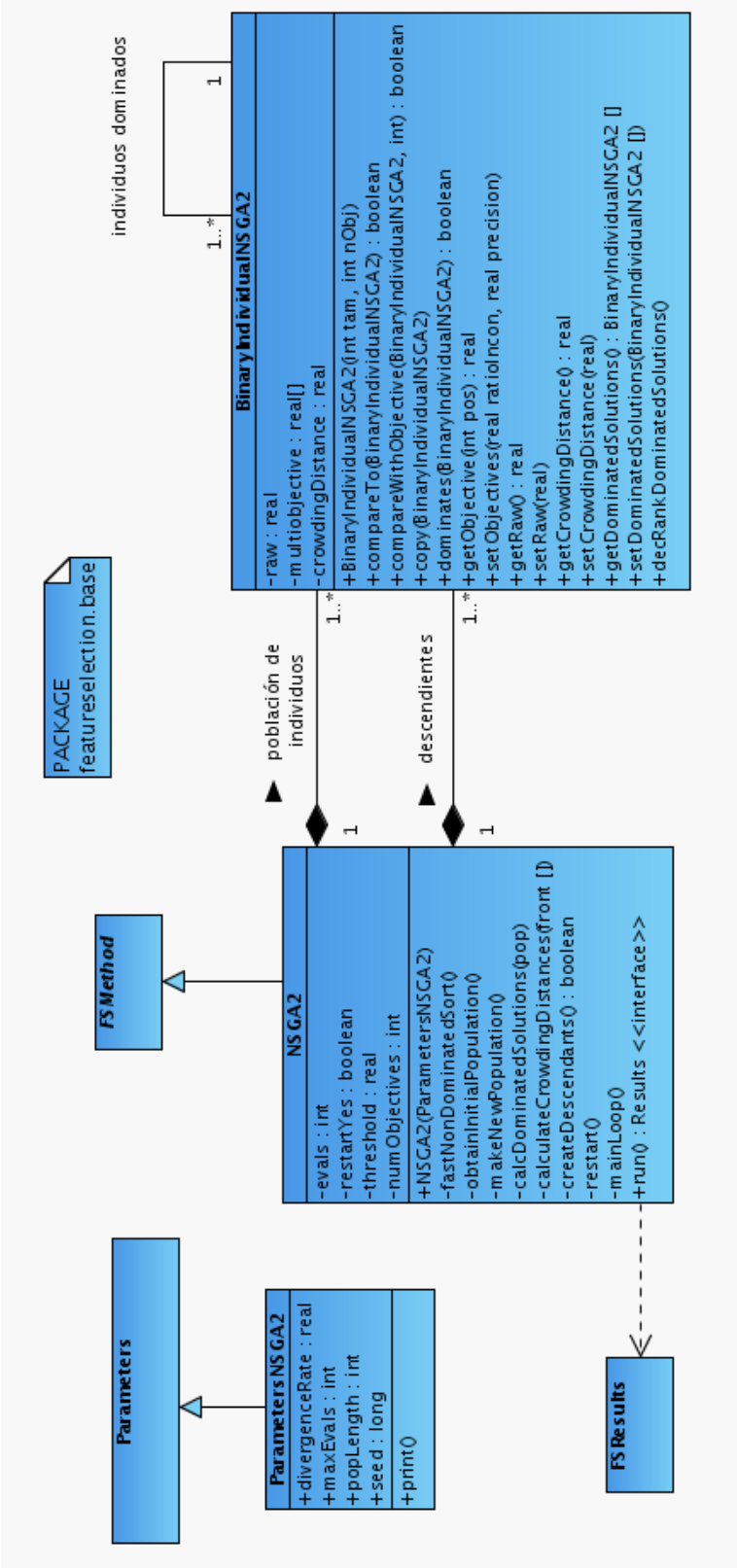

Figura 4.30: Diagramas de clases del paquete *featureselection.nsga2*.

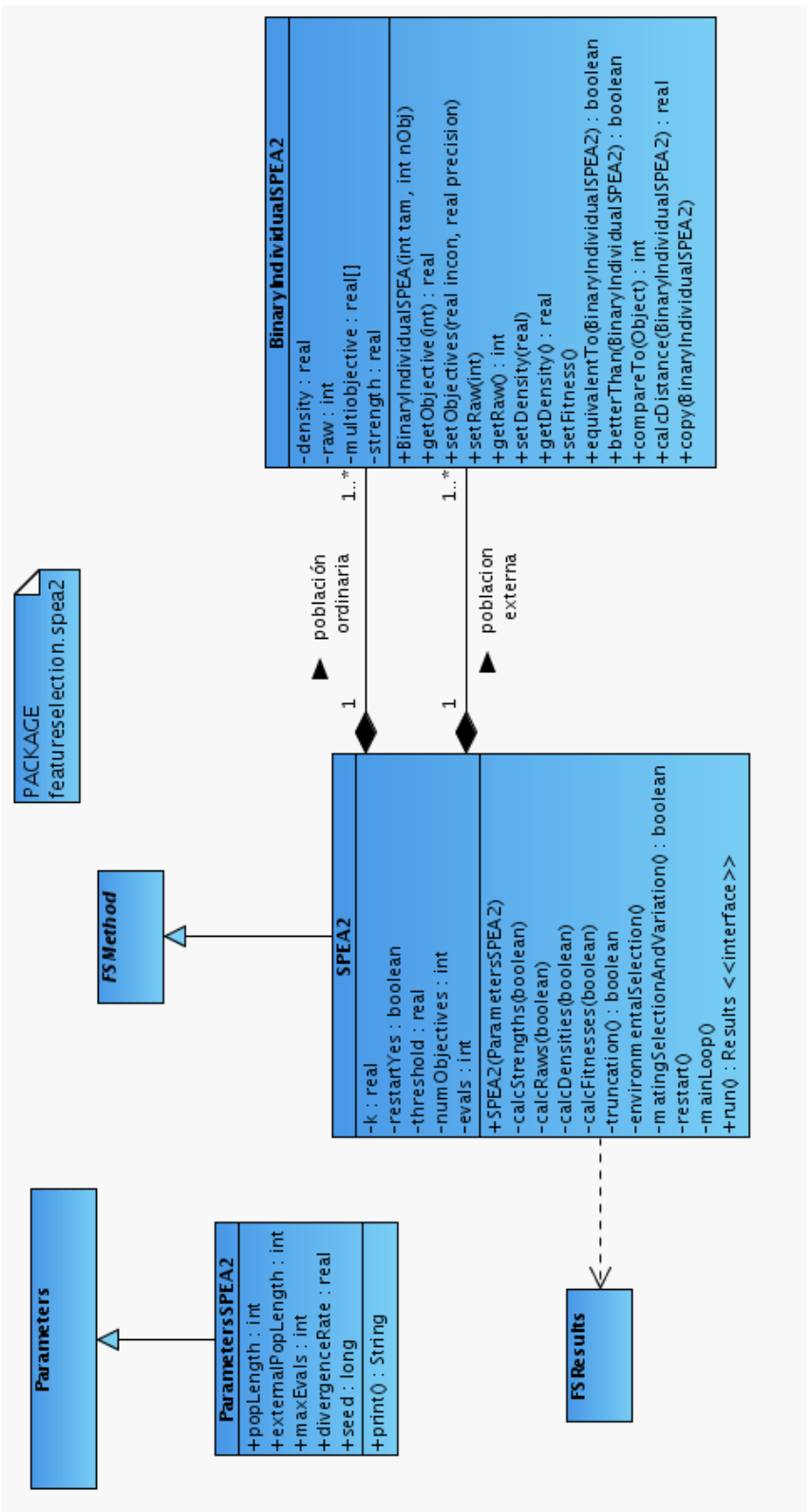

Figura 4.31: Diagramas de clases del paquete *featureselection.spea2*.

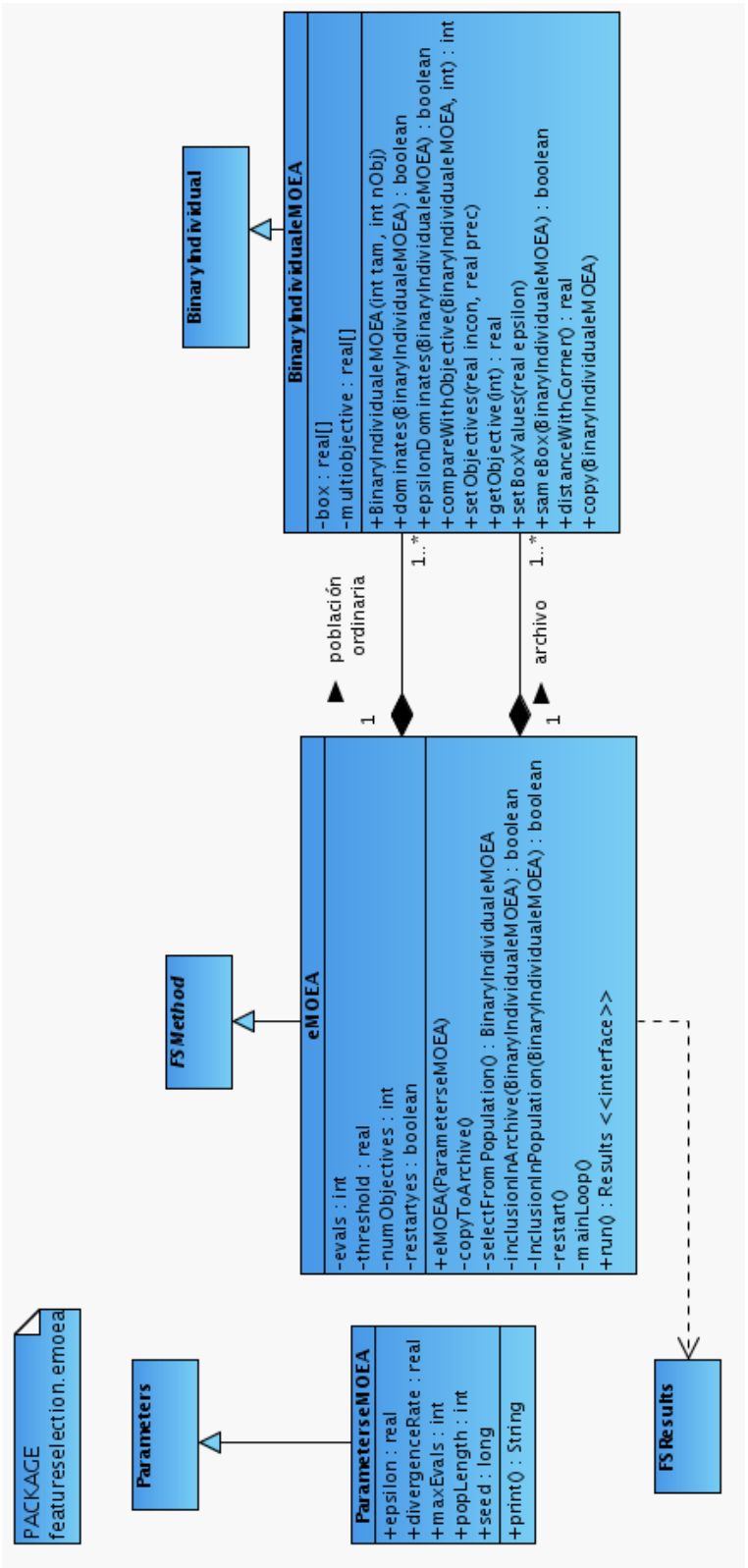

Figura 4.32: Diagramas de clases del paquete *featureselection.emoea*.

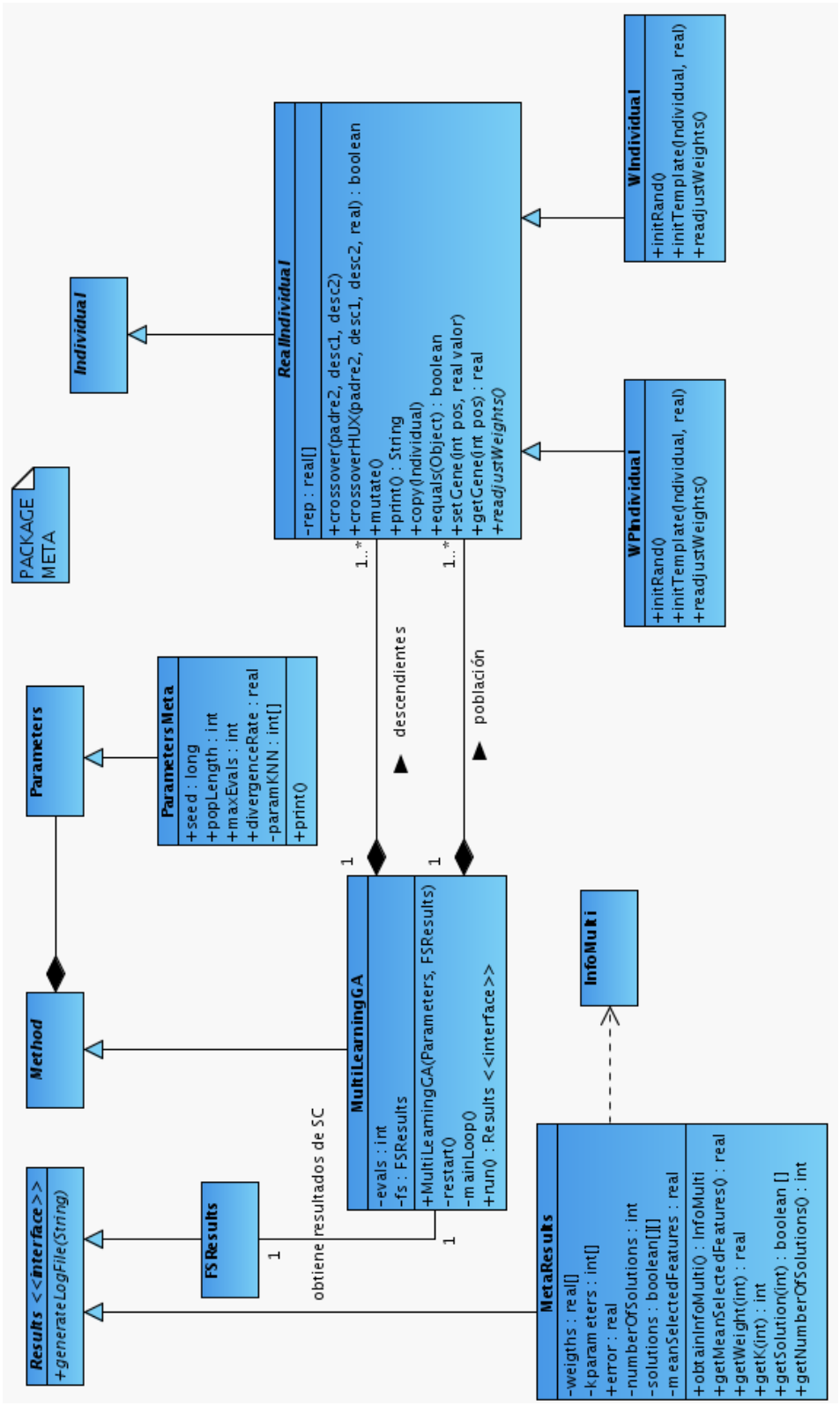

Figura 4.33: Diagramas de clases del paquete *meta*.

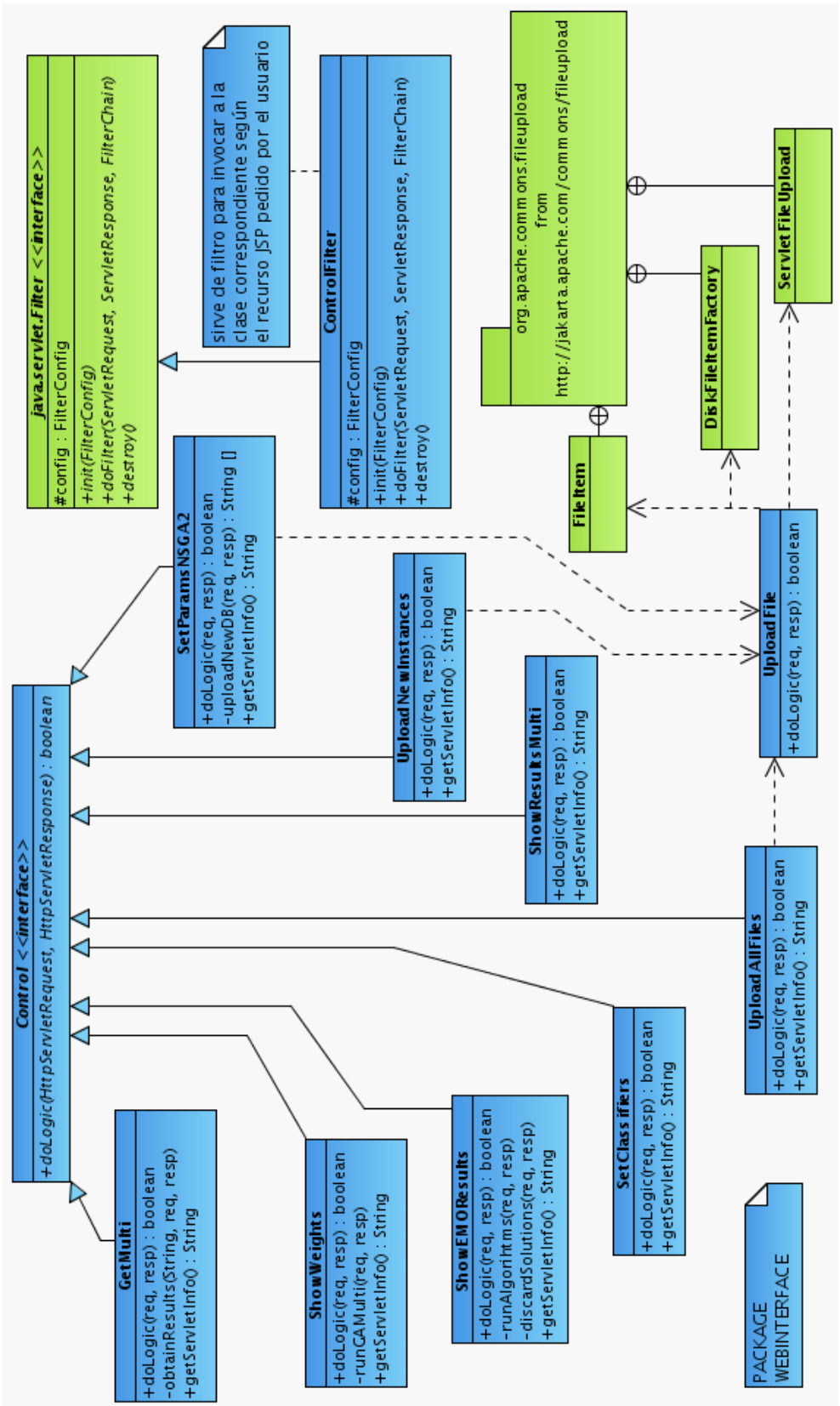

Figura 4.34: Diagramas de clases del paquete *webinterface*.

Capítulo 5

Experimentación y análisis de resultados

5.1. Organización de la experimentación

La experimentación y análisis de resultados que vamos a realizar en este capítulo está dividida en dos bloques, que se corresponden con los dos grandes objetivos del *Proyecto*:

- Experimentación con algoritmos de selección de características.
- Resultados del sistema multiclasificador.

Por tanto, el capítulo va a contener las siguientes secciones:

1. Experimentación y análisis para selección de características.
   a) Comparación de los algoritmos híbridos y estándar.
   b) EMO híbridos para cada problema.
   c) Comparación de los EMO híbridos frente a otras técnicas de selección de características.
2. Resultados y análisis del multiclasificador evolutivo.

Las tablas con los resultados más importantes y referidos en el texto se encuentran presentes en este capítulo. Algunas tablas como en las que se recogen el número de evaluaciones necesarias para obtener la solución en los distintos algoritmos, o los resultados por partición de otras técnicas de SC ajenas al *Proyecto* se hayan en el apéndice 3.

Las tablas de experimentación muestran los resultados de los algoritmos **híbridos** y **estándar**. Las notaciones y abreviaturas utilizadas se describen a continuación:

- **TODAS**: error en test clasificando con el algoritmo **1NN**, teniendo en cuenta todas las características iniciales del problema, es decir, sin haber aplicado reducción de la dimensionalidad.
- $\overline{x}_{prec}$: media del error en *test*, clasificando con el algoritmo **1NN**, de todas las soluciones devueltas por el algoritmo multiobjetivo.
- σ_{prec}: desviación típica del error en *test*, clasificando con el algoritmo **1NN**, de todas las soluciones devueltas por el algoritmo multiobjetivo.
- $\overline{x}_{caracs}$: media del número de características de las soluciones devueltas por el algoritmo multiobjetivo.
- σ_{caracs}: desviación típica del número de características de las soluciones devueltas por el algoritmo multiobjetivo.
- **MEJOR PREC.**: mejor error en *test* de entre todas las soluciones devueltas por el EMO.
- **CARACS MEJOR.**: número de características de la solución que obtenía el mejor error en test.
- **MEJOR CARACS.**: la mejor reducción de dimensionalidad obtenida por una de las soluciones devueltas por el EMO.
- **PREC. MEJOR**: error en *test* de la solución del EMO que obtenía la mejor reducción de características.

También se mostrarán tablas de resultados que contienen información respecto a los multiclasificadores creados. Las abreviaturas que encontramos en éstas son las siguientes:

- **TODAS**: error en test clasificando con el algoritmo **1NN**, teniendo en cuenta todas las características iniciales del problema.
- $\overline{x}_{prec}$: media del error en *test*, clasificando con el algoritmo **1NN**, de todas las soluciones devueltas por el mejor algoritmo multiobjetivo de entre los estudiados.
- **ERROR TEST**: error en *test* del multiclasificador generado a partir de las soluciones del mejor EMO.
- **CLASIF. INICIALES**: número de clasificadores iniciales que poseía el multiclasificador, que es igual al número de conjuntos de características devueltos por el mejor EMO.
- **CLASIF. FINALES**: número de clasificadores finales que posee el multiclasificador tras haber realizado una selección y eliminado los clasificadores que menos influían en la clasificación.

5.2. Descripción de las BBDD utilizadas

A lo largo de todo el proceso de toma de resultados y de análisis hemos estado utilizando cuatro bases de datos obtenidas del **Proyecto KEEL** y del almacén de datos de la **Universidad de California at Irvine** [1] con las cuáles hemos ido probando el rendimiento de los algoritmos. Reseñamos sus nombres y sus características más importantes:

- **Base de datos IONOSPHERE** trata de clasificar distintas pruebas realizadas con radares, viendo si la estructura que se detecta pertenece o no a la capa ionosfera. Tiene 351 instancias y 34 variables más la clase (dos posibles valores nominales que indica si la prueba pertenece o no a la ionosfera).
- **Base de datos WISCONSIN** que almacena datos de pacientes con cáncer del estado americano de Wisconsin. Con 570 instancias y 30 variables más la clase, la cuál posee 2 posibles valores: *malingo* y *benigno*.
- **Base de datos VEHICLE** que contiene información técnica sobre las características de vehículos. La forman 846 instancias y 18 variables, pertenecientes a una de las 4 posibles clases. El problema consiste en clasificar una silueta dada de un vehículo entre cuatro posibles: *saab*, *van*, *opel* y *bus*.

  Nosotros hemos trabajado con una transformación de esta base de ejemplos, en la que el número final de clases queda reducido a dos: *van* (23.52 %) y el resto (76.48 %).
- **Base de datos SPLICE-IE** de temática médico-genética y formada por 3176 instancias y 60 variables más la clase (valores *positivo* y *negativo*). Una base de datos de tamaño superior a las anteriores para ver cómo responden los algoritmos ante BBDD de mayor entidad.

5.3. Experimentación y análisis para selección de características

5.3.1. Comparación de los algoritmos híbridos y estándar

Los parámetros utilizados en los algoritmos **EMO estándar** son los siguientes:

- Tamaño de la población: 100
- Tamaño de la población externa en SPEA2: 100
- Epsilon para algoritmo ϵ-MOEA: 0.01

[1] Repositorio UCI: `http://www.ics.uci.edu/~mlearn/MLRepository.html`

- Número máximo de evaluaciones: 20000
- Probabilidad de cruce: 0.8
- Probabilidad de mutación: 0.125 respecto al número de cromosomas

Mientras que los utilizados en los **EMO híbridos** han sido los siguientes:

- Tamaño de la población: 100
- Tamaño de la población externa en **SPEA2**: 100
- Epsilon para algoritmo ϵ-**MOEA**: 0.01
- Número máximo de evaluaciones: 20000
- Ratio de divergencia: 0.35

Queríamos comparar las dos versiones desarrolladas aplicadas a estos cuatro problemas, obteniendo resultados sobre todas las soluciones devueltas del frente de Pareto ($\overline{x}_{prec}$, σ_{prec}, $\overline{x}_{caracs}$ y σ_{caracs}), valores de la mejor solución del frente respecto a la precisión obtenida y al menor número de características, así como el número de evaluaciones necesarias para obtener al individuo con mejor precisión en *training* (que no tiene porqué ser el mejor individuo con precisión en *test*).

	IONOSPHERE		VEHICLE		WISCONSIN	
	HIB	**STD**	**HIB**	**STD**	**HIB**	**STD**
NSGA-II	9925,9	2491,3	2384,6	1711,6	7413,6	3502,9
SPEA2	7172,5	2371,2	3646,9	1524,8	7551,9	1614,7
ϵ-MOEA	8884,5	278,5	7292,7	161,2	3749,3	376,3

Tabla 5.1: Número medio de evaluaciones necesarias para conseguir al individuo con mejor precisión en *training*

El primer dato que vamos a analizar es el número de evaluaciones. La tabla 5.1 nos recoge la media de evaluaciones necesarias para cada tipo de algoritmo (*HIB*: versión híbrida, *STD*: versión estándar). Vemos claramente como **el algoritmo ϵ-MOEA estándar tiene problemas de convergencia prematura**, obteniendo el mejor individuo tan sólo en la segunda o tercera generación. Los demás algoritmos estándar no poseen convergencia prematura, pero cabe destacar que consiguen el mejor individuo en generaciones mucho más tempranas que los algoritmos híbridos. Con estos algoritmos estamos desperdiciando tiempo de computación al haber establecido la condición de parada en 20000 evaluaciones, ya que en la mayor parte de estas no se estaría consiguiendo nada productivo. Esto significa que **los algoritmos estándar exploran peor el campo de soluciones**, mientras que **los**

algoritmos híbridos consiguen una exploración más amplia, obteniendo soluciones mejores como veremos ahora en las tablas de resultados de este capítulo.

También apreciamos cómo todos los algoritmos necesitan menos de 20.000 evaluaciones para resolver los problemas **WISCONSIN**, **IONOSPHERE** y **VEHICLE**. Este número de evaluaciones se debe fijar de acuerdo a la dimensionalidad del problema al que nos enfrentamos.

Vamos a empezar analizando los resultados obtenidos por las dos implementaciones del **NSGA-II** respecto a los cuatro problemas que tenemos. Hemos dividido los resultados en dos tipos de tablas. En la primera de ellas mostramos las medias y desviaciones típicas de todas las soluciones del Pareto devueltas por los algoritmos estándar e híbrido para cada problema (tablas 5.2, 5.3, 5.4 y 5.5). El segundo tipo de tablas nos permite ver las mejores soluciones por frente tanto en número de características seleccionadas como en precisión en test (tablas 5.6, 5.7, 5.8 y 5.9).

Los resultados son claros, **tanto la versión estándar del algoritmo como la híbrida mejoran la precisión alcanzada clasificando con todas las características**. Sin embargo, **el algoritmo híbrido suele seleccionar tres veces menos características que el estándar** aunque su precisión media está un poco por debajo de la precisión media del NSGA-II estándar.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	11,11 %	15,97 %	0,06	5	0,82	8,33 %	0	10,5	0,71
PART. 2	17,14 %	6,43 %	0,05	5,75	0,5	12,38 %	0,02	13	1
PART. 3	13,89 %	14,44 %	0,04	5,2	0,84	12,50 %	0,02	10	1,41
PART. 4	11,76 %	9,80 %	0,02	4,67	1,15	5,88 %	0	9,5	0,71
PART. 5	2,94 %	8,82 %	0	5	0	2,94 %	0	13,5	0,71
PART. 6	22,86 %	12,38 %	0,02	5	1	5,71 %	0	10,5	0,71
PART. 7	17,14 %	14,29 %	0,05	4,33	0,58	11,43 %	0	15,67	0,58
PART. 8	13,89 %	13,89 %	0	5,33	0,58	13,89 %	0	10,5	0,71
PART. 9	11,76 %	8,82 %	0,04	5,25	0,5	5,88 %	0	13	0
PART. 10	11,11 %	11,11 %	0,03	4,67	0,58	13,89 %	0	15,5	0,71
MEDIA	13,36 %	11,60 %	0,03	5,02	0,65	9,28 %	0	12,17	0,72

Tabla 5.2: Soluciones devueltas por el NSGA-II híbrido y estándar para IONOSPHERE.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	2,35 %	4,28 %	0,03	4,73	1,1	1,18 %	0	11,67	0,58
PART. 2	8,24 %	4,26 %	0,02	4,63	1,09	2,35 %	0	10	0
PART. 3	4,71 %	3,10 %	0,01	4,73	1,27	0,00 %	0	11,5	0,58
PART. 4	4,71 %	3,82 %	0,01	4,75	1,14	3,53 %	0,01	11	1,41
PART. 5	7,06 %	5,51 %	0,02	4,56	1,03	3,82 %	0,01	11,25	1,71
PART. 6	7,06 %	3,06 %	0,02	4,4	1,07	1,18 %	0	11,67	0,58
PART. 7	7,14 %	4,29 %	0,02	4,53	1,06	4,76 %	0,04	10	0,71
PART. 8	3,57 %	4,40 %	0,02	4,6	0,97	1,98 %	0,01	11	1
PART. 9	7,14 %	4,30 %	0,02	4,38	1,12	1,59 %	0,01	11,67	1,15
PART. 10	9,52 %	8,18 %	0,01	4,5	1,2	3,57 %	0	11,5	0,71
MEDIA	6,15 %	4,52 %	0,02	4,58	1,1	2,40 %	0,01	11,13	0,84

Tabla 5.3: Soluciones devueltas por el NSGA-II híbrido y estándar para VEHICLE.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	1,75 %	4,09 %	0,01	4,17	0,98	5,26 %	0	15,75	0,96
PART. 2	1,75 %	3,70 %	0,02	4	1	1,75 %	0	15,6	1,14
PART. 3	0,00 %	1,54 %	0,01	4,25	1,04	0,00 %	0	14	0
PART. 4	3,51 %	3,51 %	0,02	4,22	0,97	1,75 %	0	16,5	0,71
PART. 5	5,26 %	7,31 %	0,02	4,17	0,75	3,51 %	0	16,5	0,71
PART. 6	5,26 %	3,24 %	0,02	4,62	1,04	5,26 %	0	15	0
PART. 7	3,51 %	5,26 %	0,02	4,63	1,3	3,51 %	0	12,83	1,17
PART. 8	7,02 %	5,13 %	0,01	4,15	0,69	5,26 %	0	19	0
PART. 9	10,53 %	9,65 %	0,03	4,17	0,75	7,02 %	0	16,67	0,58
PART. 10	10,71 %	5,06 %	0,02	4,17	1,17	10,71 %	0	14	1
MEDIA	4,93 %	4,85 %	0,02	4,25	0,97	4,40 %	0	15,58	0,63

Tabla 5.4: Soluciones devueltas por el NSGA-II híbrido y estándar para WISCONSIN.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	17,61 %	9,39 %	0,02	7,52	1,91	16,98 %	0	23	0
PART. 2	20,44 %	9,49 %	0,02	8,48	1,12	13,52 %	0	27	0
PART. 3	19,81 %	8,87 %	0,02	8,15	1,63	15,09 %	0	30	0
PART. 4	18,24 %	10,75 %	0,01	8,43	1,44	18,87 %	0	27	0
PART. 5	16,04 %	9,20 %	0,01	8,53	1,60	13,52 %	0	28	0
PART. 6	22,08 %	8,14 %	0,01	7,88	1,87	16,09 %	0	18	0
PART. 7	21,77 %	10,33 %	0,02	8,07	1,49	18,93 %	0	27	0
PART. 8	16,40 %	8,25 %	0,02	8,19	1,33	16,72 %	0	30	0
PART. 9	17,67 %	8,87 %	0,02	8,11	1,32	18,93 %	0	23	0
PART. 10	19,56 %	9,17 %	0,02	7,88	1,78	14,20 %	0	23	0
MEDIA	18,96 %	9,24 %	0,02	8,12	1,55	16,28 %	0	25,6	0

Tabla 5.5: Soluciones devueltas por el NSGA-II híbrido y estándar para SPLICE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	11,11 %	8,33 %	5	4	19,44 %	8,33 %	11	10	8,33 %
PARTICIÓN 2	17,14 %	2,86 %	6	5	2,86 %	11,43 %	14	12	14,29 %
PARTICIÓN 3	13,89 %	11,11 %	6	4	11,11 %	11,11 %	9	9	11,11 %
PARTICIÓN 4	11,76 %	8,82 %	6	4	8,82 %	5,88 %	9	9	5,88 %
PARTICIÓN 5	2,94 %	8,82 %	5	5	8,82 %	2,94 %	14	13	2,94 %
PARTICIÓN 6	22,86 %	11,43 %	5	4	14,29 %	5,71 %	10	10	5,71 %
PARTICIÓN 7	17,14 %	11,43 %	5	4	11,43 %	11,43 %	15	15	11,43 %
PARTICIÓN 8	13,89 %	13,89 %	6	5	13,89 %	13,89 %	11	10	13,89 %
PARTICIÓN 9	11,76 %	2,94 %	6	5	11,76 %	5,88 %	13	13	5,88 %
PARTICIÓN 10	11,11 %	8,33 %	4	4	8,33 %	13,89 %	16	15	13,89 %
MEDIA	13,36 %	8,80 %	5,40	4,40	11,08 %	9,05 %	12,20	11,60	9,34 %

Tabla 5.6: Mejores soluciones del NSGA-II híbrido y estándar para IONOSPHERE.

	TODAS	VERSIÓN HÍBRIDA			VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	2,35 %	1,18 %	3	3	10,59 %	1,18 %	12	11	1,18 %
PARTICIÓN 2	8,24 %	1,18 %	5	3	8,24 %	2,35 %	10	10	2,35 %
PARTICIÓN 3	4,71 %	1,18 %	4	3	4,71 %	0,00 %	12	11	0,00 %
PARTICIÓN 4	4,71 %	1,18 %	6	3	3,53 %	2,35 %	12	9	3,53 %
PARTICIÓN 5	7,06 %	2,35 %	6	3	4,71 %	2,35 %	11	9	5,88 %
PARTICIÓN 6	7,06 %	1,18 %	6	3	5,88 %	1,18 %	12	11	1,18 %
PARTICIÓN 7	7,14 %	1,19 %	4	3	2,38 %	1,19 %	10	9	9,52 %
PARTICIÓN 8	3,57 %	2,38 %	5	3	8,33 %	1,19 %	12	10	2,38 %
PARTICIÓN 9	7,14 %	1,19 %	4	3	8,33 %	1,19 %	11	11	2,38 %
PARTICIÓN 10	9,52 %	5,95 %	4	3	9,52 %	3,57 %	11	11	3,57 %
MEDIA	6,15 %	1,89 %	4,7	3,0	6,62 %	1,66 %	11,3	10,2	3,20 %

Tabla 5.7: Mejores soluciones del NSGA-II híbrido y estándar para VEHICLE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	1,75 %	3,51 %	5	3	5,26 %	5,26 %	16	15	5,26 %
PARTICIÓN 2	1,75 %	1,75 %	4	2	8,77 %	1,75 %	16	14	1,75 %
PARTICIÓN 3	0,00 %	0,00 %	4	3	1,75 %	0,00 %	14	14	0,00 %
PARTICIÓN 4	3,51 %	1,75 %	4	3	1,75 %	1,75 %	16	16	1,75 %
PARTICIÓN 5	5,26 %	5,26 %	5	3	5,26 %	3,51 %	17	16	3,51 %
PARTICIÓN 6	5,26 %	0,00 %	3	3	1,75 %	5,26 %	15	15	5,26 %
PARTICIÓN 7	3,51 %	1,75 %	4	3	5,26 %	3,51 %	14	11	3,51 %
PARTICIÓN 8	7,02 %	3,51 %	4	3	3,51 %	5,26 %	19	19	5,26 %
PARTICIÓN 9	10,53 %	7,02 %	5	3	10,53 %	7,02 %	16	16	7,02 %
PARTICIÓN 10	10,71 %	3,57 %	5	3	5,36 %	10,71 %	14	13	10,71 %
MEDIA	4,93 %	2,81 %	4,3	2,9	4,92 %	4,40 %	15,7	14,9	4,40 %

Tabla 5.8: Mejores soluciones del NSGA-II híbrido y estándar para WISCONSIN.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	17,61 %	6,92 %	5	3	9,12 %	16,98 %	23	23	16,98 %
PARTICIÓN 2	20,44 %	6,60 %	8	6	7,86 %	13,52 %	27	27	13,52 %
PARTICIÓN 3	19,81 %	5,97 %	8	4	7,55 %	15,09 %	30	30	15,09 %
PARTICIÓN 4	18,24 %	8,81 %	7	5	9,75 %	18,87 %	27	27	18,87 %
PARTICIÓN 5	16,04 %	6,60 %	9	5	7,23 %	13,52 %	28	28	13,52 %
PARTICIÓN 6	22,08 %	5,99 %	6	3	7,26 %	16,09 %	18	18	16,09 %
PARTICIÓN 7	21,77 %	6,94 %	6	5	8,52 %	18,93 %	27	27	18,93 %
PARTICIÓN 8	16,40 %	5,36 %	6	6	5,99 %	16,72 %	30	30	16,72 %
PARTICIÓN 9	17,67 %	6,31 %	8	5	6,62 %	18,93 %	23	23	18,93 %
PARTICIÓN 10	19,56 %	6,62 %	6	4	8,52 %	14,20 %	23	23	14,20 %
MEDIA	18,96 %	6,61 %	6,9	4,6	7,84 %	16,28 %	25,6	25,6	16,28 %

Tabla 5.9: Mejores soluciones del NSGA-II híbrido y estándar para SPLICE.

Para el problema **SPLICE**, que como sabemos es un problema de mayor complejidad que los otros tres, se obtienen resultados diferentes. El **algoritmo híbrido obtiene mucha mejor precisión que el algoritmo NSGA-II estándar** (ver tabla 5.5 y 5.9) tanto en media (9 % de error respecto a 16 %) como en mejor solución (6,61 % respecto a 16 %). Y no sólo eso, **se suelen seleccionar sólo 8 características de las 60** (se consigue una reducción de la dimensionalidad de más del 85 %), mientras que el algoritmo estándar selecciona una media de 25.

Luego está claro que **cuanto mayor sea el problema mejor responde la versión híbrida respecto a la estándar** y mayores son los beneficios que nos proporciona el algoritmo de selección de características respecto a reducción y precisión.

Esta diferencia de resultados entre uno y otro tipo de algoritmos se puede explicar por las siguientes razones:

- Al cambiar el primer objetivo (con la versión híbrida ponderamos la precisión con el número de características) estamos forzando a conseguir soluciones con menos características, aspecto que no teníamos en cuenta cuando considerábamos los otros objetivos. También deducimos que la modificación de la *distancia de crowding* y *función de densidad* para premiar a soluciones con menos características no tiene mucha influencia.

- Además, este objetivo ponderado soluciona un problema que podría ser una de las causas de que se consiguieran individuos en menos evaluaciones en la versión estándar y se explorara peor todo el campo de soluciones. El EMO estándar buscaba siempre soluciones que tenían alrededor de la mitad de las características (16 o 20 de las 34 que posee **IONOSPHERE** por ejemplo), esto hacía que el ratio de inconsistencias siempre obtuviera un valor de 0, por lo que el algoritmo multiobjetivo se convertía en un algoritmo monoobjetivo. Es decir, estábamos considerando dos objetivos, de los cuáles uno de ellos siempre valía cero por lo que no influía en la búsqueda. Fijando el primer objetivo a la ponderación ya mencionada conseguimos muchas menos características en la solución, y por tanto el ratio de inconsistencia va a ser mayor que 0, implicándose ahora sí en el proceso de búsqueda de la solución.

- La utilización de un enfoque híbrido del algoritmo CHC y la no permisividad de soluciones repetidas entre las no-dominadas para la versión híbrida hace que se tenga más diversidad en el algoritmo y que se explore mejor, obteniendo conjuntos de características con mejores precisiones en las soluciones del **NSGA-II híbrido**. Esto se nota sobre todo en problemas más grandes como el **SPLICE-IE** de 60 características.

Una vez vistos los resultados para el **NSGA-II** vamos a pasar a analizar los del **SPEA2** y **ϵ-MOEA**. Los resultados de las versiones **híbrida** y **estándar** del **SPEA2** (tablas 5.10 a 5.17) no difieren mucho de las conclusiones sacadas del **NSGA-II**. Sí diremos que el **SPEA2 híbrido** realiza más reinicialización que el **NSGA-II híbrido** que practicamente no tenía que reiniciar la población.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	11,11 %	17,36 %	0,04	4,75	0,96	11,11 %	0	10	0
PART. 2	17,14 %	5,71 %	0,04	5	0	11,43 %	0,03	12,5	0,58
PART. 3	13,89 %	14,44 %	0,04	5,2	0,84	11,11 %	0	9,5	0,71
PART. 4	11,76 %	9,80 %	0,02	4,67	1,15	8,82 %	0	13	0
PART. 5	2,94 %	8,82 %	0	5	0	1,76 %	0,02	11,6	0,97
PART. 6	22,86 %	12,38 %	0,02	5	1	11,43 %	0	10,5	0,71
PART. 7	17,14 %	14,29 %	0,05	4,33	0,58	17,14 %	0	9,5	0,71
PART. 8	13,89 %	13,19 %	0,03	4,75	0,5	11,11 %	0	10,67	0,58
PART. 9	11,76 %	10,29 %	0,02	5,25	0,5	8,82 %	0	10,5	0,71
PART. 10	11,11 %	9,44 %	0,03	4,4	0,55	11,81 %	0,01	13,75	0,96
MEDIA	13,36 %	11,57 %	0,03	4,84	0,61	10,46 %	0,01	11,15	0,59

Tabla 5.10: Soluciones devueltas por el SPEA2 híbrido y estándar para IONOSPHERE.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	2,35 %	4,59 %	0,03	4,9	0,99	1,41 %	0,01	10,8	0,45
PART. 2	8,24 %	4,26 %	0,02	4,63	1,09	3,53 %	0	9	0
PART. 3	4,71 %	3,41 %	0,01	4,8	1,32	0,59 %	0,01	10,67	1,51
PART. 4	4,71 %	3,74 %	0,01	4,91	1,04	3,24 %	0,01	11,13	1,25
PART. 5	7,06 %	5,51 %	0,02	4,56	1,03	3,70 %	0,02	11	1,53
PART. 6	7,06 %	2,75 %	0,02	4,44	1,13	1,18 %	0	12,33	0,58
PART. 7	7,14 %	4,76 %	0,03	4,57	1,34	3,10 %	0,04	11,2	1,3
PART. 8	3,57 %	4,40 %	0,02	4,6	0,97	1,67 %	0,01	10,8	0,84
PART. 9	7,14 %	4,30 %	0,02	4,38	1,12	1,64 %	0,01	11,63	0,92
PART. 10	9,52 %	8,16 %	0,01	4,71	1,11	3,57 %	0	11,5	0,71
MEDIA	6,15 %	4,59 %	0,02	4,65	1,11	2,36 %	0,01	11,01	0,91

Tabla 5.11: Soluciones devueltas por el SPEA2 híbrido y estándar para VEHICLE.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	1,75 %	5,01 %	0,02	4,29	0,95	5,26 %	0	17,5	0,71
PART. 2	1,75 %	3,51 %	0,02	3,89	0,93	5,26 %	0	13,75	0,96
PART. 3	0,00 %	1,00 %	0,01	4,29	1,11	1,75 %	0	17	0
PART. 4	3,51 %	3,33 %	0,02	4,2	0,92	2,63 %	0,01	18	2,83
PART. 5	5,26 %	6,77 %	0,02	4,14	0,69	3,51 %	0	16	1,1
PART. 6	5,26 %	3,12 %	0,02	4,11	0,78	5,26 %	0	16,25	1,28
PART. 7	3,51 %	5,26 %	0,02	4,63	1,3	4,39 %	0,01	14,5	2,12
PART. 8	7,02 %	5,26 %	0	4	0,82	5,56 %	0,01	14,83	1,33
PART. 9	10,53 %	9,65 %	0,03	4,17	0,75	8,77 %	0	13	0
PART. 10	10,71 %	6,38 %	0,03	4,29	1,11	8,93 %	0	16	0
MEDIA	4,93 %	4,93 %	0,02	4,2	0,94	5,13 %	0	15,68	1,03

Tabla 5.12: Soluciones devueltas por el SPEA2 híbrido y estándar para WISCONSIN.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	17,61 %	9,95 %	0,02	8,24	1,83	17,14 %	0	21	2,83
PART. 2	20,44 %	9,66 %	0,02	8,44	1,20	16,35 %	0	30	0
PART. 3	19,81 %	9,30 %	0,02	8,24	1,55	16,35 %	0	21	0
PART. 4	18,24 %	10,78 %	0,01	8,17	1,42	17,61 %	0	21	0
PART. 5	16,04 %	9,38 %	0,01	8,33	1,67	13,52 %	0	22	0
PART. 6	22,08 %	8,62 %	0,02	7,81	1,89	18,93 %	0	30	0
PART. 7	21,77 %	10,67 %	0,02	8,44	1,20	20,82 %	0	25	0
PART. 8	16,40 %	8,54 %	0,02	8,12	1,17	17,03 %	0	27	0
PART. 9	17,67 %	9,26 %	0,02	8,30	1,38	14,20 %	0	25	0
PART. 10	19,56 %	9,06 %	0,01	8,17	1,89	18,93 %	0	25	0
MEDIA	18,96 %	9,52 %	0,02	8,23	1,52	17,09 %	0	24,7	0,28

Tabla 5.13: Soluciones devueltas por el SPEA2 híbrido y estándar para SPLICE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	11,11 %	13,89 %	4	4	19,44 %	11,11 %	10	10	11,11 %
PARTICIÓN 2	17,14 %	2,86 %	5	5	8,57 %	8,57 %	12	12	8,57 %
PARTICIÓN 3	13,89 %	11,11 %	4	4	11,11 %	11,11 %	10	9	11,11 %
PARTICIÓN 4	11,76 %	8,82 %	4	4	11,76 %	8,82 %	13	13	8,82 %
PARTICIÓN 5	2,94 %	8,82 %	5	5	8,82 %	0,00 %	10	10	0,00 %
PARTICIÓN 6	22,86 %	11,43 %	5	4	14,29 %	11,43 %	11	10	11,43 %
PARTICIÓN 7	17,14 %	11,43 %	5	4	11,43 %	17,14 %	9	9	17,14 %
PARTICIÓN 8	13,89 %	8,33 %	5	4	16,67 %	11,11 %	10	10	11,11 %
PARTICIÓN 9	11,76 %	8,82 %	6	5	11,76 %	8,82 %	10	10	8,82 %
PARTICIÓN 10	11,11 %	5,56 %	4	4	5,56 %	11,11 %	13	13	11,11 %
MEDIA	13,36 %	9,11 %	4,7	4,3	11,94 %	9,92 %	10,8	10,6	9,92 %

Tabla 5.14: Mejores soluciones del SPEA2 híbrido y estándar para IONOSPHERE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	2,35 %	2,35 %	5	3	10,59 %	1,18 %	11	10	1,18 %
PARTICIÓN 2	8,24 %	1,18 %	5	3	4,71 %	3,53 %	9	9	3,53 %
PARTICIÓN 3	4,71 %	1,18 %	4	3	3,53 %	0,00 %	11	8	1,18 %
PARTICIÓN 4	4,71 %	1,18 %	5	3	3,53 %	2,35 %	10	9	3,53 %
PARTICIÓN 5	7,06 %	2,35 %	6	3	8,24 %	2,35 %	11	9	5,88 %
PARTICIÓN 6	7,06 %	1,18 %	5	3	5,88 %	1,18 %	12	12	1,18 %
PARTICIÓN 7	7,14 %	1,19 %	4	2	10,71 %	1,19 %	10	10	1,19 %
PARTICIÓN 8	3,57 %	2,38 %	4	3	8,33 %	1,19 %	10	10	1,19 %
PARTICIÓN 9	7,14 %	1,19 %	4	3	4,76 %	1,19 %	12	10	3,57 %
PARTICIÓN 10	9,52 %	5,95 %	4	3	9,52 %	3,57 %	11	11	3,57 %
MEDIA	6,15 %	2,01 %	4,6	2,9	6,98 %	1,77 %	10,7	9,8	2,60 %

Tabla 5.15: Mejores soluciones del SPEA2 híbrido y estándar para VEHICLE.

	TODAS	VERSIÓN HÍBRIDA			VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	1,75 %	3,51 %	3	3	3,51 %	5,26 %	18	17	5,26 %
PARTICIÓN 2	1,75 %	1,75 %	4	2	8,77 %	5,26 %	14	13	5,26 %
PARTICIÓN 3	0,00 %	0,00 %	4	3	1,75 %	1,75 %	17	17	1,75 %
PARTICIÓN 4	3,51 %	1,75 %	3	3	1,75 %	1,75 %	20	16	3,51 %
PARTICIÓN 5	5,26 %	3,51 %	4	3	5,26 %	3,51 %	14	14	3,51 %
PARTICIÓN 6	5,26 %	0,00 %	3	3	0,00 %	5,26 %	16	15	5,26 %
PARTICIÓN 7	3,51 %	1,75 %	4	3	5,26 %	3,51 %	13	13	3,51 %
PARTICIÓN 8	7,02 %	5,26 %	3	3	5,26 %	5,26 %	13	13	5,26 %
PARTICIÓN 9	10,53 %	7,02 %	5	3	10,53 %	8,77 %	13	13	8,77 %
PARTICIÓN 10	10,71 %	3,57 %	4	3	7,14 %	8,93 %	16	16	8,93 %
MEDIA	4,93 %	2,81 %	3,7	2,9	4,92 %	4,93 %	15,4	14,7	5,10 %

Tabla 5.16: Mejores soluciones del SPEA2 híbrido y estándar para WISCONSIN.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	17,61 %	6,92 %	5	3	9,12 %	16,98 %	23	19	17,30 %
PARTICIÓN 2	20,44 %	6,60 %	8	6	7,86 %	16,35 %	30	30	16,35 %
PARTICIÓN 3	19,81 %	5,97 %	7	4	7,55 %	16,35 %	21	21	16,35 %
PARTICIÓN 4	18,24 %	8,81 %	7	5	9,75 %	17,61 %	21	21	17,61 %
PARTICIÓN 5	16,04 %	7,55 %	5	5	7,55 %	13,52 %	22	22	13,52 %
PARTICIÓN 6	22,08 %	5,99 %	6	3	7,26 %	18,93 %	30	30	18,93 %
PARTICIÓN 7	21,77 %	5,99 %	6	6	5,99 %	20,82 %	25	25	20,82 %
PARTICIÓN 8	16,40 %	5,36 %	6	6	6,31 %	17,03 %	27	27	17,03 %
PARTICIÓN 9	17,67 %	6,31 %	8	5	6,62 %	14,20 %	25	25	14,20 %
PARTICIÓN 10	19,56 %	6,62 %	6	4	8,52 %	18,93 %	25	25	18,93 %
MEDIA	18,96 %	6,61 %	6,4	4,7	7,65 %	17,07 %	24,9	24,5	17,10 %

Tabla 5.17: Mejores soluciones del SPEA2 híbrido y estándar para SPLICE.

Por último compararemos las dos versiones **estándar** e **híbrida** del ϵ-**MOEA** (resultados en tablas 5.18 a 5.25). Ya adelantamos que el algoritmo estándar tenía una gran convergencia prematura por lo que los resultados obtenidos con este algoritmo no son muy buenos, siendo superados ampliamente por su rival híbrido. Al aplicar al ϵ-**MOEA** (que es un algoritmo genético estacionario) un enfoque híbrido con un *cruce intensivo* la reinicialización va a ser bastante constante en la ejecución del algoritmo ya que la población sólo cambia, como mucho, en dos individuos cada generación.

En las tablas 5.18, 5.19, 5.20 y 5.21 apreciamos cómo con el modelo estándar de ϵ**MOEA** la desviación típica casi siempre es 0. Esto es debido a que este algoritmo sólo suele devolver una solución del frente, por lo que no hay variación respecto a la media, justificándose así ese valor nulo: sólo suele conseguir frentes con una única solución. Es este algoritmo el peor de los que hemos aplicado al problema de selección de características en este *Proyecto*, ya que aunque para los otros dos EMO también se ha demostrado que las versiones híbridas presentan un mejor rendimiento que las estándar, en ninguno de los otros dos casos se obtiene tanta convergencia prematura ni frentes de Pareto tan poco diversos.

La mayor diferencia entre uno y otro enfoque se produce en el problema más grande de los cuatro, **SPLICE-IE** (ver tablas 5.21 y 5.25), por tanto nuevamente podemos afirmar que cuanto más grande sea el problema a abordar en cuanto a su dimensionalidad, mayor va a ser la diferencia positiva para la versión híbrida y más se va a justificar la utilización del EMO de selección de características.

Puede que la versión estándar del algoritmo tenga éxito aplicándose a otros problemas de optimización, pero respecto a la selección de características, sus resultados son inferiores a los obtenidos por otros algoritmos híbridos.

En conclusión podemos decir que el **enfoque híbrido supera la precisión global en la mayoría de los problemas al enfoque estándar**. **Selecciona muchas menos características** y es **muy superior en la base de datos más compleja**.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	11,11 %	16,67 %	0	5	0	16,67 %	0	15	0
PART. 2	17,14 %	10,48 %	0,02	4,33	1,15	5,71 %	0	14	0
PART. 3	13,89 %	5,56 %	0	4	0	5,56 %	0	18	0
PART. 4	11,76 %	5,88 %	0,04	5,5	0,71	8,82 %	0	15	0
PART. 5	2,94 %	5,88 %	0	4	0	5,88 %	0	19	0
PART. 6	22,86 %	12,86 %	0,06	4,5	0,71	17,14 %	0	17	0
PART. 7	17,14 %	13,33 %	0,06	5	1,73	17,14 %	0	12	0
PART. 8	13,89 %	13,89 %	0	5	0	19,44 %	0	13	0
PART. 9	11,76 %	0,00 %	0	5	0	5,88 %	0	13	0
PART. 10	11,11 %	16,67 %	0	5	0	13,89 %	0	17	0
MEDIA	13,36 %	10,12 %	0,02	4,73	0,43	11,61 %	0	15,3	0

Tabla 5.18: Soluciones devueltas por el ϵMOEA híbrido y estándar para IONOSPHERE.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	2,35 %	5,38 %	0,03	4,43	0,79	1,18 %	0	13	0
PART. 2	8,24 %	4,71 %	0,01	4,44	0,73	2,35 %	0	10	0
PART. 3	4,71 %	3,53 %	0	4,6	0,7	2,35 %	0	14	0
PART. 4	4,71 %	4,44 %	0,01	4,44	0,73	2,35 %	0	12	0
PART. 5	7,06 %	6,94 %	0,03	4,4	0,84	5,88 %	0	10	0
PART. 6	7,06 %	3,38 %	0,01	4,38	0,74	1,18 %	0	13	0
PART. 7	7,14 %	4,76 %	0,01	4,5	0,53	3,57 %	0	12	0
PART. 8	3,57 %	2,53 %	0,02	4,38	0,74	2,38 %	0	10	0
PART. 9	7,14 %	3,72 %	0,01	4,5	0,76	3,57 %	0	12	0
PART. 10	9,52 %	7,29 %	0,01	4,38	0,74	4,76 %	0	10	0
MEDIA	6,15 %	4,67 %	0,01	4,44	0,73	2,96 %	0	11,6	0

Tabla 5.19: Soluciones devueltas por el ϵMOEA híbrido y estándar para VEHICLE.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	1,75 %	3,51 %	0	4	0	3,51 %	0	15	0
PART. 2	1,75 %	6,14 %	0,02	4,5	0,84	1,75 %	0	14	0
PART. 3	0,00 %	1,32 %	0,02	4,75	0,5	1,75 %	0	18	0
PART. 4	3,51 %	3,51 %	0,02	4,2	0,84	1,75 %	0	19	0
PART. 5	5,26 %	7,02 %	0,04	4	0,58	8,77 %	0	15	0
PART. 6	5,26 %	3,22 %	0,01	4,83	0,41	8,77 %	0	15	0
PART. 7	3,51 %	5,61 %	0,02	4,4	0,89	3,51 %	0	16	0
PART. 8	7,02 %	4,21 %	0,01	4,2	0,84	5,26 %	0	13	0
PART. 9	10,53 %	8,77 %	0	5	0	12,28 %	0	17	0
PART. 10	10,71 %	5,80 %	0,01	4,5	0,58	8,93 %	0	16	0
MEDIA	4,93 %	4,91 %	0,01	4,44	0,55	5,63 %	0	15,8	0

Tabla 5.20: Soluciones devueltas por el ϵMOEA híbrido y estándar para WISCONSIN.

		VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
	TODAS	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}	$\overline{x}_{prec}$	σ_{prec}	$\overline{x}_{caracs}$	σ_{caracs}
PART. 1	17,61 %	11,95 %	0,01	8	1,73	20,13 %	0	23	0
PART. 2	20,44 %	11,08 %	0,01	9	1,41	14,47 %	0	22	0
PART. 3	19,81 %	14,31 %	0,02	8	1,41	18,24 %	0	24	0
PART. 4	18,24 %	11,64 %	0,00	10	0,00	19,18 %	0	22	0
PART. 5	16,04 %	11,95 %	0,00	10	0,00	16,35 %	0	23	0
PART. 6	22,08 %	11,36 %	0,02	8,5	0,71	16,40 %	0	23	0
PART. 7	21,77 %	12,93 %	0,00	11	0,00	17,35 %	0	22	0
PART. 8	16,40 %	13,56 %	0,02	9	1,41	16,09 %	0	22	0
PART. 9	17,67 %	12,30 %	0,00	9	0,00	18,93 %	0	22	0
PART. 10	19,56 %	13,41 %	0,02	9	0,00	16,40 %	0	22	0
MEDIA	18,96 %	12,45 %	0,01	9,15	0,67	17,35 %	0	22,5	0

Tabla 5.21: Soluciones devueltas por el ϵMOEA híbrido y estándar para SPLICE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	11,11 %	16,67 %	5	5	16,67 %	16,67 %	15	15	16,67 %
PARTICIÓN 2	17,14 %	8,57 %	5	3	11,43 %	5,71 %	14	14	5,71 %
PARTICIÓN 3	13,89 %	5,56 %	4	4	5,56 %	5,56 %	18	18	5,56 %
PARTICIÓN 4	11,76 %	2,94 %	6	5	8,82 %	8,82 %	15	15	8,82 %
PARTICIÓN 5	2,94 %	5,88 %	4	4	5,88 %	5,88 %	19	19	5,88 %
PARTICIÓN 6	22,86 %	8,57 %	5	4	17,14 %	17,14 %	17	17	17,14 %
PARTICIÓN 7	17,14 %	8,57 %	6	3	20,00 %	17,14 %	12	12	17,14 %
PARTICIÓN 8	13,89 %	13,89 %	5	5	13,89 %	19,44 %	13	13	19,44 %
PARTICIÓN 9	11,76 %	0,00 %	5	5	0,00 %	5,88 %	13	13	5,88 %
PARTICIÓN 10	11,11 %	16,67 %	5	5	16,67 %	13,89 %	17	17	13,89 %
MEDIA	13,36 %	8,73 %	5,0	4,3	11,61 %	11,61 %	15,3	15,3	11,61 %

Tabla 5.22: Mejores soluciones del ϵMOEA híbrido y estándar para IONOSPHERE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	2,35 %	2,35 %	5	3	10,59 %	1,18 %	13	13	1,18 %
PARTICIÓN 2	8,24 %	3,53 %	5	3	5,88 %	2,35 %	10	10	2,35 %
PARTICIÓN 3	4,71 %	3,53 %	5	3	3,53 %	2,35 %	14	14	2,35 %
PARTICIÓN 4	4,71 %	3,53 %	5	3	3,53 %	2,35 %	12	12	2,35 %
PARTICIÓN 5	7,06 %	3,53 %	5	3	8,24 %	5,88 %	10	10	5,88 %
PARTICIÓN 6	7,06 %	2,35 %	5	3	3,53 %	1,18 %	13	13	1,18 %
PARTICIÓN 7	7,14 %	3,57 %	4	4	5,95 %	3,57 %	12	12	3,57 %
PARTICIÓN 8	3,57 %	1,19 %	5	3	8,33 %	2,38 %	10	10	2,38 %
PARTICIÓN 9	7,14 %	2,38 %	5	3	5,95 %	3,57 %	12	12	3,57 %
PARTICIÓN 10	9,52 %	5,95 %	4	3	9,52 %	4,76 %	10	10	4,76 %
MEDIA	6,15 %	3,19 %	4,8	3,1	6,51 %	2,96 %	11,6	11,6	2,96 %

Tabla 5.23: Mejores soluciones del ϵMOEA híbrido y estándar para VEHICLE.

	TODAS	VERSIÓN HÍBRIDA				VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	1,75 %	3,51 %	4	4	3,51 %	3,51 %	15	15	3,51 %
PARTICIÓN 2	1,75 %	1,75 %	5	3	7,02 %	1,75 %	14	14	1,75 %
PARTICIÓN 3	0,00 %	0,00 %	5	4	1,75 %	1,75 %	18	18	1,75 %
PARTICIÓN 4	3,51 %	1,75 %	4	3	1,75 %	1,75 %	19	19	1,75 %
PARTICIÓN 5	5,26 %	3,51 %	4	3	5,26 %	8,77 %	15	15	8,77 %
PARTICIÓN 6	5,26 %	1,75 %	4	4	1,75 %	8,77 %	15	15	8,77 %
PARTICIÓN 7	3,51 %	3,51 %	5	3	5,26 %	3,51 %	16	16	3,51 %
PARTICIÓN 8	7,02 %	3,51 %	5	3	5,26 %	5,26 %	13	13	5,26 %
PARTICIÓN 9	10,53 %	8,77 %	5	5	8,77 %	12,28 %	17	17	12,28 %
PARTICIÓN 10	10,71 %	5,36 %	5	4	7,14 %	8,93 %	16	16	8,93 %
MEDIA	4,93 %	3,34 %	4,6	3,6	4,75 %	5,63 %	15,8	15,8	5,63 %

Tabla 5.24: Mejores soluciones del ϵMOEA híbrido y estándar para WISCONSIN.

	TODAS	VERSIÓN HÍBRIDA			VERSIÓN ESTÁNDAR			
		MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR	MEJOR PREC.	CARACS. MEJOR	MEJOR CARACS.	PREC. MEJOR
PARTICIÓN 1	17,61 %	11,32 %	9	6	11,95 %	20,13 %	23	23	20,13 %
PARTICIÓN 2	20,44 %	9,43 %	9	7	11,01 %	14,47 %	22	22	14,47 %
PARTICIÓN 3	19,81 %	12,58 %	7	7	12,58 %	18,24 %	24	24	18,24 %
PARTICIÓN 4	18,24 %	11,64 %	10	10	11,64 %	19,18 %	22	22	19,18 %
PARTICIÓN 5	16,04 %	11,95 %	10	10	11,95 %	16,35 %	23	23	16,35 %
PARTICIÓN 6	22,08 %	10,09 %	8	8	10,09 %	16,40 %	23	23	16,40 %
PARTICIÓN 7	21,77 %	12,93 %	11	11	12,93 %	17,35 %	22	22	17,35 %
PARTICIÓN 8	16,40 %	11,99 %	10	8	15,14 %	16,09 %	22	22	16,09 %
PARTICIÓN 9	17,67 %	12,30 %	9	9	12,30 %	18,93 %	22	22	18,93 %
PARTICIÓN 10	19,56 %	12,30 %	9	9	12,30 %	16,40 %	22	22	16,40 %
MEDIA	18,96 %	11,65 %	9,2	8,5	12,19 %	17,35 %	22,5	22,5	17,35 %

Tabla 5.25: Mejores soluciones del ϵMOEA híbrido y estándar para SPLICE.

5.3.2. EMOs híbridos para cada problema

Las figuras 5.1, 5.2, 5.3 y 5.4 muestran claramente los resultados obtenidos por los tres algoritmos con **enfoque híbrido**, que son los que mejores prestaciones han ofrecido, y los que hemos implementado en el *Proyecto* para los diferentes problemas que tratamos.

En estas gráficas se muestra la siguiente información: en la primera fila de barras se presenta el error medio de precisión de todas las particiones (todos los datos de error que se están mostrando en este capítulo son sobre el error en datos de test de cada partición, nunca sobre los mismos datos de entrenamiento) clasificando con **1NN**. Se representa tanto el error medio de la mejor solución por partición como la media de todas las soluciones devueltas por cada partición (los datos de los gráficos han sido obtenidos de las tablas 5.2 a 5.25).

La segunda hilera de columnas muestra el porcentaje de reducción de dimensionalidad respecto al problema original que se ha obtenido para cada uno de los algoritmos explicados.

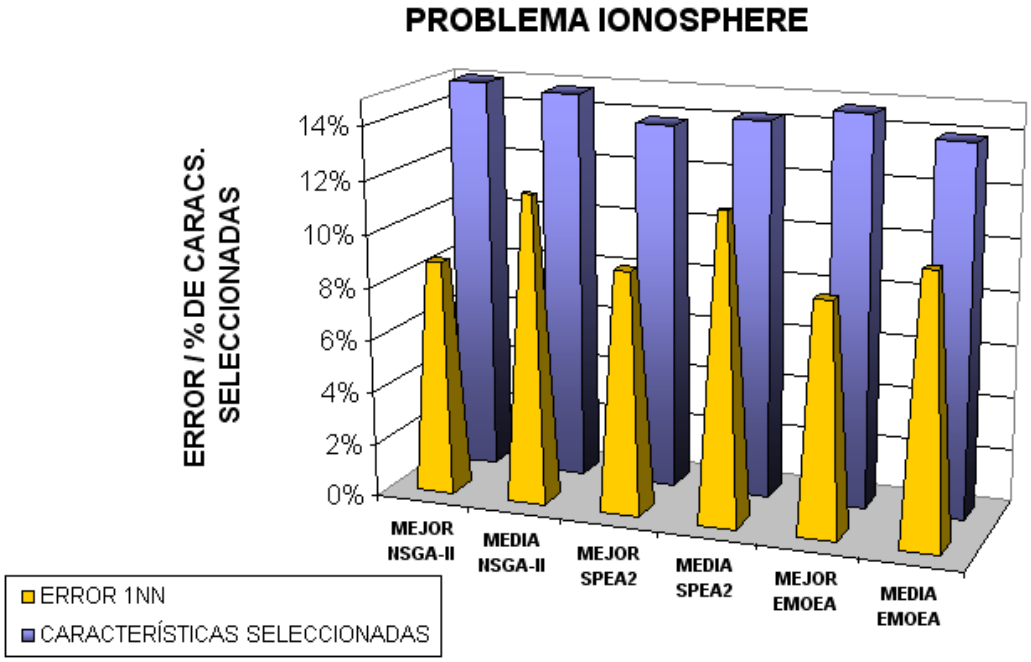

Figura 5.1: Precisión y reducción alcanzada por los EMO híbridos para IONOSPHERE.

De aquí obtendremos, para cada problema, cuál es el algoritmo que da mejores resultados, quedándonos finalmente con el mejor global para los cuatro problemas. Este será el que utilicemos para generar el multiclasificador evolutivo del que obtendremos los resultados que recogemos en la sección 5.4 de este capítulo. Cabe destacar que el sistema que hemos desarrollado permite que el multiclasificador y el AG que aprende la configuración del mismo trabajen con las soluciones de uno u otro EMO, no necesariamente con el mejor de todos.

Para el problema **IONOSPHERE** (figura 5.1) se aprecia como el algoritmo ϵ-MOEA es

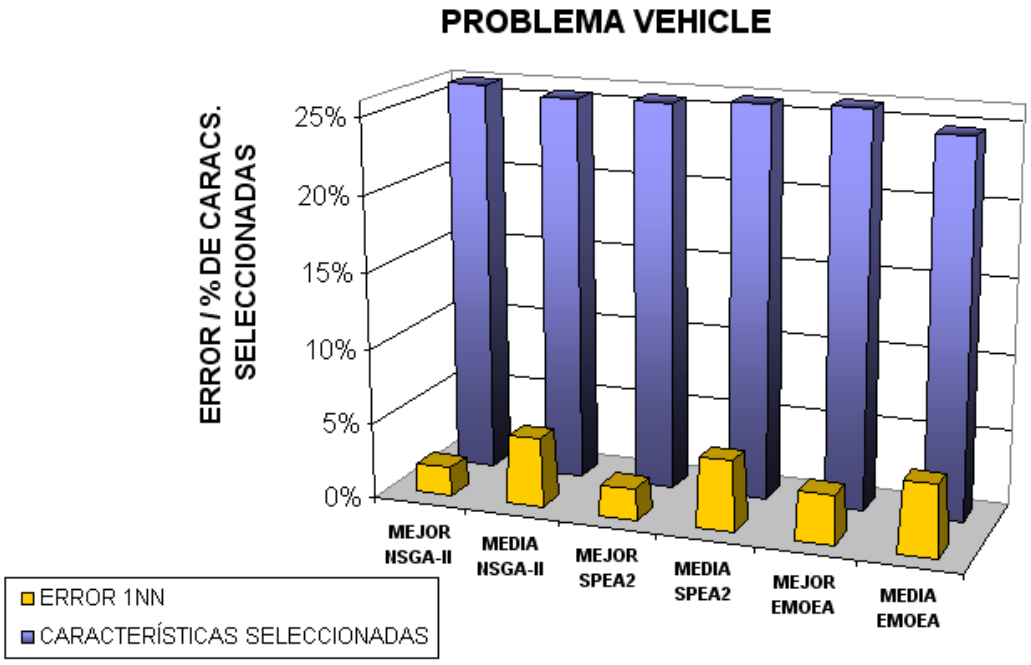

Figura 5.2: Precisión y reducción alcanzada por los EMO híbridos para VEHICLE.

el que menos error consigue, y que SPEA2 y NSGA-II están bastante a la par en cuanto a precisión. Respecto a las características seleccionadas, el algoritmo de *K. Deb*, NSGA-II, es el que más características selecciona, situándose por delante de él SPEA2 y ϵ-MOEA. Luego podemos concluir con que para la base de datos **IONOSPHERE**, el algoritmo **ϵ-MOEA híbrido** es el más idóneo.

El segundo problema que estamos tratando es **VEHICLE** (ver figura 5.2). En esta base de datos es el algoritmo NSGA-II el que consigue mejores resultados de precisión. Los tres algoritmos seleccionan más o menos las mismas características, aunque sorprende que sea la media de soluciones del ϵ-MOEA el que consiga reducir en un mayor número el conjunto de variables.

La base de datos de pacientes de cáncer **WISCONSIN** (ver figura 5.3), que como ya sabemos tiene 30 características, se ha podido reducir en cuanto a características a menos del 14 %. Respecto a la comparación de métodos vemos que en este caso el **ϵ-MOEA híbrido** es el que peores resultados ofrece tanto en características como en precisión. Los otros dos métodos tienen resultados más parecidos, obteniendo el **SPEA2 híbrido** menos características pero con peor precisión que el **NSGA-II híbrido**. Esto último es algo que se va a repetir para la mayoría de problemas.

El problema **SPLICE-IE** (figura 5.4) es el más importante que estamos observando y es el que nos proporciona los datos más interesantes. Vemos cómo se obtiene una media del 12 % de variables seleccionadas, esto significa que con sólo 8 variables (de 60 que poseía el

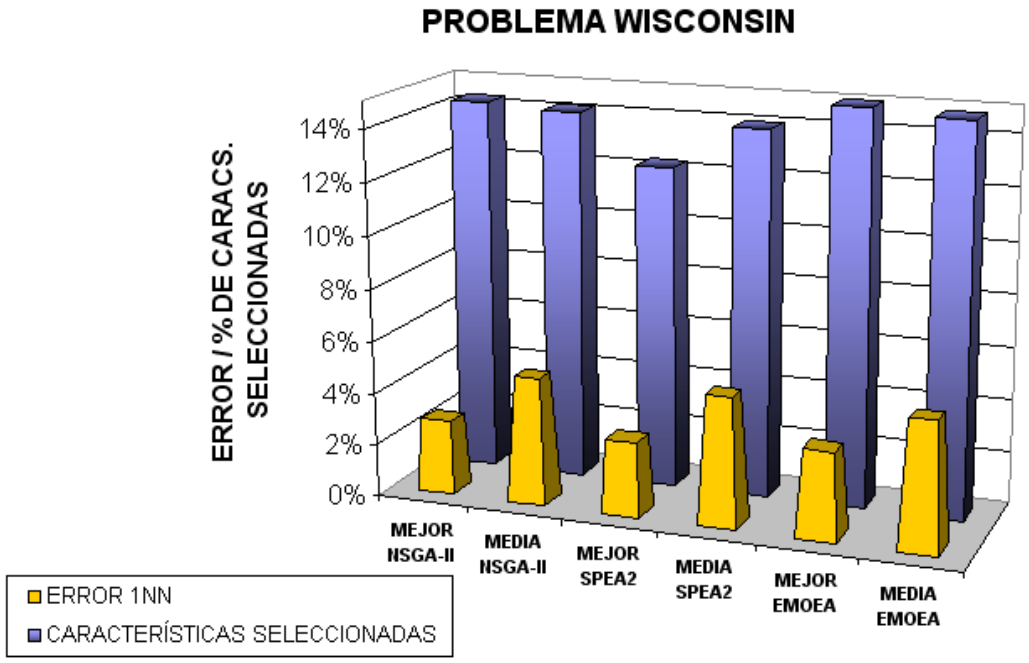

Figura 5.3: Precisión y reducción alcanzada por los EMO híbridos para WISCONSIN.

archivo de datos) estamos consiguiendo rebajar el error de un 18 % a un 8 %. Queda patente la enorme importancia de la selección de características, que no sólo se conforma con eliminar el 85 % de los datos, sino que además consigue reducir en más de la mitad el error clasificando con el **1NN**.

Para problemas grandes y con muchas características los EMO híbridos se comportan mejor que para problemas medianos. Son con estos problemas de alta dimensionalidad con los que se justifican enormemente estos EMO.

En la comparación entre los 3 algoritmos vemos que el **ϵ-MOEA híbrido** se distancia más de **NSGA-II híbrido** y **SPEA2 híbrido** con problemas grandes, mientras que los otros dos tienen resultados casi calcados.

En resumen, y tomando los cuatro problemas como guía, podemos concluir con los siguientes análisis:

- **El SPEA2 y NSGA-II son muy parecidos cuando se aplican al problema de la selección de características**.

- El **ϵ-MOEA** consigue resultados más desiguales según el problema y funciona peor con problemas grandes. Mientras que casi siempre, el NSGA-II obtiene un poco más de precisión que SPEA2 en detrimento de seleccionar un poco más de características.

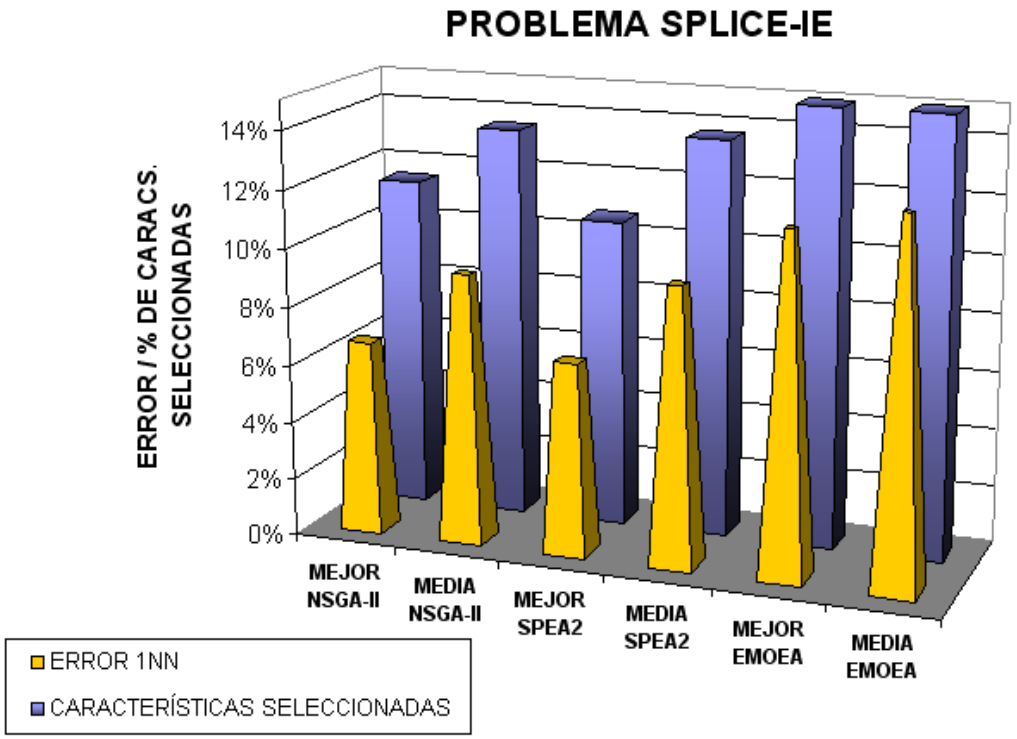

Figura 5.4: Precisión y reducción alcanzada por los EMO híbridos para SPLICE.

Seleccionamos al NSGA-II híbrido como el algoritmo que mejores resultados obtiene aunque seguido de cerca por el algoritmo SPEA2. Esta decisión la justificamos porque es el que mejor precisión obtiene como media en los cuatro problemas.

5.3.3. Comparación de los EMO híbridos frente a otras técnicas de selección de características

Aparte de las comparaciones entre los distintos algoritmos implementados en el *Proyecto* es necesario comparar con otras técnicas de selección de características para ver si consiguen mejorar lo ya existente. Para tal fin hemos utilizado la herramienta de Minería de Datos **KEEL** que incluye muy diversos algoritmos de selección de características. En concreto hemos elegido los que enumeramos a continuación (ver tablas de parámetros):

- Algoritmo *Forward* (hacia adelante) y *Backward* (hacia atrás).
- LVF.
- *Grasp* de *Battiti*.
- Dos AAGG de tipo envolvente y un AG de tipo filtro.

ALGORITMO FORWARD y BACKWARD	
Tipo de algoritmo de SC	Wrapper
Valor k del kNN	1

ALGORITMO LAS VEGAS FILTER	
Tipo de algoritmo de SC	Filter
Máximo número de ejecuciones	770
Ratio de inconsistencias permitido	0

ALGORITMO GRASP DE BATTITI	
Tipo de algoritmo de SC	Filter
Número de características a seleccionar	25 % del total de caracs.
Valor k de ponderación del algoritmo	1
Valor *beta*	0.5

AG Generacional Wrapper	
Representación	Binaria
Tamaño de la población	100
Número máximo de evaluaciones	5,000
Ponderación *alfa* para el fitness	1 y 0.7
Probabilidad de cruce	0.6
Probabilidad de mutación	0.01 respecto al num. de genes

AG Generacional Filter	
Representación	Binaria
Tamaño de la población	100
Número máximo de evaluaciones	5,000
Ponderación *alfa* para el fitness	0.9
Probabilidad de cruce	0.6
Probabilidad de mutación	0.01 respecto al num. de genes
Selección de individuos	k-torneo con k=5

Todos estos algoritmos utilizan una validación cruzada con *test* clasificando con el algoritmo **1NN**. Diremos también que los algoritmos *Forward* y *Backward* son los únicos algoritmos determinísticos de los comparados por lo que para los demás se han tenido que establecer semillas para números aleatorios.

Las gráficas 5.5, 5.6, 5.7 y 5.8 representan, para cada problema tratado, los errores en *test* y el número de características seleccionadas por cada algoritmo. En las gráficas están presentes todos los algoritmos nombrados anteriormente junto con la media de todas las soluciones devueltas por el NSGA-II y la mejor de todas las soluciones devueltas.

Comenzaremos comentando los resultados para el problema **IONOSPHERE**. **La mejor solución del NSGA-II, al igual que ocurrirá con todos los demás problemas, es la que mejor precisión obtiene sin ningún género de dudas**. El *AG Wrapper #2* con un 9,93 % de error y el *Grasp* de *Battiti* con un 10,25 % son los que le siguen en cuanto a precisión. Sin embargo, seleccionan más del doble de características. Es por ello por lo que el algoritmo *Forward* y la media de soluciones del NSGA-II, aunque se equivocan más en la predicción seleccionan menos características, presentando resultados generales más convenientes.

Con el problema **VEHICLE** (gráfica 5.6) la mejor solución del NSGA-II obtiene un error bastante bajo del 1.89 %. Detrás del NSGA-II se encuentran como mejores técnicas el *Forward* y los *AAGG Wrapper #1* y *#2*.

El tercer problema que estamos analizando es **WISCONSIN** (gráfica 5.7). Nuevamente el NSGA-II, tanto con su mejor solución como con su media de soluciones, es el que ofrece los mejores resultados.

Por último vemos en la gráfica 5.8 del problema **SPLICE** cómo existen dos grupos de algorimos según sus resultados. El primer grupo consigue unos resultados buenos más o menos parecidos, mientras que el otro grupo demuestra que sus resultados están por debajo de los demás. En el primer grupo está el NSGA-II tanto en media de soluciones como en mejor solución (que vuelve a ser el mejor de todos), el *Forward* y el *AG Wrapper*. Estos dos algoritmos son los que, sin contar al NSGA-II, mejores resultados obtienen como media en todos los problemas que hemos visto.

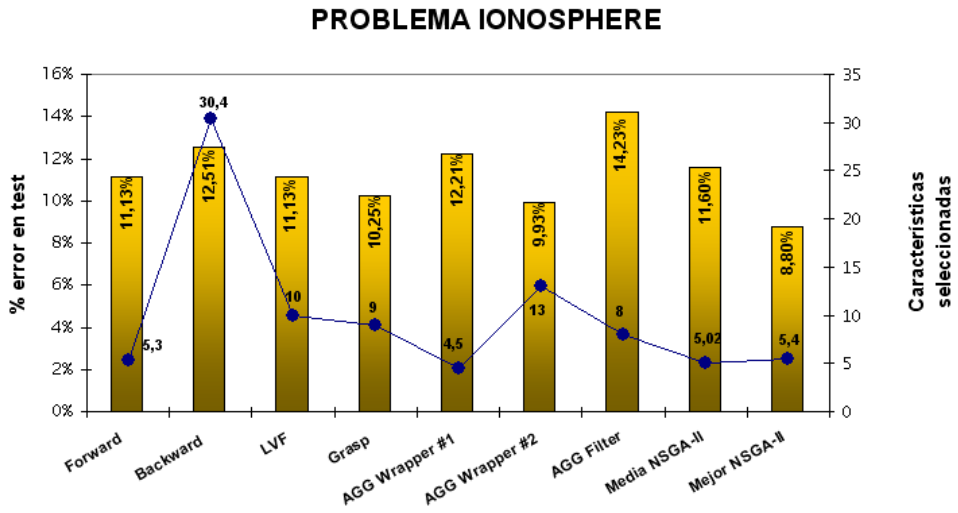

Figura 5.5: Resultados de distintas técnicas de SC para el problema IONOSPHERE.

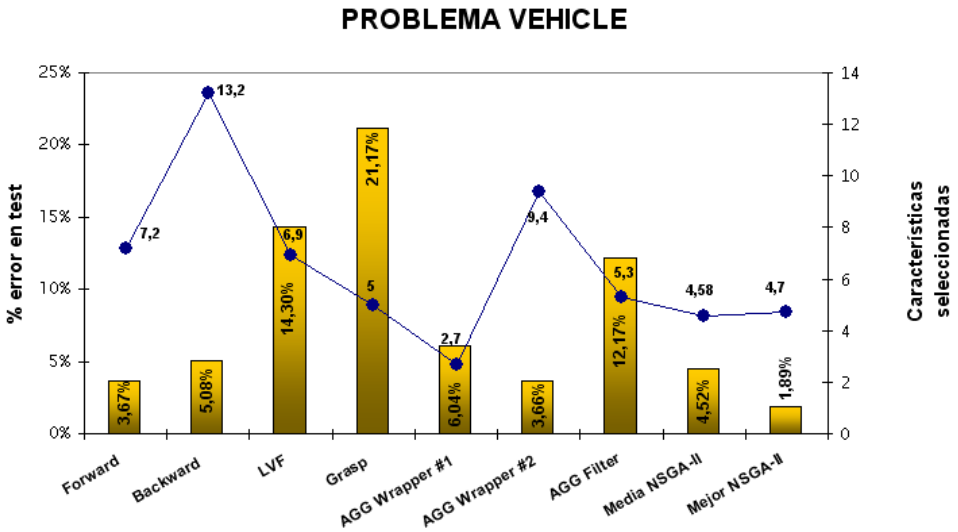

Figura 5.6: Resultados de distintas técnicas de SC para el problema VEHICLE.

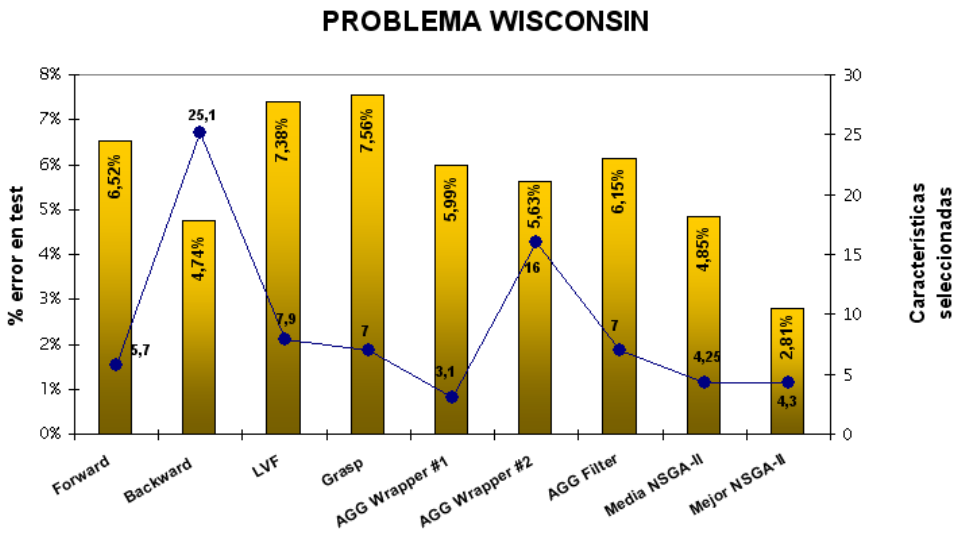

Figura 5.7: Resultados de distintas técnicas de SC para el problema WISCONSIN.

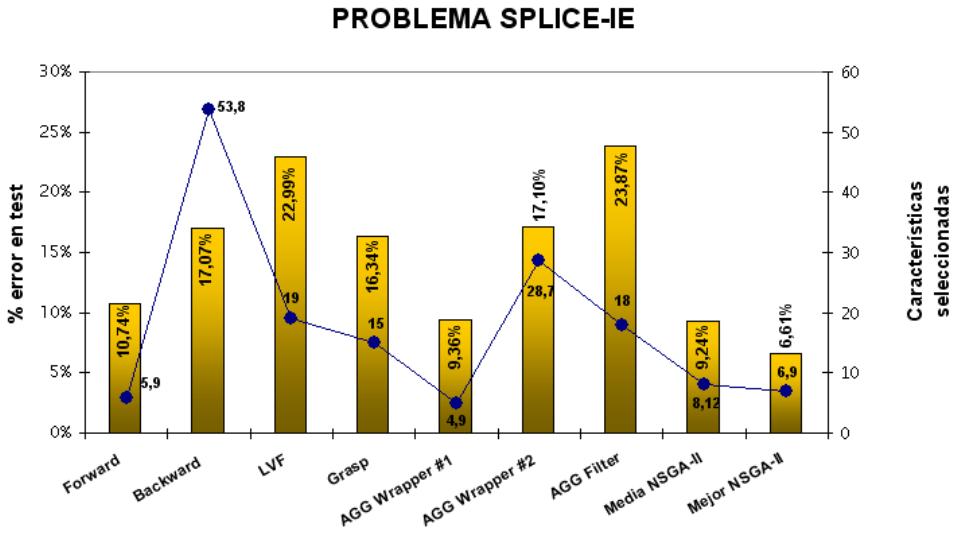

Figura 5.8: Resultados de distintas técnicas de SC para el problema SPLICE.

5.4. Resultados y análisis del multiclasificador evolutivo

Hemos realizado experimentaciones con el multiclasificador que aprende tanto los pesos de cada clasificador particular como los valores del algoritmo **kNN** para los problemas **IONOSPHERE**, **VEHICLE**, **WISCONSIN** y **SPLICE-IE** sobre las soluciones devueltas por el algoritmo **NSGA-II híbrido**.

Los valores de los parámetros necesarios por el algoritmo evolutivo del multiclasificador son los que se muestran a continuación:

- Tamaño de la población: 100
- Número máximo de evaluaciones: 5,000
- Probabilidad de mutación: 0.01 a nivel de gen
- Ratio de divergencia: 0.35

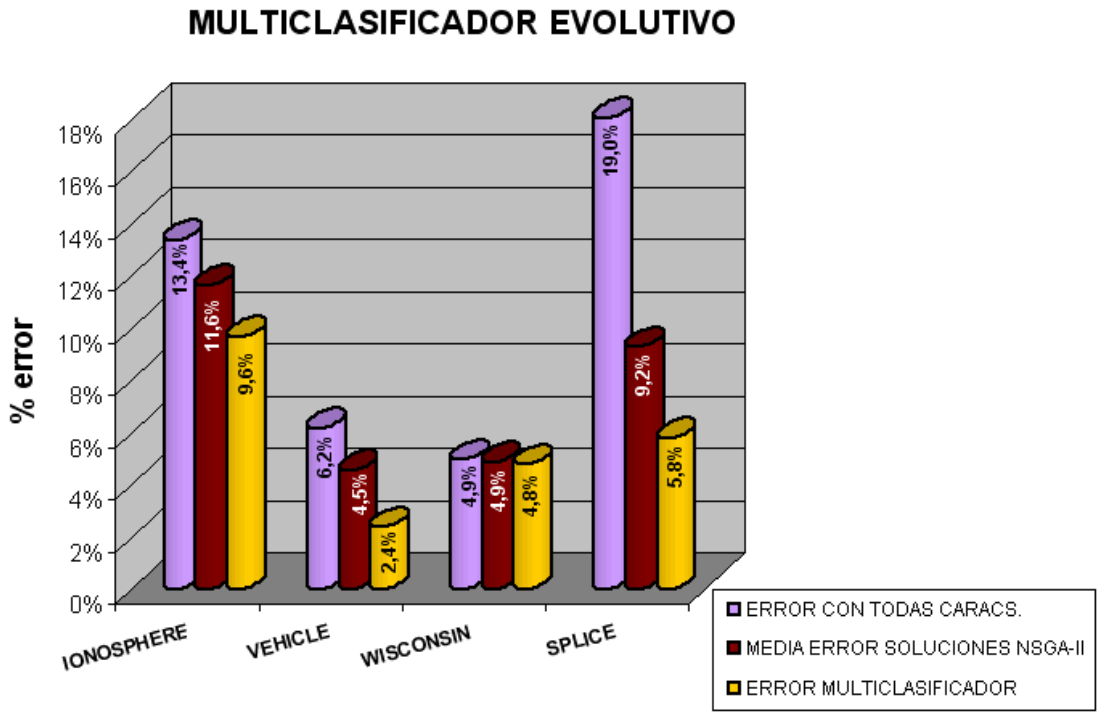

Figura 5.9: Resultados del multiclasificador para los cuatro problemas.

Los resultados obtenidos se muestran para los cuatro problemas en las tablas 5.26, 5.27, 5.28 y 5.29, comparándolos tanto con el error de clasificación en *test* considerando todas las

variables, como con la media del error de todas las soluciones devueltas por el algoritmo de selección de características multiobjetivo **NSGA-II híbrido** ($\overline{x}_{prec}$). Mostramos además el número de clasificadores iniciales (número de conjuntos de características devueltos por el EMO), el número de clasificadores finales y el error con los datos de *training* del multiclasificador.

Se aprecia como en todos los problemas el multiclasificador mejora los dos valores. Además, cuando tenemos un buen número de clasificadores, el multiclasificador elimina una buena parte de ellos gracias a la función umbral que establecimos (ver sección 3 capítulo 3), ahorrándonos su almacenamiento y utilización.

La gráfica 5.9 representa los valores de error de la clasificación con todas las características, la media de error de las soluciones devueltas por el **NSGA-II híbrido** y el error del multiclasificador.

Aunque en todos los problemas se consigue mejorar el error medio del NSGA-II, es con el problema **SPLICE** con el que mayor diferencia positiva se produce. Además, la precisión obtenida con el multiclasificador consigue incluso mejorar la de la mejor solución devuelta por el EMO.

Concluimos con que **la utilización del multiclasificador y del AG que aprende su configuración**, aparte de ser **necesario para ayudarnos a fusionar todos los conjuntos de características** devueltos por el EMO, nos es útil porque **mejora también el error medio** que obteníamos clasificando con los conjuntos de características, incluso en BBDD como **SPLICE** logramos obtener errores inferiores a los que obteníamos devolviendo la mejor solución del EMO.

	ERROR TODAS	$\overline{x}_{prec}$	ERROR TEST	CLASIF. INICIALES	CLASIF. FINALES	ERROR TRA
PART. 1	11,11 %	15,97 %	13,89 %	4	4	2,86 %
PART. 2	17,14 %	6,43 %	5,71 %	4	4	2,85 %
PART. 3	13,89 %	14,44 %	11,11 %	5	5	2,86 %
PART. 4	11,76 %	9,80 %	11,76 %	3	3	3,15 %
PART. 5	2,94 %	8,82 %	0,00 %	2	2	4,42 %
PART. 6	22,86 %	12,38 %	8,57 %	3	3	3,80 %
PART. 7	17,14 %	14,29 %	11,43 %	3	3	4,43 %
PART. 8	13,89 %	13,89 %	11,11 %	3	3	4,13 %
PART. 9	11,76 %	8,82 %	8,82 %	4	4	2,52 %
PART. 10	11,11 %	11,11 %	13,89 %	3	3	4,44 %
MEDIA	13,36 %	11,60 %	9,63 %	3,40	3,40	3,55 %

Tabla 5.26: Tabla de resultados del multiclasificador evolutivo para IONOSPHERE.

	ERROR TODAS	$\overline{x}_{prec}$	ERROR TEST	CLASIF. INICIALES	CLASIF. FINALES	ERROR TRA
PART. 1	2,35 %	4,28 %	0,00 %	11	5	1,71 %
PART. 2	8,24 %	4,26 %	2,35 %	16	7	0,66 %
PART. 3	4,71 %	3,10 %	1,18 %	11	4	1,18 %
PART. 4	4,71 %	3,82 %	2,35 %	12	7	0,79 %
PART. 5	7,06 %	5,51 %	3,53 %	16	4	0,66 %
PART. 6	7,06 %	3,06 %	2,35 %	10	5	1,45 %
PART. 7	7,14 %	4,29 %	1,19 %	15	7	0,79 %
PART. 8	3,57 %	4,40 %	3,57 %	10	4	1,31 %
PART. 9	7,14 %	4,30 %	2,38 %	13	7	1,05 %
PART. 10	9,52 %	8,18 %	4,76 %	8	6	1,18 %
MEDIA	6,15 %	4,52 %	2,36 %	12,2	5,6	1,07 %

Tabla 5.27: Tabla de resultados del multiclasificador evolutivo para VEHICLE.

	ERROR TODAS	$\overline{x}_{prec}$	ERROR TEST	CLASIF. INICIALES	CLASIF. FINALES	ERROR TRA
PART. 1	1,75 %	4,09 %	3,51 %	6	4	1,37 %
PART. 2	1,75 %	3,70 %	3,51 %	9	6	1,37 %
PART. 3	0,00 %	1,54 %	1,75 %	8	2	1,76 %
PART. 4	3,51 %	3,51 %	5,26 %	9	5	1,56 %
PART. 5	5,26 %	7,31 %	5,26 %	6	4	1,37 %
PART. 6	5,26 %	3,24 %	3,51 %	13	7	1,37 %
PART. 7	3,51 %	5,26 %	5,26 %	8	7	0,98 %
PART. 8	7,02 %	5,13 %	3,51 %	13	7	1,56 %
PART. 9	10,53 %	9,65 %	8,77 %	6	4	1,17 %
PART. 10	10,71 %	5,06 %	7,14 %	6	4	1,36 %
MEDIA	4,93 %	4,85 %	4,75 %	8,40	5,00	1,39 %

Tabla 5.28: Tabla de resultados del multiclasificador evolutivo para WISCONSIN.

	ERROR TODAS	$\overline{x}_{prec}$	ERROR TEST	CLASIF. INICIALES	CLASIF. FINALES	ERROR TRA
PART. 1	17,61 %	9,39 %	4,76 %	22	9	1,45 %
PART. 2	20,44 %	9,49 %	4,72 %	21	11	3,40 %
PART. 3	19,81 %	8,87 %	4,70 %	21	10	3,49 %
PART. 4	18,24 %	10,75 %	5,90 %	23	12	2,98 %
PART. 5	16,04 %	9,20 %	8,18 %	16	11	2,11 %
PART. 6	22,08 %	8,14 %	5,88 %	20	11	1,30 %
PART. 7	21,77 %	10,33 %	6,31 %	15	10	3,22 %
PART. 8	16,40 %	8,25 %	5,26 %	23	13	4,33 %
PART. 9	17,67 %	8,87 %	5,36 %	18	9	3,04 %
PART. 10	19,56 %	9,17 %	6,92 %	18	8	2,56 %
MEDIA	18,96 %	9,24 %	5,80 %	19,7	10,4	2,79 %

Tabla 5.29: Tabla de resultados del multiclasificador evolutivo para SPLICE.

Conclusiones finales

Síntesis y presentación del trabajo realizado

En el *Proyecto* que acabamos de finalizar hemos estudiado y desarrollado diversas técnicas evolutivas aplicadas a Minería de Datos. Con él se han cubierto los objetivos planteados al inicio:

- Se ha conseguido diseñar un sistema de clasificación para problemas complejos de alta dimensionalidad, probándolo en distintas BBDD reales.
- Hemos aplicado algoritmos de **Optimización Evolutiva Multiobjetivo** (**EMO**) a la tarea de selección de características.
- Se ha desarrollado un sistema Web de Minería de Datos que toma como base la propuesta evolutiva estudiada.

En relación a la **aplicación de técnicas de Optimización Evolutiva Multiobjetivo al problema de la selección de características** se han tocado los siguientes aspectos:

- Generación de varios conjuntos de características a partir de la BD de partida, creando un enfoque original y novedoso hasta la fecha.
- Desarrollo inicial, tras un amplio estudio previo, de tres algoritmos **EMO** estándar diferentes: NSGA-II, SPEA2 y ϵMOEA.
- Modificación de los tres algoritmos estándar anteriores gracias a una hibridación. Esta hibridación ha conseguido adaptar los algoritmos de una forma más conveniente al problema de la selección de características. Para esta nueva versión de los algoritmos nos hemos basado en el CHC de *Eshelman* [9] y en modificaciones a nivel de objetivos.
- Tras implementar las dos versiones, **estándar** e **híbrida**, las hemos aplicado a cuatro problemas de clasificación diferentes: IONOSPHERE, VEHICLE, WISCONSIN y SPLICE. Las conclusiones más importantes a las que hemos llegado son:
  - **Mejor rendimiento del enfoque híbrido respecto al estándar** en los tres algoritmos multiobjetivo.

  - **Reducción media del 85 % de las características originales de la BBDD**.
  - **Aumento considerable de la precisión original que se obtenía clasificando con todas las características**. Así, para el problema IONOSPHERE se ha mejorado la precisión en un 1,70 %, para VEHICLE en otro 1,80 %, y para el problema más complejo utilizado, SPLICE, se ha pasado del 18.96 % de error al 9.24 %, a pesar de haber reducido los datos en más de un 86 %.
  - Se han obtenido **mejores resultados en los algoritmos NSGA-II y SPEA2 respecto a ϵMOEA**. Aunque se ha encontrado una gran similitud entre los dos primeros, **hemos seleccionado al NSGA-II híbrido como el mejor de todos los implementados**.

- También hemos comparado este mejor algoritmo, **NSGA-II híbrido**, diseñado e implementado totalmente por nosotros, con técnicas clásicas de selección de características (algoritmos voraces, LVF, Grasp de *Battiti*, y AAGG básicos tanto filtros como envolventes). La comparativa obtenida nos ha permitido llegar a conclusiones bastante positivas:

  - Sin ningún género de duda, **la mejor solución devuelta por el NSGA-II híbrido supera ampliamente a todas las técnicas clásicas**.
  - **La media de las soluciones devueltas por el NSGA-II**, aunque obviamente no llega a la bondad de la mejor solución, **consigue en todos los problemas resultados bastante buenos**, situándose siempre entre la primera y tercera mejor técnica de SC.
  - Observando únicamente los algoritmos clásicos de SC podemos afirmar que para las cuatro BBDD analizadas, los AAGG envolventes y el algoritmo voraz hacia adelante (forward) son los que mejores resultados ofrecen.

Con relación al **sistema de clasificación diseñado** podemos reseñar lo siguiente:

- Se ha diseñado un sistema de clasificación basado en los conjuntos de características devueltos por los EMO. Se ha seguido un modelo de multiclasificación mediante pesos de importancia.

- Hemos desarrollado también dos tipos de AAGG para el aprendizaje y configuración automática del multiclasificador.

- Utilizando los conjuntos de características devueltos por el algoritmo NSGA-II híbrido se ha aplicado el AG que aprende la totalidad de la configuración del multiclasificador, tomando resultados para los cuatro problemas anteriormente mencionados. Hemos obtenido los siguientes resultados:

  - En todos los problemas, **el multiclasificador mejora el error medio de los conjuntos de características devueltos por el NSGA-II híbrido**. También se mejora ampliamente la precisión de clasificación utilizando toda la BD original.

- No se consiguen igualar los resultados de las mejores soluciones devueltas por el NSGA-II híbrido.
- Gracias a la función umbral utilizada en el diseño del multiclasificador (ver sección 3.3, página 96 del *Proyecto*) **se reduce en una media del 50 % el número de clasificadores iniciales que necesitaba el multiclasificador**. Así nos ahorramos almacenar y tener que utilizar clasificadores cuyo resultado no va a influir en la decisión final.

Por último indicaremos las conclusiones obtenidas sobre el **sistema Web** final que hemos desarrollado basándonos en nuestros algoritmos evolutivos propuestos y en el multiclasificador. Dicho sistema ha cumplido las siguientes expectativas:

- Se ha utilizado un modelado **UML** tanto para la especificación de requisitos, análisis y diseño del sistema.
- También hemos hecho uso del patrón de diseño **Modelo-Vista-Controlador** y de la tecnología **JSP/Servlet** para el diseño de la interfaz.
- Se ha desarrollado la documentación completa de todo el sistema, documentando exhaustivamente todas las clases y métodos desarrollados. La documentación tiene formato **HTML**, es fácilmente navegable y sigue los estándares **JAVA**.
- El prototipo del sistema Web construido es:
  - Amigable y de fácil uso para el usuario.
  - Posee ayuda integrada en cada paso dentro de la misma página Web.
  - Todo archivo generado tiene formato de intercambio **XML**.

Mejoras potenciales futuras

Destacaremos también la posible continuidad y mejoras que se pueden aplicar al *Proyecto* desarrollado:

- Inclusión de nuevos modelos de clasificación al sistema. Recordemos que en esta versión sólo está disponible el clasificador **kNN**. Añadiendo otros modelos posibles tales como árboles de decisión, reglas difusas o nítidas etc... conseguiremos, además de darle al usuario una mayor gama de elección, mejorar las prestaciones del AG que aprende los parámetros que componen el multiclasificador.
- Mejora de la interfaz Web. La implementación realizada en el *Proyecto* es únicamente un prototipo que cumple los requisitos básicos ya expuestos en el capítulo 4.

  Puede ser una mejora futura el diseño de una interfaz con mayores prestaciones, tales como:

- Ayuda contextual más amplia e inteligente.
- Visualización gráfica de estadísticas y comparativas de resultados.
- Ilustraciones dinámicas que vayan mostrando los pasos dados por el multiclasificador a la hora de clasificar, justificando así su decisión ante el usuario.

Apéndice A

MVC sobre tecnología Servlet/JSP de Java

En este apéndice no queremos realizar manual básico sobre *servlets* ni JSP, sólo queremos reseñar la metodología y patrones que hemos empleado en el diseño de la aplicación Web para intentar separar totalmente el núcleo del sistema de la interfaz. Antes de hablar de estos patrones de diseño anotaremos algo sobre la tecnología JAVA para la Web.

A.1. Servlets

Un *servlet* HTTP puede ser referenciado a partir de la dirección de una página HTML. Del mismo modo, puede ser accedido explícitamente por su nombre dentro de un hiperenlace URL o como resultado de un envío de información a un formulario Web.

Los *servlets* puede ser encadenados para producir efectos complejos o filtrar información (*ver subsección A.1.1*). También el código puede ser embebido dentro de páginas Web utilizando la tecnología *Java Server Pages* o JSP, que describiremos más adelante.

Para comprender la tecnología *servlet* habrá que entender perfectamente el protocolo HTTP, qué significa *request* y *response*, GET y POST etc...

El API *servlets* utiliza las clases e interfaces de dos paquetes JAVA: **javax.servlet** y **javax.servlet.http**. El primero contiene clases para un soporte genérico e independiente del protocolo, mientras que el segundo ya añade la funcionalidad específica del protocolo HTTP. Cualquier *servlet* debe implementar la interfaz **javax.servlet.Servlet**. Si nuestro *servlet*, como suele ser habitual, va a estar destinado a utilizar el protocolo HTTP, implementaremos la interfaz **javax.servlet.http.HttpServlet**.

Al igual que un *applet*[1], un *servlet* no tiene un método *main()* como cualquier programa JAVA estándar. En vez de este método, para que un *servlet* entre en acción se tienen que

[1] Otra tecnología JAVA para la Web. El *applet* se tiene que descargar y ejecutar en el cliente.

utilizar otras vías. Centrándonos en el caso de un *servlet* HTTP, habrá que implementar las operaciones **doGet()**, para manejar peticiones GET, y **doPost()** para las peticiones POST. En función del tipo de peticiones que acepte se utilizará uno u otro método.

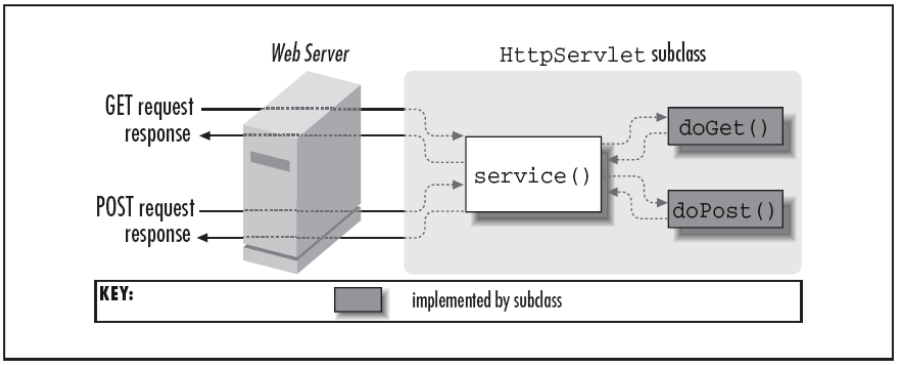

Figura A.1: Funcionamiento del *servlet* y sus métodos *doGet()* y *doPost()* ([44]).

Además, un *servlet* HTTP podrá sobrescribir las operaciones heredadas de la interfaz **HttpServlet doPut()** y **doDelete()**, que son los métodos que manejan las peticiones PUT y DELETE respectivamente. Aunque existen, los *servlets* HTTP generalmente no utilizan otros métodos similares como son **doHead()**, **doTrace()**, o **doOptions()**.

El ejemplo inicial típico de *servlet* es el *Hola Mundo* que genera una página HTML con ese texto más el nombre que el usuario envía. El *servlet* será accedido mediante un navegador Web, siendo su ejecución transparente al usuario. El código siguiente resuelve este sencillo programa:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {

      public void doGet(HttpServletRequest req, HttpServletResponse res)
                throws ServletException, IOException {

            res.setContentType("text/html");
            PrintWriter out = res.getWriter();
            out.println("<HTML>");
            out.println("<HEAD><TITLE>Hello World");
            out.println(req.getParameter("nombre"));
            out.println("</TITLE></HEAD>");
            out.println("<BODY>");
```

```
            out.println("<BIG>Hello World</BIG>");
            out.println("</BODY></HTML>");
        }
}
```

El *servlet* del ejemplo sobrescribe el método **doGet()**, heredado de la interfaz. Cada vez que el servidor Web reciba una petición GET para este *servlet*, el servidor invocará el método **doGet()**, pasando los objetos **HttpServletRequest** y **HttpServletResponse** como argumentos.

El objeto **HttpServletRequest** representa la petición del cliente, permitiendo al *servlet* acceder a la información sobre el cliente, parámetros enviados por él, y cabeceras HTTP de la petición. En el ejemplo, el cliente le ha enviado por medio de un formulario una variable llamada *nombre*. El *servlet*, mediante el método **getParameter(*nombreVariable*)**, obtiene ese valor para mostrarlo en la página.

El objeto **HttpServletResponse** representa la respuesta del *servlet* al cliente. Se utiliza para poner en él la información que será devuelta al cliente. Esta puede ser de cualquier tipo. En este ejemplo hemos especificado que será información del tipo *text/html*, que es el estándar MIME para páginas HTML. Para ello usamos el método **setContentType()**.

Después, el *servlet* usará el método **getWriter()** para obtener un objeto de la clase *PrintWriter*, que será el *stream* en el que iremos guardando la información a imprimir en la página que verá el cliente.

El ejemplo es bastante sencillo, pero la manera de actuar cuando programamos un *servlet* es siempre más o menos la misma. Obtener los datos enviados por el usuario mediante el objeto **HttpServletRequest**, procesar la información y mostrársela al usuario utilizando el objeto de respuesta **HttpServletResponse**.

A.1.1. Filtros

Un *servlet* aislado puede crear una página completa desde el servidor, sin embargo, los *servlets* pueden cooperar entre ellos en un proceso conocido como **cadena de *servlets*** (*servlet chaining*).

La petición realizada por el cliente es enviada por el servidor al primer *servlet* de la cadena, la respuesta del primer *servlet* pasa como petición al segundo *servlet*, y así hasta que el último de la cadena devuelve la respuesta al navegador.

Hay dos formas de crear una cadena de *servlets* a partir de la petición de un usuario. La primera consiste en que se le indique al servidor que cierta URL (dirección de una página Web) será manejada por una cadena de *servlets* específica. La otra forma es decirle al servidor que envíe todas las salidas de un determinado tipo a un *servlet* antes de que se devuelva al cliente, creando una cadena al vuelo.

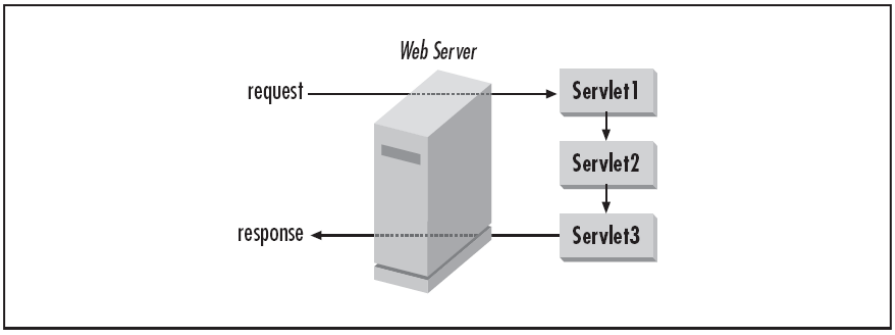

Figura A.2: Esquema de las cadenas de *servlets* o filtros ([44]).

Cuando un *servlet* convierte un tipo de información en otra se conoce con el nombre de **filtrado** o **filtro**. Esta técnica la utilizaremos para diseñar el sistema Web mediante el patrón **Modelo-Vista-Controlador** (ver sección A.3).

A.2. JSP

JSP (*Java Server Pages*) es un método alternativo a los *servlets* para definir una página Web dinámica en JAVA, aunque se puede, y de hecho se aconseja, una utilización conjunta.

La filosofía de JSP es similar a la de PHP o ASP: partir de una página HTML e incluir porciones de código para crear contenido dinámico. Ese contenido dinámico se genera en el servidor, el cliente sólo recibe el código HTML final (no se parece en eso a los *applets* o *ActiveX*[2]).

Realmente JSP es una tecnología que se apoya en los *servlets*. La primera vez que se accede a una página JSP el servidor genera y compila automáticamente un *servlet* que produce una salida equivalente. Es este *servlet* el que se ejecuta realmente.

El código JSP se inserta dentro de etiquetas de la siguiente forma:

```
<% .... %>
```

Existe una lista de objetos implícitos que pueden utilizarse directamente en el código JSP. Los más importantes son:

- request: representa la petición recibida por parte del usuario.
- response: la respuesta de JSP a la petición.
- session: la sesión actual que está utilizando el usuario.

[2]Tecnología similar a los *applets* de JAVA. Creada por *Microsoft*.

- application: es el contexto de la aplicación Web.
- out: *stream* de salida asociado a la respuesta del JSP.

Es muy frecuente el uso de un script JSP para la impresión de una determinada información. Existe la siguiente abreviatura para este fin:

```
<%= .... %>
```

Así por ejemplo, las dos líneas siguientes son equivalentes:

```
<%= request.getParameter("edad")%>

<% out.println(request.getParameter("edad")%>
```

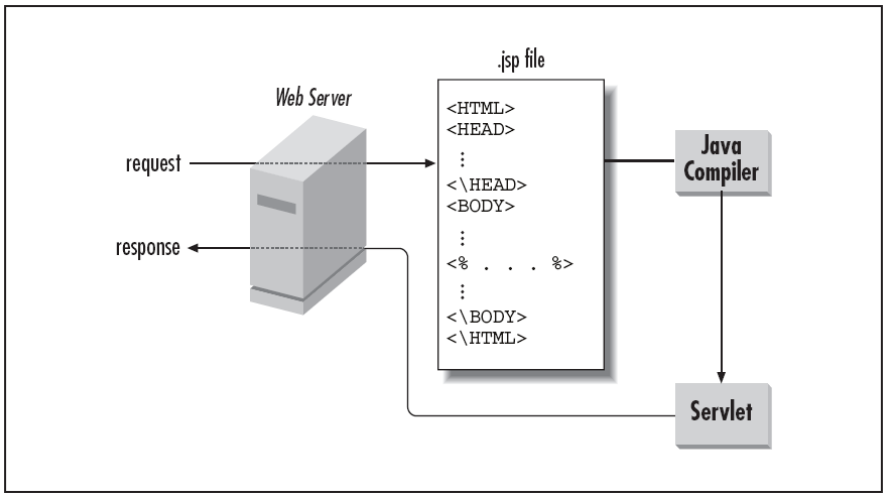

Figura A.3: Arquitectura JSP ([44]).

Podemos diferenciar ciertos elementos dentro del código JSP. Tenemos las directivas JSP, que son unos elementos especiales indicados al comienzo de la página que permiten configurar ciertos aspectos como por ejemplo, si se requiere soporte de sesiones o no:

```
<%@ page session = "true" %>
```

También permiten indicar los paquetes de clases que se van a necesitar en la página JSP:

```
<%@ page import = "java.util.*" %>
```

Para declarar atributos y operaciones existe un elemento especial. Hay que pensar que esto es debido a que un *script* JSP se va a transformar en un *servlet*:

```
<%! .... %>
```

El atributo *k* y la operación *incrementarK()* se definirían como sigue:

```
<%!
int k = 0;
void incrementarK () { ++k; }
%>
```

El siguiente ejemplo ilustra los distintos elementos de una página JSP:

```
<%@ page session="false" %>
<%!
String generarMensaje (String nombre){
    if (nombre != null)
      return "Bienvenido, " + nombre + ".";
    else
      return "No sé tu nombre";
}
%>

<html>
<head> <title>Prueba de JSP</title> </head>
<body>
<h1>JSP Ejecutado</h1>
<%
String s = generarMensaje (request.getParameter ("nombre"));
out.println (s);
%>
</body>
</html>
```

Realmente puede construirse una aplicación Web utilizando únicamente páginas JSP. Sin embargo, esto se considera un abuso. Se recomienda utilizar JSP para diseñar páginas Web fundamentalmente estáticas con ciertos elementos dinámicos. Si es necesario incluir en la página una gran cantidad de código de procesamiento no relacionado directamente con la generación de HTML, entonces es preferible la utilización de *servlets*.

Otra opción es la mezcla de ambas tecnologías para conseguir un rendimiento mayor. De esta forma, cada tecnología se utiliza para lo que ha sido concebida, proporcionando mayor productividad y eficiencia.

A.3. Patrones de diseño

Son muchas las razones por las que se justifica un patrón de diseño cuando se implementa la interfaz de una aplicación. Algunas de las más importantes son las siguientes:

- **Reducción del tiempo de desarrollo.** Un buen diseño va a ayudar a dividir el sistema en tareas más simples que reducirán el esfuerzo de implementación.
- **Reducción del tiempo de mantenimiento.** El mantenimiento puede ser insoportable si no se utiliza un buen patrón de diseño.
- **Colaboración entre desarrolladores.** El grupo de desarrolladores que realiza el sistema puede ser dividido en subgrupos. A cada uno se les asignará la tarea que mejor se acomode a sus habilidades.

Nos centraremos en un patrón de diseño en particular que atiende a la separación clara entre la lógica de negocio y la funcionalidad de la página Web. Vamos a examinar un primer modelo de implementación directa, para más tarde ver el diseño aconsejado y el que hemos utilizado en el *Proyecto*, el ***Modelo-Vista-Controlador***.

A.3.1. Modelo 1

Normalmente se suele identificar este primer modelo con un enfoque primerizo y directo que se usa cuando se programa con JSP y *servlets*. La arquitectura que hay detrás de este modelo es bastante simple: se codifican directamente las necesidades funcionales que se van teniendo.

Si se necesita seguridad, se codifica directamente; si queremos devolver un conjunto de características de una BD, implementamos un módulo que nos presente en la página Web las características de esa BD.

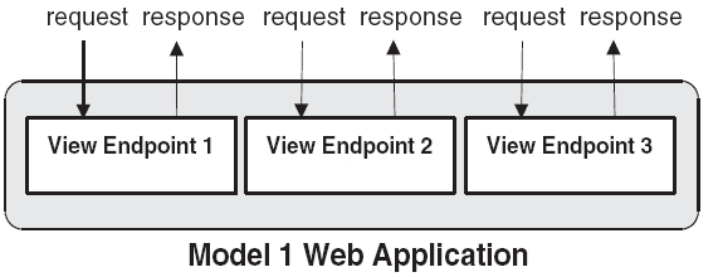

Figura A.4: Modelo 1 ([42]).

La típica página Web común realizada con el lenguaje de programación PHP en dónde se incrusta código con sentencias SQL dentro de una página HTML es un claro ejemplo de ello.

Como es lógico, para aplicaciones triviales éste es el estilo que habrá que seguir, ya que utilizar modelos más avanzados sería como *matar moscas a cañonazos*. El problema está en realizar aplicaciones complejas o de cierta entidad utilizando este mismo enfoque.

El siguiente código ilustra claramente el *Modelo 1*. La página Web se desarrolla con JSP, tiene la funcionalidad de mostrar las noticias que se encuentran almacenadas en un fichero XML:

```
// archivo noticias.jsp

<%@ page import="
java.io.*,
javax.xml.parsers.*,
org.w3c.dom.*" %>
<%
  ServletContext sc = pageContext.getServletContext();
  String dir = sc.getRealPath()"/ruta";
  File file = new File(dir + "/noticias.xml");
  DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
  Document doc = null;
  if(file.exists()){
      doc = db.parse(file);
  }
  if(doc != null){
      NodeList nodes = doc.getElementsByTagName("textoNoticia");
      for(int i=0; i<nodes.getLength(); i++){
          Element e = (Element)nodes.item(i);
%>

   <hr>
   <h3><a href="<%= e.getAttribute("enlaceNoticia") %>">
   <%= e.getAttribute("tituloNoticia") %></a></h3>
   <%= e.getAttribute("textoNoticia") %>

<%
      }   //se cierra llave del bucle for
  }    //se cierra llave del if anterior
%>
```

Aún para usuarios poco acostumbrados a la programación con JSP queda claro que se abre un fichero XML y se van mostrando secuencialmente las entradas de dicho fichero que corresponden a noticias. Incrustado en el código HTML se encuentra un *script* que imprime el título, enlace y texto principal de cada una de las noticias.

Este primer modelo no posee muchas ventajas y no se recomienda su uso, excepto como ya hemos dicho, para casos triviales. En cuanto al funcionamiento, el código anterior funciona perfectamente y da la utilidad deseada.

Este es el modelo utilizado por desarrolladores con poca experiencia y pocas habilidades en JSP y *servlets*. Se utiliza una gran cantidad de código embebido mediante *scripts*. Dichos *scripts* son bastante difíciles de mantener conforme aumenta su número, y es también bastante complicada la colaboración entre los desarrolladores.

Directamente se inserta el código JAVA en la página HTML estática, lo que implica que el desarrollador tiene que tener habilidades tanto de diseño Web como de programación más compleja.

Otro aspecto que puede complicar el mantenimiento de esta arquitectura es la poca independencia que se tiene respecto a la forma de representación de la información. Si en vez de utilizar un archivo XML se utilizara un base de datos en Access por ejemplo, habría que ir modificando uno a uno todos los *scripts* del sistema. **No existe ninguna separación entre el acceso a los datos y la representación de la respuesta que espera el usuario.**

Todo son desventajas en este *Modelo 1*, por lo que se desaconseja su uso y por lo que no ha sido utilizado en absoluto en este *Proyecto Fin de Carrera*.

A.3.2. Modelo 2. Modelo Vista Controlador

Viene a solventar cada uno de los problemas con los que nos encontrábamos en el anterior modelo. Es reconocido como **el mejor método de implementar una aplicación Web usando tecnología JAVA**. Y es también la arquitectura que utiliza el famoso *framework* **Jakarta Struts** [3].

Este *Modelo 2* define una clara separación entre la *lógica de negocio* y la *lógica de representación* de la aplicación Web. La *lógica de negocio* es todo aquello necesario para devolver la información pedida por el usuario, mientras que la *lógica de representación* se encarga de dar formato a esa información que el usuario espera.

Como ya hemos comentado, este modelo también se llama **MVC**, patrón clásico que separa la aplicación en tres componentes:

- **Modelo:** es una representación de los datos de la aplicación y del código encargado de manejar, leer o escribir dicha información.
- **Vista:** es la responsable de interactuar con el usuario. Es lo que el usuario ve y con lo que se relaciona.
- **Controlador:** enlaza las dos componentes anteriores, siendo el responsable de dar la *Vista* apropiada al usuario y de mantener el *Modelo* de la aplicación.

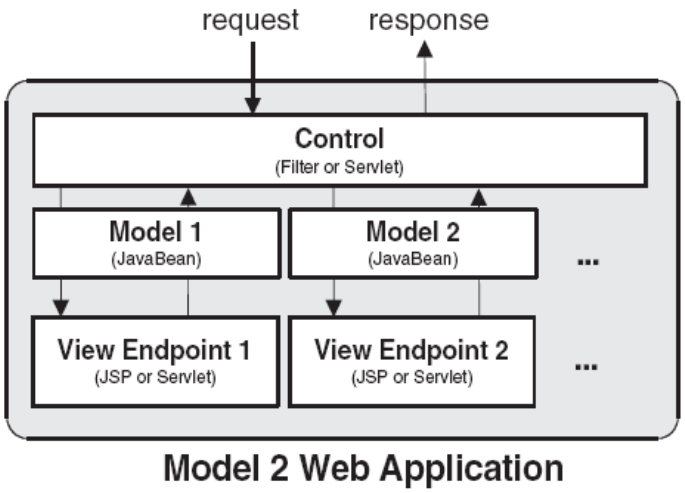

Figura A.5: Modelo Vista Controlador amoldado a la tecnología Servlet/JSP ([42]).

En términos de tecnología Web de JAVA, la *Vista* sería realizada via JSP, ya que es el lenguaje de programación más cercano a HTML o XML, y más fácil de usar por diseñadores. El *Modelo* serían las clases robustas y clásicas de JAVA (como lo hemos hecho nosotros), o bien los **JavaBeans**[4].

Por contra, el *Controlador* suele ser implementado mediante *servlets* o *filtros*, diseñados especificamente para aceptar y dirigir las peticiones y respuestas del cliente.

Siguiendo con la misma funcionalidad de ejemplo que dábamos en el *Modelo 1* (mostrar las noticias de un archivo XML en una página Web) vamos a implementar los distintos componentes que se dan en este modelo.

Empezaremos por el *Controlador* de la aplicación, que será implementado mediante un *filtro* que aceptará todas las peticiones del usuario y ejecutará implícitamente la clase de JAVA que corresponde a cada petición:

```
package com.noticias;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ControlFilter implements Filter{
   protected FilterConfig config = null;
```

[3]Jakarta Struts: `http://struts.apache.org`

[4]Los *JavaBeans* son clases de **J2EE** que tienen que implementar ciertos métodos especiales.

```
public void init(FilterConfig filterConfig){
    config = filterConfig;
}

public void doFilter(ServletRequest req, ServletResponse res,
                     FilterChain chain) throws IOException,
                                               ServletException{

   if(!(req instanceof HttpServletRequest)){
        throw new ServletException("Se requiere peticion HTTP");
   }

   HttpServletRequest request = (HttpServletRequest)req;
   HttpServletResponse response = (HttpServletResponse)res;
   String uri = request.getRequestURI();
   int start = uri.lastIndexOf("/")+1;
   int stop = uri.lastIndexOf(".");
   String name = "default";
   if(start < stop){
        name = uri.substring(start, stop);
   }
   boolean doFilter = true;

   // se intenta ejecutar la clase correspondiente
   try{
        Object o = Class.forName("com.noticias."+name).newInstance();
        if(!(o instanceof Control)){
             throw new ServletException("La clase no implementa Control");
        }

        Control control = (Control)o;
        doFilter = control.doLogic(request, response);
   }
   catch(.....){
   .....
   }

   if(doFilter){
        chain.doFilter(request, response);
   }
}

public void destroy(){
```

```
      // nada
   }
}
```

Esta clase *ControlFilter* opera de la siguiente manera: cuando el usuario pide un recurso, por ejemplo el archivo *index.jsp*, verifica si existe la clase *com.noticias.index* que realizará el proceso de obtención de información necesario. Estas clases tendrán que heredar de una interfaz (*Control*) para que todas ellas implementen un método común, en este caso, el método *doLogic()*:

```
package com.noticias;

import javax.servlet.http.*;
import java.io.IOException;
import javax.servlet.ServletException;

interface Control{
    public boolean doLogic(HttpServletRequest request,
                           HttpServletResponse response)
                             throws ServletException, IOException;
}
```

Esta interfaz será implementada por todas las clases controladoras que serán llamadas por el filtro según el recurso pedido por el usuario. El filtro *ControlFilter* sólo proporciona un mecanismo directo para la lógica de este *Modelo 2*, ya que el filtro lo único que hace es llamar a la clase relacionada con el recurso JSP correspondiente.

También necesitamos definir la lógica individual de cada recurso dinámico, es decir, la clase que será llamada por *ControlFilter* cuando se pida un recurso JSP. Ya sabemos que estas clases implementarán una clase común, la clase *Control*.

Para nuestro ejemplo de noticias habría que implementar una clase *noticias* que maneje la lógica de negocio de la aplicación Web. Su código podría ser el siguiente:

```
// archivo noticias.java

package com.noticias;

import javax.servlet.http.*;
import java.io.*;
import javax.servlet.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
```

```
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;
import org.xml.sax.*;
import java.util.*;

// la clase ha de llamarse como el recurso (noticias.jsp=>noticias)
public class noticias implements Control{

    public boolean doLogic(HttpServletRequest request,
                           HttpServletResponse response)
                             throws ServletException, IOException{
       try{
          ServletContext sc = request.getSession().getServletContext();

          String dir = sc.getRealPath("/noticias");
          File file = new File(dir+"/noticias.xml");
          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
          DocumentBuilder db = dbf.newDocumentBuilder();
          Document doc = null;

          if(file.exists()){
               doc = db.parse(file);
          }
          if(doc != null){
               NodeList nodes = doc.getElementsByTagName("textoNoticia");
               Properties[] ads = new Properties[nodes.getLength()];
               for(int i=0;i<nodes.getLength();i++){
                    Element e = (Element)nodes.item(i);
                    ads[i].new Properties();
                    ads[i].setProperty("enlaceNoticia",
                                       e.getAttribute("enlaceNoticia"));
                    ads[i].setProperty("tituloNoticia",
                                       e.getAttribute("tituloNoticia"));
                    ads[i].setProperty("textoNoticia",
                                       e.getAttribute("textoNoticia"));
               }

               // dejamos como respuesta el conjunto de todas las
               // noticias leídas para que la interfaz las obtenga
               // más tarde
               request.setAttribute("noticias", ads);
          }
```

```
        }
        catch (SAXException e){
           throw new ServletException(e.getMessage());
        }
        catch (ParserConfigurationException e){
           throw new ServletException(e.getMessage());
        }
    }
}
```

La función de esta clase es obtener a partir del archivo XML todas las noticias que haya, guardándolas en una especie de tabla *hash* (contenedor *Properties*) dentro de la petición del usuario (código *request.setAttribute("noticias", ads)*).

Así, la interfaz podrá obtener dicho contenedor de noticias sin tener que saber de dónde proceden esas noticias ni como se han obtenido.

El recurso encargado de la interfaz, *noticias.jsp*, se encontrará directamente en del objeto *request* el conjunto de noticias que tiene que mostrar (se las encontrará sin pedirlas gracias a la labor del filtro *ControlFilter* y de la clase *noticias*). Mostramos su código a continuación:

```
<%@ taglib uri="http://java.sun.com/jstl/core_rt" prefix="c" %>

<!--para cada noticia guardada en request obtenemos titulo, enlace
y texto para mostrarlo por pantalla --!>

<c:forEach var="not" begin="0" items="${noticias}">
  <hr>
  <h3>
     <a href=${not.enlaceNoticia}">${not.tituloNoticia}</a>
  </h3>
  ${not.textoNoticia}
</c:forEach>
```

La diferencia fundamental entre el código en JSP y el código del *Modelo 1* es que ahora es muy fácil de mantener. Es tan fácil como editar una página Web. El diseñador puede centrarse en aspectos estéticos dejando de un lado temas de implementación.

Ni que decir tiene que los dos modelos aplicados al ejemplo producen el mismo resultado en la página Web.

La aplicación ha sido dividida en múltiples capas, una *clase filtro* que actúa como *Controlador*, una clase que obtiene la información y que es parte del *Modelo*, y un *script JSP* (*Vista*) muy unido a la interfaz Web, que genera la presentación a partir de la información dada por el *Modelo*. En ningún momento hay una mezcla entre la *lógica de negocio* y la de

representación gracias a un filtro que acepta todas las peticiones del usuario, preprocesa esa petición, y la manda al *script* para que genere la respuesta en la interfaz Web.

Este modelo presenta muchas ventajas en comparación con el primero, las páginas están limpias y son elegantes. El mantenimiento es simple. Una modificación en el diseño de la página Web nunca afectará accidentalmente a la representación de la información.

Una de las partes más importantes de este modelo es el uso del filtro. También es una tecnología útil si queremos añadir seguridad, restricciones de acceso o manejo de errores. Podríamos reutilizar un filtro que controle el acceso de los usuarios y poder integrarlo fácilmente en otras aplicaciones.

Por consiguiente, una posible mejora a este modelo es la utilización de filtros reutilizables, así tendríamos componentes de la aplicación que podríamos utilizar en cualquier momento de una manera cómoda, con el gran ahorro de tiempo que supondría.

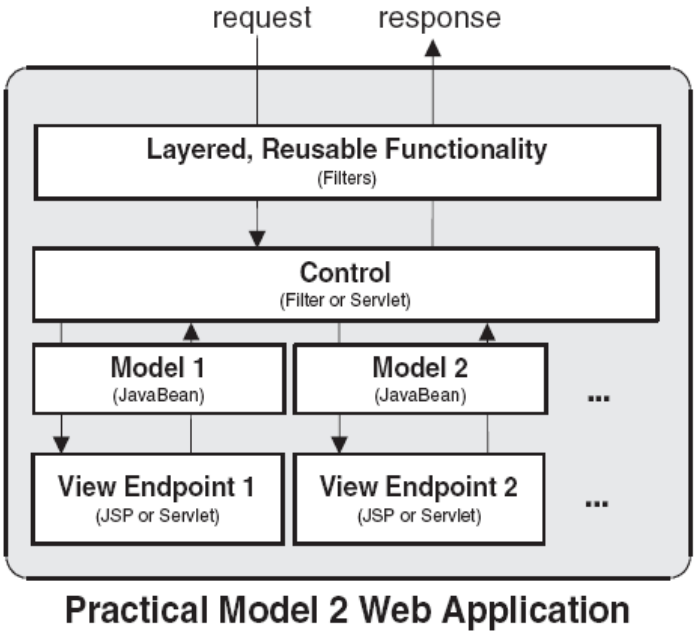

Figura A.6: Mejora de reutilización sobre el Modelo 2 ([42]).

A.3.3. Modelo $1\frac{1}{2}$

Uno de los pocos inconvenientes que tiene el *Modelo 2* es que inicialmente puede ser un poco traumático acomodarnos a él, incluso tedioso. Además, cuando estamos realizando muchos cambios en el código, la recompilación de todas las clases del servidor es más lento y pesado que el cambio en un simple *script* JSP.

El porqué introducimos el *Modelo* $1\frac{1}{2}$ en este apéndice es debido a que es una revisión muy práctica del *Modelo 2* cuando estamos dentro de la etapa de desarrollo del sistema. El *Modelo* $1\frac{1}{2}$ es una manera muy buena de desarrollar rápidamente una aplicación de forma similar al *Modelo 2* usando sólo JSP.

Una vez desarrollado el prototipo de la aplicación, este código JSP puede ser trasladado fácilmente a un *Modelo 2* real, encapsulando la lógica de negocio en clases de JAVA tal y como hemos visto en el anterior apartado.

El funcionamiento de este modelo es simple, primero se desarrolla el código JSP que va a emular a la parte de negocio del *Modelo 2*. Luego se crean los *scripts* de la interfaz que acceden a esos objetos de negocio.

En la práctica, este patrón de diseño es utilizado para escribir código volátil rápidamente. Una vez que ese código ha sido probado, se porta la parte de la *lógica de negocio* a clases de JAVA separadas y se borra el antiguo código JSP.

Haciéndolo así se mejora el tiempo de desarrollo gracias a la primera implementación en JSP, mientras que el mantenimiento futuro se asegura gracias a la modularidad de las clases JAVA.

Nosotros hemos utilizado desde un primer momento el *Modelo 2*, ya que el sistema no es tan grande como para tener que realizar, primero una aproximación mediante JSP, y luego un traspaso de código a clases de JAVA.

Apéndice B

Formato de los datos de entrada y ficheros XML

Formato de los datos de entrada

El formato de los ficheros de BBDD son los utilizados por KEEL[5] y deben seguir el siguiente patrón para que la aplicación funcione correctamente:

- Especificación del nombre del *dataset* (obligatorio):

  ```
  @relation name
  ```

- Atributos del *dataset* (obligatorio):

  ```
  @attribute name integer [{min, max}]
  @attribute name real [{min, max}]
  @attribute name {value1, value2, ... , valueN}
  ```

  (Si el usuario no facilita el intervalo de cada atributo, éste será obtenido del conjunto de datos).

- Especificación de los atributos de entrada y salida *inputs/outputs* (opcional):

  ```
  @inputs name1, name2, ..., nameX
  @outputs name1, name2, ..., nameY
  ```

- Datos en el orden en el que los atributos fueron definidos (obligatorio):

[5]KEEL: `http://www.keel.es`

```
@data
x11, x12, ... , x1N
x21, x22, ... , x2N
```

La cadena que especificará un valor perdido en los datos será ***null***.

Los archivos serán guardados con la extensión *.dat* en la medida de lo posible. Un ejemplo de fichero de datos puede ser el siguiente:

```
@relacion paint
@attribute colour {yellow, white, black}
@attribute amount integer [1, 10]
@attribute density real [0.1, 2.5]
@inputs colour, amount
@outputs density
@data
yellow, 4, 1.2
black, 1, 2.1
yellow, 2, 0.8
white, 8, 0.9
```

Formato del fichero de almacenamiento del multiclasificador

Son los ficheros que generará el sistema Web cuando haya aprendido un multiclasificador. Almacenará la ruta del fichero de entrenamiento, tipo de clasificador y parámetros de este, características seleccionadas, número de clasificadores del multiclasificador así como el peso de cada uno de ellos.

Se ha utilizado XML por su portabilidad. Un ejemplo de fichero sería el siguiente, que contiene dos clasificadores, el primero es un 1NN con un peso de 0.72, y el segundo un 3NN con 0.28 de pesado. Se ha aplicado sobre un problema de 30 características:

```
<multiclassifier id="multi01">
   <num_classifiers>2</num_classifiers>
   <training>/bd_ion/ion-10-4tra.dat</training>
   <num_features>30</num_features>

   <classifier id="cls01">
      <type>knn</type>
      <weight>0.72</weight>
      <parameters id="param01">
         <k>1</k>
      </parameters>
```

```
      <features id="feat01">
          <selected>yes</selected>
          <selected>no</selected>
          <selected>yes</selected>
                  ...
          <selected>yes</selected>
          <selected>no</selected>
          <selected>no</selected>
      </features>

   </classifier>

   <classifier id="cls02">
      <type>knn</type>
      <weight>0.28</weight>
      <parameters id="param02">
          <k>3</k>
      </parameters>

      <features id="feat02">
          <selected>yes</selected>
          <selected>no</selected>
          <selected>yes</selected>
                    ...
          <selected>yes</selected>
          <selected>no</selected>
          <selected>no</selected>
      </features>
   </classifier>
</multiclassifier>
```

Para cada característica y clasificador existe una entrada que nos especifica con *yes* o *no* si se ha seleccionado la características o no.

Formato del fichero de resultados de otras técnicas

Este fichero es interno del sistema pero también hemos considerado reseñar brevemente su estructura. Se utiliza para comparar los resultados de clasificación del sistema con otras técnicas. Para ello habrá un archivo por problema. De cada método se guardará su nombre, error en *test* con el **1NN** y número de características seleccionadas.

Veamos un ejemplo del fichero:

```
<results id="res01">
     <database>wisconsin</database>

     <method id="meth01">
        <name>LVF</name>
        <approach>filter</approach>
        <error>0.234</error>
        <features>12</features>
     </method>

     <method id="meth02">
        <name>Backward</name>
        <approach>filter</approach>
        <error>0.334</error>
        <features>11</features>
     </method>

     <method id="meth03">
        <name>Forward</name>
        <approach>filter</approach>
        <error>0.25</error>
        <features>19</features>
     </method>
</results>
```

Apéndice C

Otras tablas de resultados

	IONOSPHERE		VEHICLE		WISCONSIN	
	HIB	**STD**	**HIB**	**STD**	**HIB**	**STD**
PARTICIÓN 1	6749	1322	699	253	377	4485
PARTICIÓN 2	6879	724	1368	2638	232	773
PARTICIÓN 3	10916	529	498	3555	9317	1448
PARTICIÓN 4	8230	717	684	463	14879	636
PARTICIÓN 5	5754	1202	1124	144	311	6752
PARTICIÓN 6	15307	858	440	155	2309	3969
PARTICIÓN 7	13502	911	7153	521	5967	6080
PARTICIÓN 8	3146	17385	789	4956	19618	3363
PARTICIÓN 9	18120	456	7153	3449	7226	3486
PARTICIÓN 10	10656	809	3938	982	13900	4037
MEDIA	9925,9	2491,3	2384,6	1711,6	7413,6	3502,9

Tabla C.1: Evaluaciones necesarias para conseguir el individuo con mejor precisión en NSGA-II.

	IONOSPHERE		VEHICLE		WISCONSIN	
	HIB	**STD**	**HIB**	**STD**	**HIB**	**STD**
PARTICIÓN 1	6829	3450	178	839	3742	1732
PARTICIÓN 2	1081	2591	16805	108	578	1752
PARTICIÓN 3	2590	1401	2884	1877	3025	2682
PARTICIÓN 4	4818	2851	721	2656	8521	1607
PARTICIÓN 5	2413	1832	4792	1257	13967	1500
PARTICIÓN 6	3236	2566	329	861	596	719
PARTICIÓN 7	9565	3255	380	4058	12622	1475
PARTICIÓN 8	12122	2134	218	1071	352	1146
PARTICIÓN 9	9555	2047	8772	1663	19835	1525
PARTICIÓN 10	19516	1585	1390	858	12281	2009
MEDIA	7172,5	2371,2	3646,9	1524,8	7551,9	1614,7

Tabla C.2: Evaluaciones necesarias para conseguir el individuo con mejor precisión en SPEA2.

	IONOSPHERE		VEHICLE		WISCONSIN	
	HIB	**STD**	**HIB**	**STD**	**HIB**	**STD**
PARTICIÓN 1	7794	185	7935	105	1783	112
PARTICIÓN 2	13467	209	8577	103	571	1253
PARTICIÓN 3	1274	136	6726	309	7506	257
PARTICIÓN 4	1480	126	14124	194	2646	647
PARTICIÓN 5	5646	127	163	116	1968	375
PARTICIÓN 6	16209	283	756	105	3002	169
PARTICIÓN 7	14452	380	12691	116	6525	358
PARTICIÓN 8	10570	283	1710	236	10657	280
PARTICIÓN 9	15109	288	17271	116	1557	98
PARTICIÓN 10	2844	768	2974	212	1278	214
MEDIA	8884,5	278,5	7292,7	161,2	3749,3	376,3

Tabla C.3: Evaluaciones necesarias para conseguir el individuo con mejor precisión en ϵMOEA.

	FORWARD		BACKWARD		LVF		GRASP	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	13,89 %	8	11,11 %	30	5,56 %	10	11,11 %	9
PART.2	8,57 %	6	14,29 %	27	11,43 %	10	20,00 %	9
PART.3	8,33 %	5	13,89 %	29	13,89 %	10	8,33 %	9
PART.4	11,76 %	4	11,76 %	32	8,82 %	10	8,82 %	9
PART.5	14,71 %	5	2,94 %	30	14,71 %	10	2,94 %	9
PART.6	2,86 %	5	17,14 %	32	14,29 %	10	5,71 %	9
PART.7	14,29 %	10	14,29 %	32	11,43 %	10	17,14 %	9
PART.8	8,33 %	3	11,11 %	31	11,11 %	10	5,56 %	9
PART.9	14,71 %	3	14,71 %	30	11,76 %	10	11,76 %	9
PART.10	13,89 %	4	13,89 %	31	8,33 %	10	11,11 %	9
MEDIA	11,13 %	5,3	12,51 %	30,4	11,13 %	10	10,25 %	9

Tabla C.4: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para IONOSPHERE (parte 1).

	AGG FILTER		AGG WRAPPER #1		AGG WRAPPER #2	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	16,67 %	5	11,11 %	9	16,67 %	8
PART.2	8,57 %	5	14,29 %	13	11,43 %	8
PART.3	11,11 %	4	5,56 %	12	16,67 %	8
PART.4	2,94 %	5	8,82 %	14	14,71 %	8
PART.5	11,76 %	4	0,00 %	14	14,71 %	8
PART.6	17,14 %	4	11,43 %	13	14,29 %	8
PART.7	22,86 %	5	14,29 %	14	20,00 %	8
PART.8	8,33 %	4	13,89 %	16	13,89 %	8
PART.9	8,82 %	4	8,82 %	12	8,82 %	8
PART.10	13,89 %	5	11,11 %	13	11,11 %	8
MEDIA	12,21 %	4,5	9,93 %	13	14,23 %	8

Tabla C.5: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para IONOSPHERE (parte 2).

	FORWARD		BACKWARD		LVF		GRASP	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	1,18 %	9	1,18 %	14	11,76 %	7	16,47 %	5
PART.2	3,53 %	3	4,71 %	13	21,18 %	7	22,35 %	5
PART.3	0 %	10	7,06 %	14	7,06 %	7	17,65 %	5
PART.4	3,53 %	9	7,06 %	14	5,88 %	7	21,18 %	5
PART.5	4,71 %	9	4,71 %	13	16,47 %	7	25,88 %	5
PART.6	2,35 %	7	1,18 %	13	23,53 %	7	14,12 %	5
PART.7	2,38 %	10	9,52 %	13	9,52 %	6	22,62 %	5
PART.8	3,57 %	5	4,71 %	13	25,00 %	7	22,62 %	5
PART.9	5,95 %	5	1,19 %	13	14,29 %	7	21,43 %	5
PART.10	9,52 %	5	9,52 %	12	8,33 %	7	27,38 %	5
MEDIA	3,67 %	7,2	5,08 %	13,2	14,30 %	6,9	21,17 %	5

Tabla C.6: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para VEHICLE (parte 1).

	AGG FILTER		AGG WRAPPER #1		AGG WRAPPER #2	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	3,53 %	3	2,35 %	9	9,41 %	5
PART.2	3,53 %	3	7,06 %	9	15,29 %	5
PART.3	2,35 %	3	0 %	10	4,71 %	5
PART.4	4,71 %	3	3,53 %	10	14,12 %	5
PART.5	8,24 %	3	4,71 %	8	16,47 %	6
PART.6	3,53 %	3	4,71 %	8	17,65 %	6
PART.7	10,71 %	2	2,38 %	10	11,90 %	6
PART.8	2,38 %	3	2,38 %	10	11,90 %	5
PART.9	7,14 %	2	4,76 %	10	10,71 %	5
PART.10	14,29 %	2	4,76 %	10	9,52 %	5
MEDIA	6,04 %	2,7	3,66 %	9,4	12,17 %	5,3

Tabla C.7: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para VEHICLE (parte 2).

	FORWARD		BACKWARD		LVF		GRASP	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	3,51 %	8	5,26 %	26	5,26 %	8	7,02 %	7
PART.2	0 %	4	3,51 %	26	8,77 %	8	5,26 %	7
PART.3	1,75 %	5	1,75 %	24	5,26 %	7	1,75 %	7
PART.4	3,51 %	6	1,75 %	26	7,02 %	7	5,26 %	7
PART.5	14,04 %	3	3,51 %	23	7,02 %	8	7,02 %	7
PART.6	1,75 %	5	5,26 %	26	3,51 %	8	12,28 %	7
PART.7	14,04 %	5	3,51 %	23	12,28 %	8	10,53 %	7
PART.8	3,51 %	11	7,02 %	25	5,26 %	8	5,26 %	7
PART.9	8,77 %	7	12,28 %	25	10,53 %	9	14,04 %	7
PART.10	14,29 %	3	3,57 %	27	8,93 %	8	7,14 %	7
MEDIA	6,52 %	5,7	4,74 %	25,1	7,38 %	7,9	7,56 %	7

Tabla C.8: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para WISCONSIN (parte 1).

	AGG FILTER		AGG WRAPPER #1		AGG WRAPPER #2	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	5,26 %	4	5,26 %	15	3,51 %	7
PART.2	1,75 %	3	7,02 %	15	5,26 %	7
PART.3	0 %	3	0 %	14	1,75 %	7
PART.4	5,26 %	3	1,75 %	18	7,02 %	7
PART.5	7,02 %	3	5,26 %	13	5,26 %	7
PART.6	10,53 %	2	5,26 %	18	10,53 %	7
PART.7	1,75 %	4	3,51 %	18	5,26 %	7
PART.8	3,51 %	3	8,77 %	13	7,02 %	7
PART.9	10,53 %	3	10,53 %	18	10,53 %	7
PART.10	14,29 %	3	8,93 %	18	5,36 %	7
MEDIA	5,99 %	3,1	5,63 %	16	6,15 %	7

Tabla C.9: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para WISCONSIN (parte 2).

	FORWARD		BACKWARD		LVF		GRASP	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	8,49 %	5	17,30 %	48	20,13 %	19	19,50 %	15
PART.2	8,49 %	7	18,55 %	51	14,51 %	23	16,98 %	15
PART.3	24,53 %	1	17,30 %	54	18,93 %	15	19,18 %	15
PART.4	11,95 %	6	17,61 %	58	19,18 %	15	16,04 %	15
PART.5	7,86 %	7	14,15 %	52	20,13 %	19	10,38 %	15
PART.6	11,04 %	8	22,40 %	53	20,13 %	19	12,93 %	15
PART.7	9,46 %	7	18,61 %	56	25,87 %	19	15,14 %	15
PART.8	8,52 %	7	15,14 %	56	18,93 %	15	17,03 %	15
PART.9	9,46 %	6	15,14 %	57	18,93 %	15	18,93 %	15
PART.10	7,57 %	5	14,51 %	53	22,40 %	20	17,35 %	15
MEDIA	10,74 %	5,90	17,07 %	53,80	19,91 %	17,90	16,35 %	15

Tabla C.10: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para SPLICE (parte 1).

	AGG FILTER		AGG WRAPPER #1		AGG WRAPPER #2	
	PREC.	CAR.	PREC.	CAR.	PREC.	CAR.
PART.1	9,12 %	3	19,18 %	33	23,27 %	18
PART.2	10,06 %	5	15,09 %	32	21,07 %	18
PART.3	9,12 %	6	21,38 %	27	25,47 %	18
PART.4	9,75 %	5	20,44 %	27	27,36 %	18
PART.5	7,55 %	5	15,41 %	27	22,33 %	18
PART.6	9,15 %	4	15,77 %	33	22,71 %	18
PART.7	9,15 %	6	18,61 %	29	23,66 %	18
PART.8	9,46 %	5	12,93 %	22	22,08 %	18
PART.9	8,52 %	5	15,14 %	21	24,92 %	18
PART.10	9,46 %	5	17,03 %	36	25,87 %	18
MEDIA	9,13 %	4,90	17,10 %	28,70	23,87 %	18

Tabla C.11: Características seleccionadas y precisión en *test* de técnicas clásicas de SC para SPLICE (parte 2).

Apéndice D

Contenidos digitales del CD-ROM

- Código del sistema de Minería de Datos.
- Archivos de instalación de la máquina virtual de **JAVA** y del servidor Web **TomCat**, necesarios para la puesta en marcha del sistema.
- Documentación detallada en inglés de todas las clases del sistema en formato HTML y siguiendo el estándar de documentación de JAVA.
- Memoria del *Proyecto Fin de Carrera* en formato PDF.
- Archivos de las bases de datos utilizadas en la experimentación:
  - **IONOSPHERE**.
  - **VEHICLE**.
  - **WISCONSIN**.
  - **SPLICE**.

Bibliografía

Computación Evolutiva y EMO

[1] T. Bäck, D.B. Fogel, Z. Michalewicz. *Handbook of Evolutionary Computation*. Institute of Physics Publishing, Bristol and Oxford University Press, New York, 1997

[2] C.A. Coello y G. Toscano. *A Micro-Genetic Algorithm for Multiobjective Optimization*. First International Conference on Evolutionary Multi-Criterion Optimization p.126-140, Ed. Springer-Verlag, 2001

[3] C.A. Coello Coello. *Evolutionary Multiobjective Optimization: Current and Future Challenges*. Advances in Soft Computing—Engineering, Design and Manufacturing pp. 243 - 256, Springer-Verlag, 2003

[4] D.W. Corne, J.D. Knowles y M. Oates. *The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization*. Proceedings of the Paralell Problem Solving from Nature VI Conference 839-848, Ed. Springer, Paris 2000

[5] C. Darwin. *The Origin of Species*. John Murray, 1859

[6] K. Deb. *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II*. IEEE Transactions on Evolutionary Computation, VOL.6, NO.2, 2002

[7] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. Editorial WILEY, 2001

[8] K. Deb. *A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions*. KanGAL Report Number 2003002, Kanpur (India), 2003

[9] L.J. Eshelman. *The CHC Adaptative Search Algorithm: how to have safe search when engaging in Nontraditional Genetic Recombination*. Foundations of Genetic Algorithms, 265-283, Morgan Kaufmann Publishers, 1991

[10] L.J. Fogel, A.J. Owens y M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Wiley, Chichester, UK, 1966

[11] C.M. Fonseca y P. Fleming. *Genetic Algorithms for multiobjective optimization, discussion and generalization.* Proceedings of the Fifth International Conference on Genetic Algorithms, 416-423, Morgan Kaufmann Publishers, 1993

[12] J.R. Koza *Genetic Programming.* MIT Press, Cambridge, MA, 1992

[13] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Reading, MA, 1989

[14] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, MA, 1992. Primera edición: 1975, The University of Michigan Press, Ann Arbor

[15] J. Horn, N. Nafpliotis y D.E. Goldberg. *A Niched pareto genetic algorithm for multiobjective optimization.* In the Proceedings of the First Conference on Evolutionary Computation, junio 1994

[16] J.D. Knowles y D.W. Corne. *Approximating the nondominated front using the pareto archived evolution strategy.* Evolutionary Computation 8(2):149-172, 2000

[17] J.B. Kollat y P. Reed. *The Value of Online Adaptive Search: A Performance Comparison of NSGA-II, ϵ-NSGA-II y ϵMOEA.* Dpto. de Ingeniería Civil y Medioambiental, Universidad de Pennsylvania, 2005

[18] M. Laumanns, L. Thiele, K. Deb y E. Zitzler. *Combining Convergence and Diversity in Evolutionary Multi-objective Optimization.* IEEE Transactions on Evolutionary Computation 10:3, Fall, pp. 263 - 282, 2002

[19] H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie.* Vol. 26 de ISR. Birkhaesuer, Basel/Stuttgart, 1977

[20] N. Srinivas y K. Deb. *Multiobjective optimization using nondominated sorting in genetic algorithms.* Evolutionary Computation, 2(3):221-248, 1994

[21] D.H. Wolpert y W.G. Macready. *No free lunch theorems for optimization.* IEEE Transactions on Evolutionary Computation, abril 1997, 67-82

[22] E. Zitzler, M. Laumans y L. Thiele. *SPEA2: Improving the Strength Pareto Evolutionary Algorithm.* TIK-Report 103, ETH Zurich (Suiza), 2001

[23] E. Zitzler. *A Tutorial on Evolutionary Multiobjective Optimization.* ETH Zurich (Suiza), 2002

[24] E. Zitzler. *Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach.* IEEE Transactions on Evolutionary Computation, Vol.3 No.4, noviembre, 1999

Minería de Datos

[25] H. Almuallim y T. Dietterich. *Learning with many irrelevant features.* In Proceedings of the Ninth National Conference on Artificial Intelligence, The MIT Press, 1991

[26] B.V. Dasarathy y B.V. Sheela. *A composite classifier system design: concepts and methodology.* Proceedings of IEEE, 67:708-713, 1978

[27] B.V. Dasarathy. *Nearest Neighbour (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press, Los Alamitos, California, 1990

[28] M.H. Dunham. *Data Mining. Introductory and advanced topics.* Pearson Education, 2003

[29] D. Hand, H. Mannila y P. Smyth. *Principles of Data Mining.* The MIT Press, 2001

[30] J. Kittler y F. Roli (Eds.). *Multiple Classifier Systems.* Editorial Springer, 2000

[31] K. Kira y L. Rendell. *A practical approach to feature selection.* Proceedings of the Ninth International Conference on Machine Learning, p.249-256, Morgan Kaufmann, 1992

[32] L.I. Kuncheva. *Fuzzy Classifier Design.* Physica-Verlag Heidelberg, 2000

[33] H. Liu y H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining.* Editorial Kluwer Academic Publishers, 1998

[34] H. Liu y R. Setiono. *Feature Selection and classification - a probabilistic wrapper approach.* In Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES p.419-424, 1996

[35] H. Liu y R. Setiono. *A probabilistic approach to feature selection - a filter solution.* Proceedings of International Conference on Machine Learning p.319-327, Morgan Kaufmann Publishers, 1996

[36] D. Pyle. *Data Preparation for Data Mining.* Morgan Kaufmann, 1999

[37] J.R. Quinlan. *C4.5 Programs for Machine Learning.* Morgan Kaufmann, Artificial Intelligence, 1988

[38] I. H. Witten y E. Frank. *Data Mining. Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2005

[39] K. Woods, W.P. Kegelmeyer y K. Bowyer. *Combination of multiple classifiers using local accuracy estimates.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 19:405-410, 1997

[40] L. Xu, A. Krzyzak y C.Y. Suen. *Methods of combining multiple classifiers and their application to handwriting recognition.* IEEE Transactions on Systems, Man, and Cybernetics, 22:418-435, 1992

Tecnología JAVA y Servlets/JSP. Ingeniería del Software

[41] R. Battiti. *Using Mutual Information for Selecting Features in Supervised Neural Net Learning*. IEEE Transactions on Neural Network, 1994

[42] J. Falkner y K. Jones. *Servlets and Java Server Pages. The J2EE Technology Web Tier*. Editorial Addison-Wesley, 2003

[43] J. García de Jalón et al. *Aprenda Servlets de Java como si estuviera en segundo*. Escuela Superior de Ingenieros Industriales, Univ. de Navarra, 1999

[44] J. Hunter. *Java Servlet Programming*. Editorial O'Reilly, 1998

[45] Robert C. Martin. *UML for Java Programmers*. Editorial Pearson Education, 2003