Universidad Carlos III de Madrid

Escuela Politécnica Superior

Proyecto Fin de Carrera

Técnicas mono y multiobjetivo para la evolución de transformaciones lineales en problemas de clasificación

Autor:
Iván Monteagudo García

Tutores:
Ricardo Aler Mur
José María Valls Ferrán

Índice general

Índice de tablas

Índice de figuras

Capítulo 1

Introducción

La clasificación automatizada es un campo en el que existe una gran diversidad de soluciones, tanto basadas en robustos planteamientos matemáticos como combinaciones de técnicas estadísticas o aquellas con menos trasfondo teórico y desarrolladas únicamente teniendo en cuenta sus resultados.

Una de las técnicas de clasificación más sencillas y conocidas es K-Vecinos, que se basa en calcular la similitud de lo que se quiere clasificar con una base de ejemplos conocidos. La mayor ventaja de esto es que la probabilidad de éxito del clasificador aumenta con el número de ejemplos. Existe, sin embargo, una limitación en cuanto a los tipos de elementos con los que este tipo de clasificadores funcionan correctamente, es decir, hay ámbitos o dominios, aparentemente simples en muchos casos, en los que estas técnicas no clasifican correctamente los elementos. Hay varios motivos para esto, uno de ellos es que los datos no se distribuyan de una forma organizada, es decir, que los métodos más comunes de calcular las diferencias entre elementos no produzcan resultados significativos en el sentido de diferenciar unas clases de otras.

Una solución para este problema es encontrar una manera sistemática de distribuir los datos (tanto los conocidos como los nuevos que se quieran clasificar) para que, con un cálculo estándar de las diferencias -*distancias*- sí se obtenga información fiable sobre la pertenencia de un elemento a una clase.

Con esto, el problema deriva en encontrar la forma de distribuir los datos. Sin un análisis matemático y estadístico del dominio lo que queda

es la búsqueda entre las posibles distribuciones que, aún reduciéndolas a un número finito, resultaría inabarcable si se quisieran comprobar todas de forma sistemática.

El problema de buscar entre un conjunto de soluciones sin poder comprobarlas todas es muy común en informática, y se han ideado y desarrollado multitud de técnicas -*heurísticas*- para afrontarlo. Las técnicas evolutivas, basadas con diferentes niveles de detalle en la evolución natural, han mostrado ser un mecanismo excelente para trabajar con este tipo de problemas, en los que la única información que se dispone es la forma que tienen la soluciones y cómo juzgarlas.

Un argumento en contra de las técnicas evolutivas es que, al tener en cuenta únicamente la forma de calcular la solución como argumento a la hora de elegir una solución u otra, se debe codificar en éste cálculo todo lo que se quiera obtener de la solución final; si, por ejemplo, además de querer la mejor clasificación posible, queremos que sea de entre las mejores la más sencilla, tendremos que codificar en el cálculo que juzga la solución lo que entendemos por *sencillez*.

Esto ya es de por sí bastante complicado, pero además tendría que construirse un método para decidir qué es más importante de entre las razones para elegir una solución u otra -*objetivos*-, estableciendo una escala de prioridades entre ellas.

De nuevo, al ser éste un problema bastante común, existen una serie de algoritmos que lo tienen en cuenta y permiten buscar, de entre todas las soluciones, las que mejor cumplen los objetivos y elegir la que mantenga el balance deseado entre todos ellos.

Unificando todas estas ideas, este proyecto trata sobre la construcción de un clasificador con K-Vecinos mejorado aplicando transformaciones lineales a los datos del dominio de clasificación. Estas transformaciones se representan mediante matrices obtenidas a partir de técnicas evolutivas mono y multiobjetivo, así como su evaluación mediante diversas baterías de pruebas, comparándolo con métodos más simples.

Capítulo 2

Contexto

2.1. Algoritmos de Clasificación

Los algoritmos de clasificación[1] son técnicas que permiten identificar la pertenencia de un elemento –*instancia*– con unas determinadas características –*atributos*– a un grupo –*clase*– de forma automática.

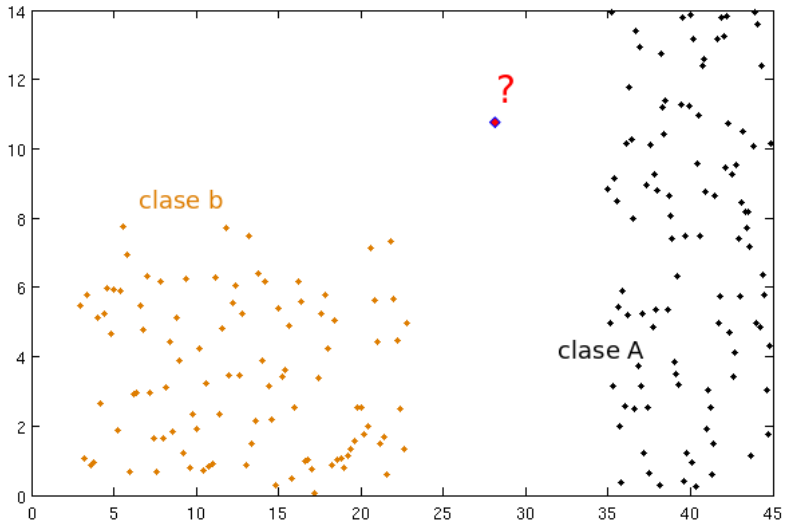
Generalmente se suelen dividir las técnicas de clasificación en dos grandes grupos:

- Clasificación supervisada: Se conocen los grupos a los que puede pertenecer una instancia y es posible determinar si la clasificación es correcta o no. v.g. K-Medias, K-Vecinos, Redes Neuronales[2].
- Clasificación no supervisada: En este caso no se conocen las clases en las que se quieren clasificar los datos desde un principio; lo que se busca es encontrar algún tipo de organización automática de los datos, posiblemente para su posterior análisis. v.g. Mapas autoorganizativos, EM (Expectation Maximization), K-Medias.

2.1.1. K-Vecinos

K-Vecinos[3] (en inglés K-NN, *K-Nearest Neighbour*) es un algoritmo supervisado que se basa en encontrar instancias ya conocidas lo más parecidas posible a la que se quiere clasificar. Para ello se calculan las diferencias entre

Figura 2.1: **Clasificación con K-Vecinos(A).**



los atributos y se determina que la nueva instancia es de la misma clase que la mayoría de las conocidas más parecidas, con K el número de ejemplos más parecidos a tener en cuenta. Esta idea se puede abstraer geométricamente como posicionar las instancias en un espacio $N - dimensional$ (con N el número de atributos) y calcular las distancias entre ellas (figuras 2.1 y 2.2). Dependiendo del tipo de atributos se podrían utilizar distintas formas de medir la distancia:

Euclídea La forma más común de calcular la distancia.

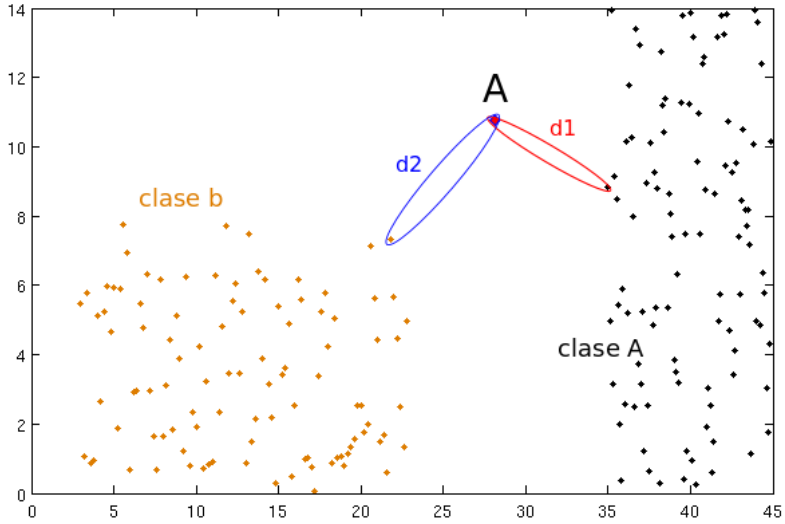
$$\sqrt{\sum_{i=0}^{n}(x_i - y_i)^2} \tag{2.1}$$

Es equivalente a calcular una longitud en un espacio euclídeo, le da la misma importancia a todas las dimensiones sin tener en cuenta posibles diferencias de escala.

Euclídea Generalizada

$$\sqrt{\sum_{i=0}^{n}(x_i - y_i)M_d M_d^T (x_i - y_i)^T} \tag{2.2}$$

Figura 2.2: **Clasificación con K-Vecinos(B).** La distancia d_1 al punto más cercano de la clase A es menor que la distancia al punto más cercano de la clase b d_2, por lo que el nuevo elemento se clasifica como miembro de la clase A.



En similar a la anterior, pero se introduce el término $M_d \in \mathbb{R}^{n \times n}$, llamado *Matriz de distancias*, que permite adaptar la distancia a cualquier espacio.

Hamming Se utiliza cuando los elementos entre los que se quiere calcular la distancia no están definidos de forma numérica, o las diferencias entre estos valores no son significativas de la distancia. Calcular esta distancia consiste simplemente en contar el número de modificaciones que deberían hacerse en un elemento para convertirlo en otro. Por ejemplo, entre dos números binarios se calcularía el número de dígitos distintos: $D_H(01100101, 10100110) = 4$.

Mahalanobis [4]

$$D_M = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)} \tag{2.3}$$

Se utiliza principalmente cuando los vectores entre los que se calcula la distancia pertenecen a una distribución conocida, siendo Σ la matriz de covarianzas. Lo que se consigue es ignorar la escala de los valores.

K-Vecinos es un algoritmo que, aún siendo sencillo, obtiene unos resultados excelentes en muchos dominios, aumentando su eficacia a medida que aumenta el tamaño de la base de ejemplos disponibles y se disponen de ejemplos muy cercanos a todas las nuevas instancias que se quieran clasificar. Sin embargo, si se tiene también en cuenta el rendimiento, éste disminuye a medida que aumenta el número de ejemplos, ya que se calcula la distancia desde cada instancia que se quiere clasificar a todos ellos. Este problema se ha intentado aliviar mediante la agrupación de los ejemplos y la división del espacio en áreas [5], de forma que se calcule la distancia sólo a los ejemplos del mismo área y, o bien se ignore el resto, o se calcule con un nivel de granularidad menor, teniendo en cuenta grupos de ejemplos en lugar de cada uno por separado.

Además de la idea básica de clasificar según los K elementos más cercanos, existen algunas variaciones para adaptar el algoritmo al uso que se le quiera dar. Por ejemplo, se puede dar distinta importancia a los K vecinos según la distancia a la que se encuentren (los más cercanos contarán más, etc.) o se pueden establecer reglas de desempate si K es par: El punto más cercano, un punto aleatorio, etc. Además, se pueden proporcionar niveles de confianza, al estilo de la lógica difusa, sobre la pertenencia del individuo a distintas clases. Así, en lugar de sólo la clase, se obtendría un porcentaje de

probabilidad de pertenencia a cada una de las posibles clases o un nivel de fiabilidad de la clase que se indique.

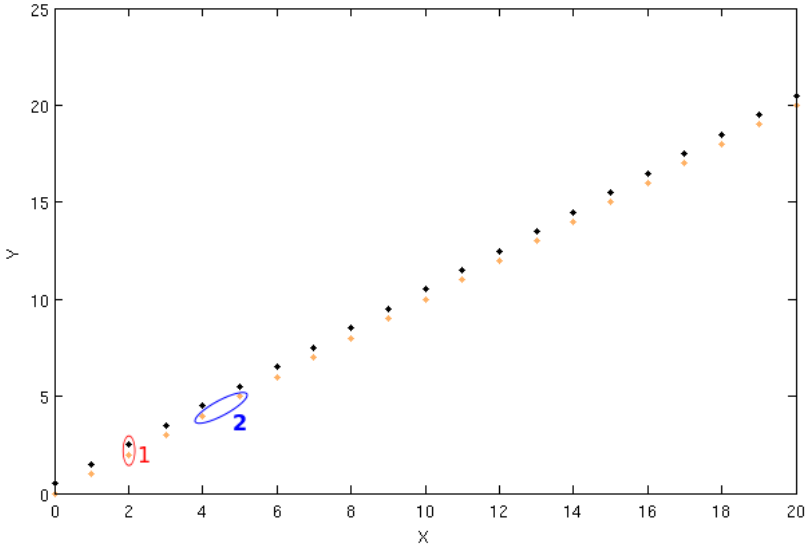
A pesar de las ventajas enumeradas y las distintas mejoras o adaptaciones compatibles con K-Vecinos, existen dominios en los que la forma en la que se calcula la distancia entre instancias no permite la correcta clasificación de los datos, esto puede deberse, entre otros, a los siguientes motivos:

- Falta de homogeneidad entre las escalas de los atributos: Si un atributo toma, por ejemplo, valores en [0,1] y otro los toma en [-10000,10000] con la distancia euclídea que se suele usar por defecto se le dará mucha más importancia al segundo atributo. Para arreglar esto debe utilizarse otra forma de medir la distancia o se deben normalizar los valores de los atributos.
- Falta de una medida de la importancia de cada atributo. Una variable puede ser notablemente más importante que otra a la hora de clasificar un dato, pero para que el algoritmo de clasificación tenga en cuenta esto se debe de utilizar algún mecanismo específico, modificando la forma de calcular la distancia mediante, por ejemplo, valores ponderados.
- Dependencias entre atributos. Si un atributo resulta ser una combinación lineal de otros, puede estar reforzándose la importancia de estos en la clasificación más de lo debido al incluirlo.
- De forma similar, si se introducen atributos que no aporten información útil, el ruido que introducen puede resultar desastroso en la clasificación, al modificar las distancias aleatoriamente.

Por ejemplo, los individuos del dominio de la figura 2.3 pueden parecer fáciles de clasificar a simple vista, sin embargo los puntos están mucho más cerca unos de otros en el eje de ordenadas que en el de abscisas, por lo que todas las instancias se clasificarían incorrectamente si se utilizase la distancia euclídea.

Aunque estos problemas se puedan solucionar de una forma u otra, a medida que aumenta el número de atributos la tarea se vuelve más y más compleja, hasta requerir los datos de un análisis estadístico que puede resultar demasiado complejo o costoso dependiendo del caso. Por ello, encontrar formas de generalizar en la medida de lo posible la capacidad de clasificar de K-Vecinos con la mínima necesidad de manipulación a cuantos más dominios sea posible resulta muy deseable.

Figura 2.3: **Dominio de Ejemplo.** Se puede ver que la distancia entre los puntos en 1 es menor que entre los puntos en 2, aunque estos últimos sean de la misma clase.



2.2. Optimización de distancias

Como se ha explicado, muchos de los problemas que se producen con K-Vecinos para algunos dominios provienen de la forma en la que se calculan las distancias, por esto dos métodos que pueden ayudar a atajar estos problemas son la adaptación automática de la medida de las distancias al dominio o la transformación de los datos del dominio, de forma que una distancia estándar obtenga mejores resultados. Los dos métodos funcionan de forma muy similar:

2.2.1. Adaptación de la distancia

Como se vio en la sección 2.1.1, los algoritmos de clasificación basados en distancias pueden utilizar distintas métodos para calcular éstas y conseguir los mejores resultados. Uno de estos métodos de cálculo de distancias es la Distancia Euclídea Generalizada, que utiliza la fórmula 2.2 para calcular las distancias entre individuos La matriz M, llamada matriz de distancias,

se especifica para poder trabajar con distintos espacios. En su forma más común M es igual a la matriz Identidad I_n con n el número de atributos / dimensiones con las que se esté trabajando.

2.2.2. Transformación de los datos

Se entiende por aplicación o transformación lineal[6] a una aplicación f entre dos espacios vectoriales V, W La aplicación $f : V \to W$ es lineal si:

1. $f(u + v) = f(u) + f(v), \forall u, v \in V$.
2. $f(\alpha u) = \alpha f(u), \forall u \in V, \forall \alpha \in \mathbb{K}$.

Es decir, la suma de las transformaciones es igual a la transformación de la suma y lo mismo para el producto por un escalar.

Una transformación lineal se puede representar mediante una matriz, en particular, una transformación a un espacio con la misma dimensión $f : \mathbb{R}^n \to \mathbb{R}^n$ se puede realizar como producto de un vector $V \in \mathbb{R}^n$ por una matriz $M \in \mathbb{R}^{n \times n}$.

Se puede utilizar entonces una matriz para realizar una transformación de un espacio a otro con la misma dimensión. Con todo esto, se pueden utilizar dicha transformación lineal sobre el dominio de clasificación para, en lugar de modificar la forma de calcular las distancias, volver a posicionar todos los datos y seguir calculando la distancia euclídea estándar. La transformación de los datos se haría mediante la fórmula:

$$\vec{posicion'} = M \times \vec{posicion} \tag{2.4}$$

Dependiendo de los valores en M, esta operación tendrá distinto efecto sobre $\vec{posicion}$, entre ella distintas transformaciones lineales como, por ejemplo:

Escalado Una matriz con la forma $\begin{pmatrix} K_1 & 0 & \cdots & 0 \\ 0 & K_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ 0 & 0 & \cdots & K_n \end{pmatrix}$ escala cada dimensión por el correspondiente valor de K.

Rotación Dependiendo del espacio en el que se trabaje, se pueden rotar los datos sobre distintos ejes. En un espacio bidimensional, se pueden rotar los datos un ángulo θ en dirección opuesta a las agujas del reloj con $M = \begin{pmatrix} cos\theta & -sen\theta \\ sen\theta & cos\theta \end{pmatrix}$ si se multiplica por la izquierda por un *vector-fila*; o $M = \begin{pmatrix} cos\theta & sen\theta \\ -sen\theta & cos\theta \end{pmatrix}$ si se multiplica por la derecha con un *vector-columna*.

Como ejemplo, para mejorar las posibilidades de éxito en la clasificación de un individuo en el dominio de ejemplo (figura 2.3), se puede realizar una rotación de 45º y después una compresión sobre el eje de ordenadas. Para que está compresión fuera a la cuarta parte se tendría:

$\theta = -45^o = -\pi/4$

$K_1 = 1/4$

$K_2 = 1$

$M_p = M_{rotacion} \times M_{escalado}$

$$M_p = \begin{pmatrix} cos(\pi/4) & -sen(\pi/4) \\ sen(\pi/4) & cos(\pi/4) \end{pmatrix} \times \begin{pmatrix} 1/4 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1/4cos(\pi/4) & -sen(\pi/4) \\ 1/4sen(\pi/4) & cos(\pi/4) \end{pmatrix}$$

Con lo que se obtendría la disposición de los datos mostrada en la figura 2.4

O, para comprimir en un sólo punto el eje x:

$\theta = -45^o = -\pi/4$

$K_1 = 0$

$K_2 = 1$

$$M_p = \begin{pmatrix} 0 & -sen(\pi/4) \\ 0 & cos(\pi/4) \end{pmatrix}$$

Y el resultado sería el de la figura 2.5, más simple de clasificar e incluso con un coste menor en el cálculo de la distancia si se llegara a tener en cuenta que ya sólo se necesita la distancia en el eje y.

Figura 2.4: **Ejemplo de transformación.** Dominio de ejemplo tras aplicarle una transformación consistente en un rotación y escalado.

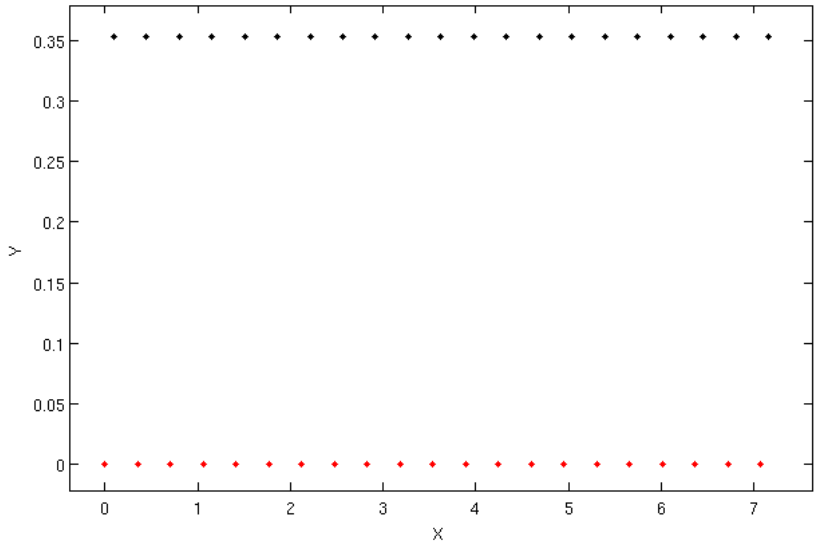
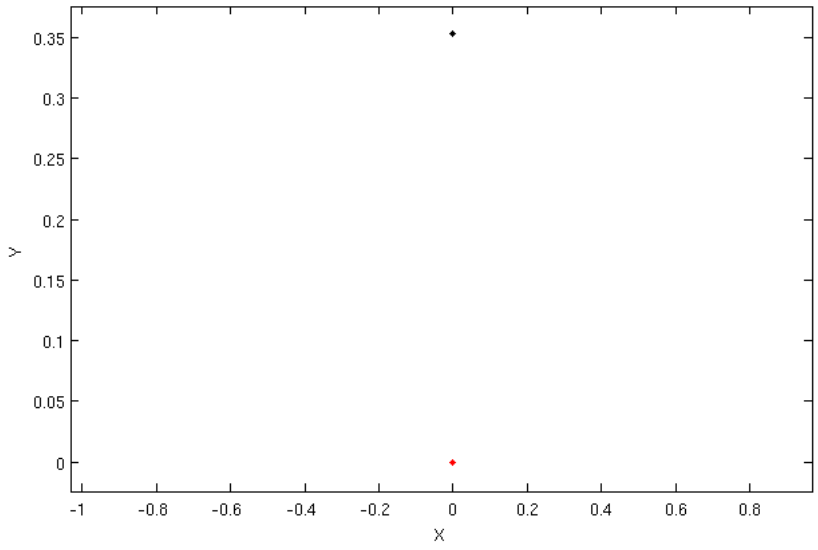


Figura 2.5: **Ejemplo de Transformación 2.**Dominio de ejemplo tras aplicarle una transformación consistente en un rotación y escalado que proyecta todos los puntos sobre el eje $y = 0$



2.2.3. Conclusión

La adaptación de la función de distancias es posiblemente el método más flexible en un principio, ya que permite por ejemplo el uso de distintas distancias para cada clase de forma más evidente que con las transformaciones, mientras que la transformación de los datos es más simple y menos costosa computacionalmente –los datos de ejemplo sólo se reposicionan una vez y más adelante cuando se quisiera utilizar el clasificador con nuevas instancias se aplicaría la transformación a cada nuevo dato, mientras que con la distancia Euclídea Generalizada hace más compleja la operación que más se repite, el cálculo de la distancia.

Cabe mencionar que los dos métodos no son exclusivos pero, por su similitud, combinar los dos no proporcionaría una mejora que compensase la sobrecarga de cómputo.

En cualquier caso, con ambos métodos se necesita una forma de obtener la matriz M que mejores resultados proporcione. Si no se dispone de información suficiente sobre el dominio, o si éste es muy complejo, se requiere alguna técnica que permita obtener dicha matriz de forma automática. Nos encontramos entonces con un problema de búsqueda dentro de un dominio específico (el de las matrices de dimensión igual al número de atributos del problema de clasificación). Una de las mejores técnicas para buscar una solución en un dominio del que no se dispone de suficiente información más allá de poder comparar una solución con otra es el uso de Estrategias Evolutivas.

2.3. Computación Evolutiva

El uso de técnicas computacionales basadas en la naturaleza se podría remontar a las neuronas artificiales en forma de células de Mc Culloch-Pitts [7] y posteriormente el perceptron [8], pero las primeras ideas sobre computación evolutiva se dieron en [9].

Conviene definir un vocabulario básico a la hora de hablar de conceptos de computación evolutiva, ya que existen una serie de términos que se utilizan continuamente en este campo:

Individuo Una solución al problema que se quiere resolver, ya sea como un

conjunto de datos, fragmento de código, etc.

Población Conjunto de varias soluciones.

Generación Población en un momento dado del proceso de evolución.

Cromosoma Codificación del individuo en un formato procesable computacionalmente.Una cadena de dígitos binarios sería un ejemplo muy común de cromosoma en este contexto, pero también existen otras como vectores de números reales (la más común en estrategias evolutivas).

Genotipo Dependiendo de la bibliografía, puede significar el cromosoma específico de un individuo o la forma de codificar el cromosoma.

Fenotipo Es el significado del cromosoma en el mundo real. Por ejemplo, una serie de dígitos binarios pueden representar una estatura, un valor de escala, etc.

Fitness También llamada adaptación, valoración que se hace de un individuo respecto a su utilidad para resolver el problema sobre el que se está trabajando.

Existe una gran cantidad de métodos que se podrían considerar inspirados a distintos niveles en la naturaleza y el concepto de supervivencia y evolución y combinándolos con más o menos rigor con conceptos matemáticos y de estadística. Se clasifican principalmente en los siguientes conjuntos:

Algoritmos Genéticos [11] Utilizan fundamentalmente en los conceptos de genotipo y fenotipo para codificar las propiedades de los individuos que se quieren evolucionar.

Estrategias Evolutivas [10] En este caso se trata de mejorar el resultado de una evaluación de un número real o un vector de números reales

Otros Técnicas como la Programación Genética[12], que trata de evolucionar un árbol de acciones y operadores en lugar de valores o el concepto de *Swarm Intelligence*[13] que trata de obtener resultados mediante el consenso de una gran cantidad de partes sencillas; no se consideran adecuadas para los fines de éste proyecto y por ello no se explican en detalle.

Los Algoritmos Genéticos y las Estrategias Evolutivas siguen unos pasos conceptualmente similares, aunque distintos en la práctica, para obtener sus objetivos. En ambos casos se comienza generando una población con soluciones aleatorias. Estas soluciones se evalúan y se seleccionan algunas para modificarlas, combinarlas y generar nuevas soluciones o individuos mediante los operadores de mutación y cruce, llamados operadores genéticos. Se continúa con este proceso hasta llegar a un punto de finalización, normalmente marcado por la calidad de la solución. El proceso completo paso a paso sería el siguiente:

1. Generación de la Población. Se generan individuos dentro del conjunto de soluciones posibles, bien aleatoriamente o bien mediante alguna heurística que obtenga individuos variados y con mejores evaluaciones que los aleatorios.

2. Valoración. Se evalúa cada individuo mediante una función de *fitness*.

3. Selección. Se utilizan varios métodos dependiendo de la implementación: se comparan individuos entre sí y se va eligiendo al mejor da cada grupo (*torneos*), se eligen aleatoriamente dando más posibilidades a los que tengan mejor *fitness* (*ruleta*); o simplemente se eligen los mejores. También se puede decidir entre tener en cuenta o no individuos de generaciones anteriores.

4. Mutación. Para aumentar la velocidad de la evolución, se introducen pequeñas modificaciones aleatorias con una baja probabilidad, de forma que se amplíen ligeramente las dimensiones de la búsqueda.P.ej. Invertir el valor de un dígito binario o sumar una pequeña cantidad aleatoria(positiva o negativa) a un número real.

5. Cruce. Se combinan individuos con buenos resultados para añadir el resultado a la población y así estudiar nuevas zonas del espacio de búsqueda. De nuevo dependiendo de la codificación del cromosoma existen distintas opciones, desde calcular una media si se trata de números reales, a mezclar fragmentos de los dos individuos si se trata de cadenas binarias (tomar la primera mitad de uno y la segunda mitad de otro, etc.).

6. Parada. Se pueden introducir diferentes condiciones de parada: Número de generaciones, un valor límite de *fitness*, un cambio mínimo entre

el mejor *fitness* de varias generaciones o un número de generaciones sin mejoras en el fitness, etc. Si esta condición no se cumple, se vuelve a 2.

Dos problemas típicos a los que hay que enfrentarse a la hora de construir una solución basada en una técnica evolutiva son la sobreadecuación a un conjunto de ejemplos y los posibles estancamientos de la evolución.

Sobreadecuación Se produce cuando, intentando conseguir los mejores valores posibles de *fitness*, los individuos consiguen resultados sustancialmente mejores para el conjunto de pruebas que se ha usado en la evolución que para cualquier otro dato. Es decir, a la hora de utilizar los resultados del evolutivo con datos no conocidos, su eficacia es mucho más baja que lo que cabría esperar a partir de los resultados de entrenamiento. Esta pérdida de generalización menoscaba la utilidad del evolutivo, y tiene varias posibles causas y soluciones.

Las causas principales son una selección de ejemplos de entrenamiento que no es representativa del dominio al que se quiere generalizar y un excesivo entrenamiento, con demasiados ciclos de evolución o generaciones.

Un buen método para evitar este problema es repartir el conjunto total de datos disponibles en subgrupos generados aleatoriamente y realizar pruebas con cada uno por separado, comparando resultados, utilizando algunos para el entrenamiento y otros para el control de resultados. Una forma de realizar esto es mediante validación cruzada, que se explica en la sección 4.4.

Estancamiento Se produce cuando no se pueden mejorar los resultados, en el mejor de los casos porque se ha llegado al límite de la técnica para el dominio tratado, pero también puede suceder por otros motivos evitables, como la convergencia prematura.

Si se da mucha importancia a al valor de evaluación para la selección de individuos, de forma que sólo se permita reproducirse a los mejores, puede darse el caso de que aparezca por puro azar un individuo que, sin ser extremadamente bueno en la visión general, sí sea notablemente superior al resto de la población para una generación dada. Este individuo, a pesar de ser estar lejos del óptimo, siendo posiblemente un mínimo local de la función de *fitness*, bajo estas condiciones sería

elegido por encima de todos los demás y podría acabarse con una generación formada por individuos iguales. Esta pérdida de diversidad genética no podría solucionarse sólo con la mutación y se acabaría en un estancamiento.

La solución de este problema es compleja y se han diseñado diversos métodos para resolverlo o paliarlo, como introducir estocástica en la selección, de forma que se siga dando preferencia a los mejores pero existan probabilidades de elegir a los demás también, o mantener de forma explícita la diversidad genética, sustituyendo no a los peores individuos sino a los más parecidos a los nuevos en cada generación.

2.3.1. CMA-ES

Esta técnica [14, 15, 16][1], cuyo nombre proviene de *Covariance Matrix Adaptation Evolution Strategy*, se basa en generar los individuos a partir de una distribución normal multidimensional. Para esto se forman generaciones de λ individuos, generados aleatoriamente a partir de la distribución, y se eligen los μ mejores. Con ellos y la información que se extrae de las anteriores generaciones se forma una nueva de la siguiente manera:

- La media de la distribución normal multidimensional se obtiene como media ponderada de los valores de los individuos, dando más peso a aquellos con los que se obtenga un mejor valor de *fitness*.
- La construcción del segundo parámetro que define a una distribución normal multidimensional, la matriz de covarianzas, es en el que más cuidado se ha puesto en CMA-ES, de ahí su nombre. Se basa en tres ideas principalmente:
  - *Rank-μ update*. No sólo se calcula la covarianza de los μ mejores individuos como en el caso de la media, sino que, una vez calculada ésta, se calcula una media ponderada con todas las anteriores, dando mucha importancia a las últimas generaciones y casi ninguna a las primeras.
  - *Rank-one update*. Se introduce la idea del *paso (step)* entre generaciones, como la dirección que siguen las modificaciones de los

[1] Una guía introductoria completa puede encontrarse en `http://www.lri.fr/~hansen/cmaesintro.html`, junto con referencias a los artículos en los que se han definido y refinado las distintas partes de CMA-ES.

valores de la matriz entre generaciones, de forma que se favorezca el repetir pasos que hayan proporcionado buenos resultados.

- *Step-size control.* El objetivo principal de las dos ideas anteriores es reducir el tamaño de la población que se necesita para formar una Matriz de covarianzas que se pueda considerar fiable a la hora de generar buenos individuos en la siguiente generación. El control del tamaño se introduce, por otro lado, para reducir en lo posible el número de generaciones necesarias antes de llegar a resultados satisfactorios, mientras se evita caer en mínimos locales. La idea es controlar el tamaño de los pasos comparando su tamaño con el que tendrían si se eligiera un camino *aleatorio* (es decir, en el que no existiera correlación alguna). Si el camino es más corto, es que los pasos son demasiado largos y se producen ciclos, si el camino es largo, se están dando varios pasos en la misma dirección, cuyo número podría reducirse si fueran más largos.

2.3.2. Evolución Multiobjetivo : NSGA-II

Una de las partes más importantes de una técnica evolutiva es la función de validación o *fitness*, ya que es la que permite comparar unos individuos con otros y decidir cuál está más cerca del objetivo que se busca. Sin embargo, en la mayoría de los casos se manejan entornos con múltiples parámetros de entrada y salida a optimizar. Cuando no se tiene un objetivo único a optimizar, o cuando se quiere encontrar un balance entre varias metas que pueden afectar negativamente unas a otras, el patrón básico de las técnicas evolutivas no es suficiente y se deben construir soluciones que tengan todo esto en cuenta. Hablamos entonces de técnicas multiobjetivo.

Un ejemplo de dominio con múltiples objetivos sería la bolsa de valores. Una inversión en acciones tiene dos objetivos contrapuestos: maximimizar el beneficio y minimizar el riesgo. Encontrar una cartera de acciones óptima incluiría balancear estos dos objetivos.

Existen diversas aproximaciones a la hora de resolver problemas multiobjetivo, tanto dentro como fuera de las técnicas evolutivas. Principalmente se pueden dividir en dos grupos, las que construyen una única función objetivo a partir de las demás (AFO, *Aggregate Objective Function*) y las que se basan en explorar el espacio multidimensional que forman los espacios

de soluciones de cada objetivo conjuntamente. Por ejemplo, en el consabido problema de la mochila se podrían tener dos objetivos: minimizar el peso total de los objetos que se llevan maximizando la utilidad total; la función agregada podría ser tan simple como, dado un valor numérico la utilidad, minimizar el peso total dividido entre la utilidad total, escalando estos totales de forma adecuada a su importancia, explorando después las soluciones dentro de unos umbrales (utilidad mínima y peso máximo) adecuados.

Al conjunto de los puntos en los que no se puede mejorar un objetivo sin perjudicar a otro se le denomina Frente de Pareto(figura 2.6). Un buena técnica de optimización multiobjetivo sería aquella que permitiera explorar este frente y elegir la solución que mejor se ajustase a las necesidades en cada caso.

Los métodos que construyen una única función objetivo son en teoría más eficientes, pero requieren una mayor adaptación del método al problema que los que se basan en aproximar y explorar un frente de Pareto. Existen también métodos que combinan ambas ideas, tratando de explorar el frente a partir de una función objetivo combinada[17].

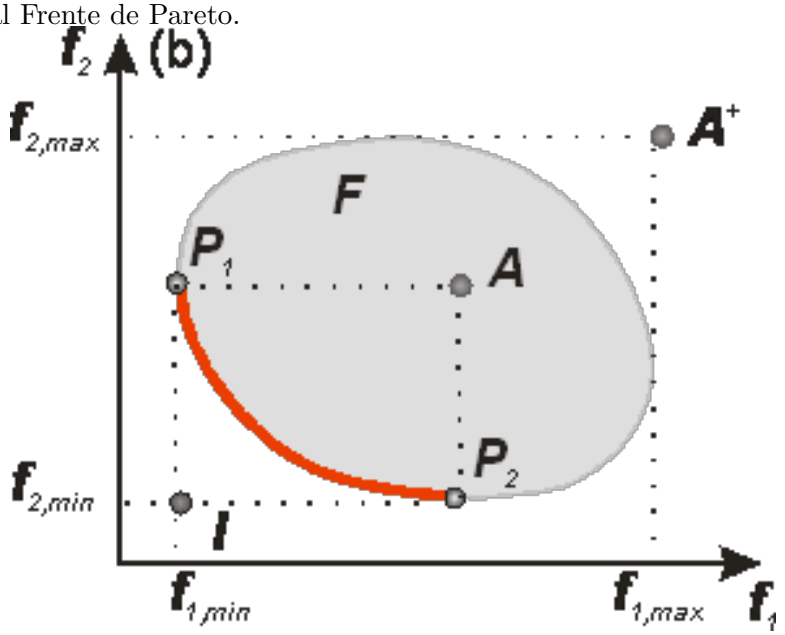
NSGA-II, definido en [18], es un algoritmo genético multiobjetivo elitista, que se basa principalmente en el concepto de dominación entre soluciones. Para que una solución x_1 domine a otra x_2 se debe cumplir:

1. x_1 no tiene un valor de *fitness* peor para ninguno de los objetivos definidos.

2. Existe al menos un objetivo para el que x_1 obtiene un mejor valor de fitness que x_2

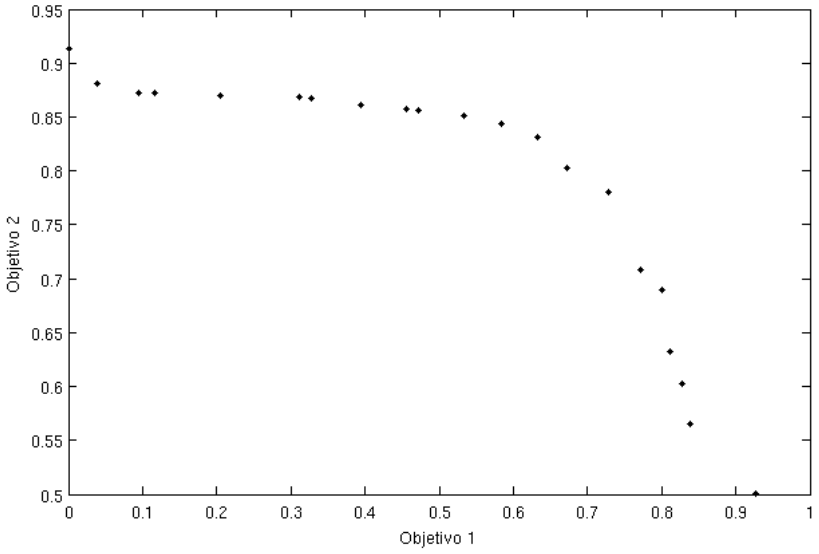
NSGA-II utiliza esta idea de dominación para organizar a los individuos en cada iteración del algoritmo genético, de forma que en cada generación se obtienen los descendientes por cruce, se ordena toda la población por dominación y los mejores sobreviven para la siguiente generación. Cuando no se puede seguir mejorando o se activa algún otro mecanismo de parada, se puede explorar la población resultante(fig. 2.7) en busca del individuo que mejor satisfaga los objetivos.

Figura 2.6: **Ejemplo de Frente de Pareto.** Si el área gris son las posibles soluciones y se trata de minimizar dos objetivos, la curva en rojo correspondería al Frente de Pareto.



Fuente: `http://www.mlahanas.de/MOEA/MO_Optimisation.htm`

Figura 2.7: **Ejemplo de frente al que llega una ejecución de NSGA-II.**



2.4. Opciones de Implementación

Para el desarrollo existen tres opciones básicas :

1. Sistema ya desarrollado.

2. Unir partes ya desarrolladas.

3. Desarrollo completo.

No existe un sistema que implemente las ideas que se quieren estudiar, aunque sí que existen implementaciones de los diferentes algoritmos que se han expuesto, por lo que se pueden diseñar soluciones basadas en la segunda y tercera opción, se han estudiado las que se consideraron más ventajosas por haberse utilizado previamente o ser similares a sistemas conocidas o ampliamente utilizados.

2.4.1. Entornos Matemáticos

Existen numerosos sistemas diseñados para facilitar todo tipo de trabajos basados en ramas más avanzadas del álgebra, cálculo, estadística... que las que utilizan normalmente los lenguajes de programación más populares.

MATLAB

MATLAB[2] (*MATrix LABoratory*) es un sistema ampliamente utilizado destinado principalmente a los cálculos algebráicos complejos pero extendido a muchos otros dominios mediante el uso de ampliaciones (*Toolboxes*). Sus principales características son:

- Sistema privativo. MATLAB es un sistema propiedad de Mathworks, con lo que un sistema desarrollado en este entorno requerirá seguir utilizándolo si se quiere extender en un futuro.
- Coste. Una licencia de MATLAB para uso académico (p.ej. para un grupo de investigación) cuesta aproximadamente 500€ a fecha de junio de 2010. A esto habría que sumar 200€ por cada *Toolbox*. Las licencias de estudiante, por otra parte, cuestan aproximadamente 75€ por el programa y 25€ por cada *Toolbox*.
- Como ya se ha mencionado, se dispone de una gran cantidad de módulos con funciones ya implementadas, entre ellas algunas de las que se quieren utilizar, como clasificadores como K-Vecinos entre otros, algoritmos genéticos, sistemas para automatizar la selección de individuos para una validación cruzada y otros muchos.
- Bajo rendimiento, o, más específicamente, arquitectura orientada a cierto tipo de operaciones que no suelen ser las típicas en programación. Por ejemplo, las operaciones con matrices están implementadas de forma que sean muy eficientes, pero a cambio otras operaciones como los bucles son notablemente lentas.
- Paralelización sencilla. MATLAB permite la paralelización transparente de la mayor parte la ejecución, la paralelización explícita requiere de poco más esfuerzo (si el diseño ya se ha realizado teniendo en cuenta la paralelización) que la definición de bucles con la instrucción *parfor*.

[2] `http://www.mathworks.com/`

- Aunque para el desarrollo se requiera del entorno MATLAB, una vez terminado se pueden compilar prototipos que ejecuten en otros ordenadores mediante bibliotecas de Java.

Octave

Octave[3] es una implementación libre de MATLAB, por lo que resulta muy parecido pero con algunas ventajas e inconvenientes respecto a éste:

- Es gratuito.
- Aunque cuenta con algunas extensiones y en teoría es completamente compatible con MATLAB, no cuenta con una biblioteca tan amplia de extensiones, por lo que para las mismas tareas hay que buscar implementaciones de terceros con peor soporte y documentación o desarrollarlas completas.
- Al ser libre y gratuito, resulta más fácil su distribución en cualquier ordenador y modificación y extensión de acuerdo a las necesidades del usuario.
- Aunque cuenta con bibliotecas de paralelización y distribución, éstas no son tan extensas y de uso tan simple como las de MATLAB, por lo que requieren de nuevo el uso de aplicaciones de terceros[4] y una mayor esfuerzo dedicado a la paralelización en el caso de que se quiera utilizar.

2.4.2. Desarrollo completo

En lugar de contar con herramientas ya existentes, se puede desarrollar un sistema completo que implemente las técnicas que se quieren estudiar. Esto por un lado aumenta enormemente las flexibilidad y las posibilidades de diseño, pero también incrementa el trabajo necesario, desviando hacia el desarrollo y la programación el trabajo originalmente orientado hacia la investigación de la combinación de una serie de algoritmos y heurísticas. Esto es un problema fundamentalmente porque se pasaría a repetir un trabajo

[3] `http://www.gnu.org/software/octave/`
[4] `http://www.shogun-toolbox.org/`

de programación de una serie de algoritmos que ya han sido ampliamente probados y utilizados en lugar de buscar formas de mejorarlos.

En realidad, existen implementaciones tanto de los algoritmos de clasificación como de los evolutivos en multitud de lenguajes de programación disponibles para su reutilización, pero al no formar parte de un sistema integrado debería realizarse el estudio de la interfaz de cada bloque por separado, encontrar los que mejor se pudieran combinar y unificarlos.

Por todo esto, aunque el coste de las herramientas de desarrollo podría ser nulo dependiendo de las elegidas, el coste de implementación podría no compensar la diferencia, por lo que finalmente se ha optado por realizar un sistema basado en MATLAB.

Capítulo 3

Objetivos

Dados un algoritmo de clasificación y una serie de técnicas evolutivas, se pretende encontrar un método que mejore la capacidad de clasificación del algoritmo bajo los siguientes criterios y restricciones:

- Eficacia: Se quiere conseguir mejorar los resultados de clasificación de la versión básica del algoritmo de clasificación.
- Generalidad: Se introducen una serie de dominios en los que el algoritmo de clasificación en su forma más básica consigue muy distintos niveles de éxito en la clasificación. Se pretende mantener los resultados buenos y mejorar los demás tanto como se pueda.
- Eficiencia y rendimiento: Teniendo en cuenta las altas necesidades de recursos (en cuanto a potencia de hardware y en tiempo) que en ocasiones se asocian a las técnicas evolutivas, se establecerán límites en cuanto a los recursos necesarios y se trabajará dentro de éstos.

Para esto, se quiere desarrollar un sistema que implemente estos conceptos y objetivos y realizar las pruebas necesarias para comprobar su cumplimiento.

Capítulo 4

Desarrollo

4.1. Objetivos

El objetivo principal del programa desarrollado es la ejecución de las pruebas, con lo cual se requieren las siguientes características:

1. Modularidad: Se deben poder cambiar las partes principales del programa para poder probar y comparar distintos algoritmos.
2. Personalización: Se deben poder modificar fácilmente los parámetros que afectan a los algoritmos.
3. Obtención de resultados: Se necesita obtener y almacenar información en detalle sobre las pruebas realizadas para poder analizarla después.
4. Formato de los resultados: Los resultados deben obtenerse en un formato fácil de comprender y de procesar en el caso de que necesiten añadirse a algún documento.
5. Rendimiento: En la medida de lo posible, se debe tener en cuenta el tiempo que se tardará en realizar las pruebas, minimizándolo en la medida de lo posible y ofreciendo métodos para limitarlo.
6. Acceso externo: Aún mejorando el rendimiento, las pruebas tendrán que ejecutarse durante horas, por lo que sería conveniente facilitar el acceso remoto al entorno de pruebas para comprobar resultados, lanzar nuevas pruebas, etc. en cualquier momento.

4.2. Diseño

Cada uno de los requisitos expuestos se cumplimentará de la siguiente forma:

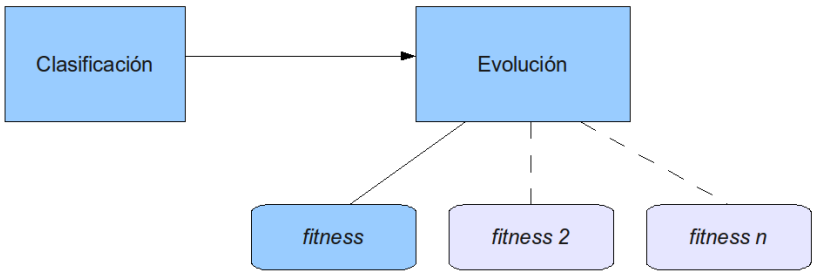
1. Modularidad: El programa se dividirá en cuatro partes diferenciadas principalmente:

   - Ejecución de las pruebas en validación cruzada[19]. Se permitirá elegir un número de *folds* en los que dividir las instancias, de forma que una se utilice para comprobar la calidad final del clasificador y el resto para el entrenamiento con la técnica evolutiva. Además se podrá elegir el número de ejecuciones del proceso, de forma que al final se calcule una media de todas las ejecuciones con distintos grupos de entrenamiento y test elegidos. Ver punto 4.4.
   - Evolución de la matriz de transformación.
   - Cálculo del *fitness*. Se aplica la transformación a los datos y se realiza una clasificación guardando el ratio de error como valor de *fitness*. De nuevo ver el punto 4.4 para más detalles sobre cómo se dividen los datos en test y entrenamiento.
   - Algoritmo de clasificación.

   En la figura 4.1 se pueden ver los módulos fundamentales, ya que la ejecución de la validación cruzada conllevaría poco más que la elección de los conjuntos de entrenamiento y prueba y la ejecución de un bucle. Como para parte de las pruebas se van a utilizar algoritmos multiobjetivo, se muestra la posibilidad de utilizar varias funciones de *fitness* a la vez.

2. Personalización: Existen varias formas de conseguir esto, como utilizar un fichero de opciones o incluso diseñar un interfaz gráfica en la que elegirlas, pero, dada la sencillez que en general ofrece MATLAB, y teniendo en cuenta que no se busca desarrollar una aplicación para su uso a largo plazo sino probar una serie de algoritmos, se van a parametrizar las opciones principales en la llamada a la ejecución del programa y el resto se dejarán al principio del código para poder modificarse fácilmente.

Figura 4.1: **Módulos de los que consta el programa.**



3. Obtención de resultados: Los resultados de las pruebas se mostraran por la pantalla principalmente para conocer el estado del programa y se almacenaran en ficheros con más detalle para su posterior estudio.

4. Formato de los resultados:Los ficheros de resultados consistirán en texto plano, fácil de manipular. Se separará el contenido en varios ficheros para los resultados de entrenamiento, resultados de *test*, estadísticas de éstos y *log* de la ejecución de la aplicación.

5. Rendimiento: Teóricamente, la mayor parte del tiempo se consumirá en la ejecución de las llamadas a la función de *fitness*, más teniendo en cuenta que cada una de ellas deberá realizar la clasificación sobre un grupo de datos.Se debe por tanto limitar o bien el número de cálculos del *fitness* o el tiempo durante el cual se puede realizar la evolución.

6. Acceso externo: Si se utiliza un entorno de pruebas basado en un sistema Unix como GNU/Linux, Mac OS X o FreeBSD, se pueden utilizar varias de las herramientas con las que cuentan este tipo de sistemas para el acceso remoto con requerimientos bajos en cuanto a velocidad de conexión y manteniendo las pruebas en ejecución cuando no se está conectado a la máquina de pruebas, sin por ello perjudicar el que se pudiera utilizar el resto de la aplicación en cualquier entorno compatible con MATLAB.

4.3. Implementación

Con mayor nivel de detalle y exponiendo ya elecciones específicas de implementación se detallan de nuevo los puntos anteriores:

1. Modularidad: MATLAB cuenta con implementaciones de diversos algoritmos de clasificación, entre los que se incluye K-Vecinos, y algoritmos genéticos, entre ellos una implementación de NSGA-II como algoritmo genético multiobjetivo. Además uno de los autores de CMA-ES proporciona en su página web implementaciones de su algoritmo en varios lenguajes, incluido MATLAB[1]. Se dispone por todo esto de una buena parte de las partes que constituirán el programa y además se encuentran ya divididas de forma que sea fácil añadirlas o sustituirlas por otras.

2. Personalización: La llamada básica a la ejecución principal tendrá la siguiente forma:

   $clasificar(entrada, salida, metodo, tipoMatriz, nEjecuciones, nFolds)$

   Permitiendo asignar valores a los parámetros siguientes:

   - *entrada* Archivo con los valores de ejemplo del dominio de clasificación.
   - *salida* Directorio en el que se deben guardar los resultados.
   - *metodo* Cómo se va a calcular la transformación.
   - $tipoMatriz$ Diagonal o Completa.
   - $nEjecuciones$ Ejecuciones externas completas de la validación cruzada para obtener una media de los resultados.
   - $nFolds$ Número de *folds* en validación cruzada.

   Además, tras una breve documentación, en el código MATLAB se pueden encontrar las asignaciones a los valores por defecto de otros parámetros, de forma que sean fáciles de modificar, entre otros se incluyen los siguientes: desviación para la generación de valores aleatorios, número máximo de llamadas a la función de *fitness* dentro de una ejecución del algoritmo evolutivo, valor de K para K-Vecinos.

3. Obtención de resultados: En la ruta introducida en la entrada se guardarán tres ficheros:

[1] `http://www.lri.fr/~hansen/cmaes_inmatlab.html`

- *Train.txt* : Resultados numerados de cada clasificación para cada conjunto de entrenamiento en cada ejecución externa.
- *Res.txt* : Resultados de clasificación para cada conjunto de test en cada ejecución externa y media final.Se incluirá también el resultado de realizar la misma comparación sin realizar transformaciones y el número de ceros en la matriz de transformación. Por último, en la última línea se incluirán las medias y desviaciones típicas de los resultados.
- *Log.txt* : Además de los mismos resultados incluidos en *Res.txt*, se incluirá más información sobre la ejecución: fecha, hora y dominio, número de ejecuciones externas y *folds*, matrices de transformación obtenidas para cada *fold* y tiempo total de ejecución.

Por pantalla se mostrará la misma información que en el archivo *Log.txt*. En el caso de utilizar evolución multiobjetivo, se guardará un archivo *Frentes.txt* con los frentes y sus *fitness* resultantes para cada ejecución.

4. Formato de los resultados:

   - *Train.txt*: 2 columnas:
     - Número de *fold* en la ejecución.
     - Ratio de error en la clasificación.
   - *Res.txt*:Un separador rodeado de "#" para cada ejecución y 4 columnas:
     - Número de *fold* en la ejecución.
     - Ratio de éxito sin transformar.
     - Ratio de éxito con transformaciones.
     - Cantidad de ceros en la matriz.

     Una última línea con las medias y desviaciones típicas, estas últimas entre paréntesis, de: éxito sin transformaciones, éxito con ellas y número de ceros.
   - *Log.txt*: Un primer bloque con la hora, fecha, fichero origen, método y número de ejecuciones externas y *folds*. Después un bloque para cada fold que incluya su número, matriz de transformación obtenida y resultados de clasificación (siguiendo el mismo patrón que en *Res.txt*: sin transformación, con ellas y número de ceros). Terminar con el tiempo total que ha tardado la prueba completa.

5. Rendimiento: El código disponible para CMA-ES permite poner una cota al número de ejecuciones y la implementación de NSGA-II de MATLAB permite limitar el tiempo que se utiliza en la evolución. Con esto es suficiente para controlar la duración de la ejecución de la aplicación. Si no se indica lo contrario, los experimentos se realizaron limitando el número de evaluaciones de la función de *fitness* a 800 o el tiempo de ejecución a 120 segundos en el caso de NSGA-II.

6. Acceso externo: Se utilizará la versión sin interfaz gráfico de MATLAB, por línea de comandos, ejecutando sobre un ordenador con Linux con acceso continuo a Internet e IP fija. De esta forma se podrá acceder en cualquier momento al entorno mediante una conexión SSH [2] y con la herramienta GNU Screen [3] se podrá acceder a la misma sesión en ejecución desde cualquier ordenador con conexión a Internet y cliente SSH.

4.4. Validación Cruzada: Necesidad y Método

Cuando se trabaja sobre conceptos que tratan de predecir y generalizar soluciones, como son los algoritmos de clasificación y las técnicas evolutivas, es necesario poner a prueba de forma exhaustiva las ideas desarrolladas para evitar llegar a conclusiones equivocadas producto de la casualidad o de un conjunto de pruebas mal formado.

La validación cruzada[19] consiste en dividir todos el conjunto de ejemplos del que se dispone para las pruebas en varios subconjuntos, conocidos a veces por su nombre en inglés, *folds*, que se utilizarán para las tareas de entrenamiento y validación de la solución. Además, los mismos grupos se van rotando en estas tareas y los resultados que se tienen en cuenta son las medias de todas las pruebas.

Uno de los métodos más comunes para realizar una validación cruzada es *K-folds*, que consiste en dividir los ejemplos en K grupos, utilizándose uno para el test (validación) y el resto para el entrenamiento. Se repite la prueba K veces, utilizando cada vez uno de los grupos como test, y se guardan los resultados de cada uno para utilizar sus estadísticas (media, varianza, etc.)

[2] `http://www.openssh.com/`
[3] `http://www.gnu.org/software/screen`

como medidas de la eficacia del método probado. Aplicándolo en concreto a las técnicas que conciernen a este proyecto:

K-folds en K-Vecinos Los $K-1$ conjuntos de entrenamientos serían todos los ejemplos con los que comparar las nuevas instancias para su clasificación, y el conjunto de test serían los nuevos individuos a clasificar. El valor de validación sería el número de individuos del conjunto de test correctamente clasificados (preferiblemente el ratio sobre el tamaño del conjunto de test). Como ya se ha dicho se repetiría el proceso K veces, rotando el conjunto que haría de test.

K-folds en Técnicas Evolutivas En este caso los conjuntos de entrenamiento se utilizarían para calcular el valor de *fitness* de cada individuo del algoritmo evolutivo, intentando en el proceso de evolución obtener los mejores resultados posibles para estos conjuntos. Después, ya con el proceso evolutivo completado y un individuo seleccionado, se calcularía de nuevo el valor de *fitness* para los datos del conjunto de test, obteniendo así una medida del funcionamiento sobre datos fuera de los usados en la evolución. Comparar los resultados obtenidos en entrenamiento y test permite detectar problemas de sobreadecuación a uno datos determinados (ver sección 2.3).

En este trabajo se da el caso de que la evaluación del *fitness* dentro del algoritmo evolutivo requiere de la evaluación de un clasificador, por lo tanto se requiere una validación cruzada "doble", que se ha diseñado de la siguiente forma:

- Validación cruzada del evolutivo: Como se indica en la sección 4.2, se permite elegir el número de *folds* que se utilizarán en la validación cruzada, y además, se pueden realizar varias ejecuciones completas con conjuntos generados de nuevo repartiendo los individuos de forma aleatoria, aumentando aún más la diversidad de las pruebas. Se dividen entonces los datos disponibles en los K *folds* que se indiquen, $K-1$ se utilizan para evaluar el *fitness* durante la evolución de la siguiente forma:

- Validación del clasificador en *fitness*: Cuando se vaya a evaluar una matriz de transformación el proceso será el siguiente:

  - Dividir el conjunto de datos del que se disponga (que serían los $K-1$ *folds* indicados anteriormente) en 5 *folds*, 4 de entrenamiento y 1 de test. Para clarificar se llamarán *entrenamiento-f* y *test-f*, y a los primeros *entrenamiento-global* y *test-global*.
  - Clasificar los datos del conjunto de *test-f* utilizando los datos de *entrenamiento-f* como ejemplos (vecinos).
  - El ratio de error que se obtenga de esta clasificación será el valor de fitness a minimizar.

- Validación final: Una vez obtenida la matriz de transformación final, ésta se utilizará para transformar todos los datos del dominio, y se clasificarán los datos transformados de test-global, utilizando entrenamiento-global como ejemplos.

- El ratio de éxito en clasificación en porcentaje será el dato que se guarde como valoración del método.

- Se elige un un nuevo *fold* como test-global y se repite el proceso hasta que se hayan usado los K *folds*.

- Si hay que realizar más ejecuciones externas, se vuelven a construir los K grupos eligiendo aleatoriamente las instancias que los formarán y se repite todo el proceso.

- La media y desviación típica de estos ratios de éxito en todas estas pruebas será el resultado utilizado para valorar todo el método para un dominio.

Capítulo 5

Experimentación

5.1. Primeros Experimentos

Las primeras pruebas, basadas en [20], se realizaron fundamentalmente para comprobar tanto la efectividad de las técnicas evolutivas en la mejora del algoritmo de clasificación como el funcionamiento básico de CMA-ES.

Para esto se enfrentaron diferentes técnicas de clasificación con K-Vecinos:

- Sin transformación de los datos. Equivalente a utilizar la distancia Euclídea.
- Transformando los datos. Se multiplicaron los datos por una matriz de transformación siguiendo el método explicado en el punto 2.2.2 del contexto. Con esto se trataba de conseguir, utilizando transformaciones lineales, un mejor posicionamiento de los datos respecto a la capacidad de clasificar los individuos del dominio mediante K-Vecinos.
    - Búsqueda local - matriz diagonal. Permite el escalado de los valores de los datos pero no operaciones más complejas.
    - Búsqueda local - completa, cualquier elemento puede ser distinto de cero. Permite operaciones que dependen de más de una dimensión a cambio de aumentar la complejidad y el tamaño del espacio de búsqueda.
    - CMA-ES - sólo matrices diagonales.

- CMA-ES - matrices completas.

Para las pruebas se utilizaron los siguientes dominios:

- *Aleatorio*: Las instancias tienen 4 atributos generados aleatoriamente, 2 con valores entre 0 y 1 otros 2 con valores entre 0 y 100. La forma de repartir las clases es: si el primer atributo es mayor al segundo, la instancia pertenece a la clase 1 y a la 0 en caso contrario. Los otros atributos, con valores mayores, se introdujeron para que la clasificación con *K-vecinos* no obtuviera buenos resultados utilizando una distancia euclídea.
- *Iris*: Dominio real sobre plantas Iris extraído del repositorio UCI[1]. Contiene 150 instancias con 4 atributos pertenecientes a 3 clases.
- *Rectas*45: Figura 5.1. Es un dominio artificial diseñado explícitamente para que la clasificación de sus instancias requiera transformar éstas con el escalado y rotación de sus valores. Consiste en 100 puntos de una clase colocados con una separación de una unidad formando una línea con un ángulo de 45º respecto a la horizontal, empezando en el punto (0,0). Las instancias de la segunda clase siguen la misma distribución pero empezando en el punto (0,-1). Cada coordenada de cada punto tiene aplicada una perturbación aleatoria con valores entre $-0{,}5$ y $-0{,}5$. Por lo tanto, para que las instancias se clasifiquen correctamente se deben rotar y escalar horizontalmente.
- *Ripley*: 1250 instancias con 2 atributos formados mediante 4 distribuciones *gaussianas* [22].
- *Wine*: Dominio real extraído del repositorio UCI. Se trata del análisis de la calidad de distintos vinos, conteniendo 178 instancias con 13 atributos pertenecientes a 3 clases.

Se realizó una prueba (Tabla 5.1) para cada dominio utilizando 10 ejecuciones externas de 10 *folds* cada una, permitiendo que cada algoritmo llamara hasta 800 veces a las función de *fitness* para cada ejecución.

Un ejemplo de las matrices de transformación que obtiene el método es ésta, obtenida para el dominio rectas45: $\begin{pmatrix} 0{,}7731 & 7{,}6867 \\ -0{,}9702 & -7{,}3557 \end{pmatrix}$, cuyo efecto

[1] `http://archive.ics.uci.edu/ml`

Figura 5.1: **Dominio Rectas Rotadas.**

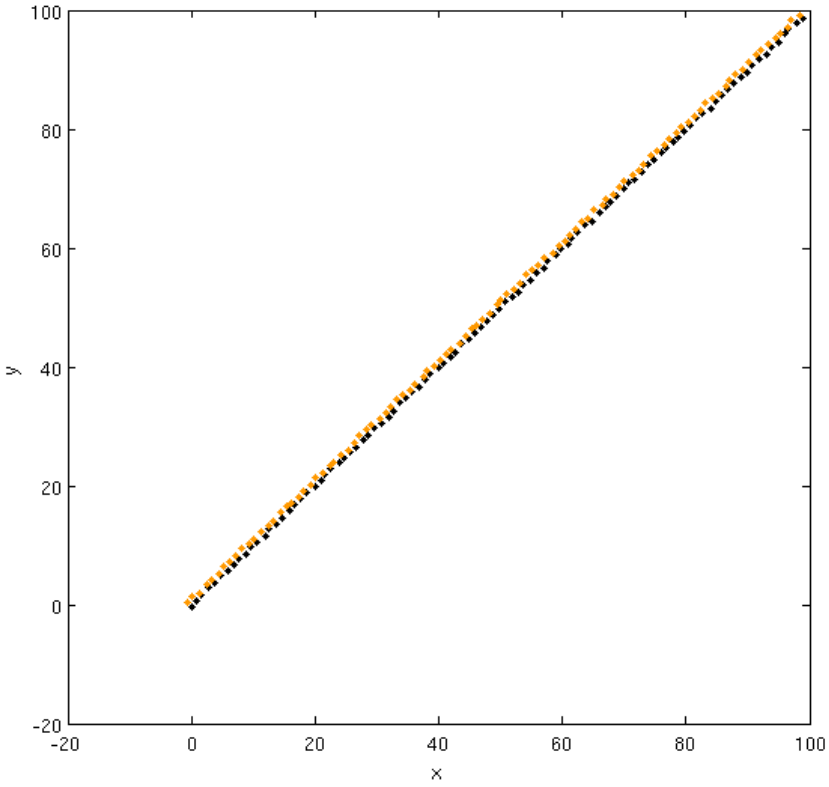


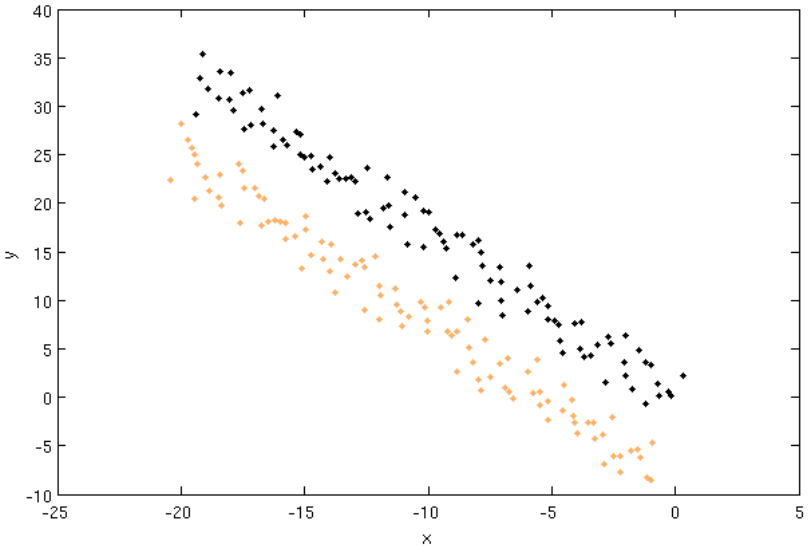
Tabla 5.1: **Resultados Iniciales.** Pruebas para todos los dominios: 1) Sin transformación 2) Búsqueda local de una matriz diagonal 3) Búsqueda con CMA-ES de una matriz diagonal 4) Búsqueda local de una matriz con valores en todas las posiciones 5) ídem para CMA-ES.

Dominio	Euclidea	LS-diag	CMA-diag	LS	CMA-comp
Aleatorio	50.96 (08.19)	94.10 (09.35)	**95.70 (05.90)**	55.50 (13.84)	62.23 (14.78)
Iris	**95.86 (04.42)**	94.66 (06.35)	94.26 (05.44)	95.13 (05.43)	95.26 (04.85
Rectas45	08.30 (08.32)	09.00 (07.62)	08.85 (07.81)	**98.70 (02.62)**	**98.70 (02.52)**
Ripley	88.72 (02.52)	**88.88 (02.79)**	88.76 (02.78)	88.54 (02.33)	88.77 (02.63)
Wine	76.48 (09.44)	94.15 (05.67)	**94.16 (04.95)**	74.35 (09.51)	80.86 (09.70)

sobre el dominio es el que se puede ver en la figura 5.2. Es una de las muchas posibles transformaciones que consigue acercar los puntos de la misma clase

entre sí a la vez que distanciarlos de los de la otra clase, con lo que se mejora como se deseaba la clasificación con *K-vecinos*.

Figura 5.2: **Dominio Rectas Rotadas tras la transformación.** Ejemplo del efecto de una transformación obtenida mediante evolución sobre el dominio.



Se puede observar que en la mayoría de los casos los resultados son mejores para CMA-ES. Sin embargo, la opción más flexible, la que evoluciona una matriz de transformación completa, obtiene en un número significativo de ocasiones peores resultados que la matriz diagonal o incluso que la búsqueda local. La explicación para esto es que el espacio de búsqueda para la matriz completa es mucho mayor que el de la diagonal (el número de elementos al cuadrado) por lo que la evolución es más lenta, además de existir más posibilidades de estancamiento por mínimos locales. Por todo esto, cuando la solución óptima se puede representar con una matriz diagonal, se encontrará mucho más fácilmente si se busca directamente entre las matrices diagonales que si se busca entre todas las posibles, aunque éstas comprendan a las diagonales.

Aún con todo esto, hay que tener en cuenta que las opciones que se utilizaron para CMA-ES fueron las elegidas por defecto, por lo que se podrían esperar mejoras ajustando los parámetros al problema.

5.2. Optimizando los parámetros de CMA-ES

Aunque la documentación de CMA-ES indica que las opciones por defecto son en general buenas para cualquier problema, el algoritmo proporciona una cantidad generosa de parámetros si se quiere ajustar lo máximo posible el algoritmo al problema al que se esté aplicando.

- *Restarts*: Inicia de nuevo el algoritmo al alcanzar alguna de las condiciones de parada, aumentando cada vez el tamaño de la población según un parámetro. Se reinicia hasta llegar al límite de reinicios o hasta alcanzar una condición definitiva de parada (v.g. número máximo de evaluaciones de la función de *fitness*.
- *StopOnStagnation*: Parada si tras un gran número de generaciones no se producen cambios significativos en el valor de *fitness*. La decisión está implementada internamente en CMA-ES y no admite parámetros.
- *DiagonalOnly*: Utiliza una matriz de covarianzas diagonal durante el número de generaciones que se le indique. Se recomienda su uso para dominios con un número elevado de dimensiones (cercano o superior a 100).
- *StopFitness*: Permite indicar un valor de *fitness* de parada para evitar más iteraciones de las necesarias buscando por ejemplo un valor negativo cuando el mínimo posible fuera el cero.
- *LBound, UBound*: Cotas, inferior y superior respectivamente, para los valores que puede tomar la variable buscada, fuera de estos valores se establece una penalización a la búsqueda.
- *DiffMinChange, DiffMaxChange*: Cambios mínimo y máximo que se pueden realizar en un valor de una generación a otra.

Se realizaron una serie de pruebas (Tabla 5.2) modificando estos parámetros para comprobar si se obtenía alguna mejora en los dominios problemáticos, es decir, en los que el CMA-ES diagonal o la búsqueda local obtenían mejores resultados que el CMA-ES completo. Asimismo, se mantuvo el dominio Rectas45 en el que CMA-ES obtenía los mejores resultados para comprobar si las mejoras en unos dominios perjudicaban a otros.

Analizando cada dominio por separado:

Tabla 5.2: **Resultados de Optimización de CMA-ES.** Pruebas para todos los dominios con: 1) 1000 evaluaciones, permitiendo volver a empezar aumentando la población si no hay mejoras durante mucho tiempo. 2) 10000 evaluaciones. 3) Sólo valores positivos. 4) No cambiar el valor de un atributo en la media que utiliza CMA-ES si el cambio va a ser menor a 0.0005.

Dominio	Reinicios (1000 ev.)	10000 evaluaciones	Cota inferior 0	Cambio mínimo 0.0005
Aleatorio	60.96 (15.39)	**88.86 (12.26)**	70.90 (24.69)	68.50 (16.31)
Iris	95.66 (04.86)	**95.93 (05.09)**	94.80 (05.65)	95.53 (05.28)
Rectas45	**99.00 (02.13)**	98.90 (02.31)	08.00 (08.58)	98.80 (02.47)
Ripley	88.51 (02.80)	88.60 (02.40)	88.63 (02.95)	**88.84 (02.60)**
Wine	80.09 (09.47)	87.14 (08.56)	**88.97 (08.87)**	83.09 (09.75)

- Aleatorio: Simplemente aumentando el límite de evaluaciones se consigue una mejora notable, acotar los valores a números positivos también parece mejorar el resultado, aunque debido a la alta varianza no se puede asegurar esto.
- Iris: En este dominio ya se conseguían resultados cercanos al 95 % con todos los métodos probados, con las modificaciones no se ha mejorado ni empeorado este valor.
- Rectas45: En este dominio queda claro que las modificaciones que mejoran los resultados en un dominio pueden perjudicar a otro, en este caso al forzar los valores positivos se han eliminado transformaciones que producían la rotación de los datos imprescindibles para conseguir clasificar las instancias de este dominio.
- Ripley: Tampoco se ha encontrado ninguna modificación que mejore o empeore los resultados.
- Wine: Tanto el aumentar el número de evaluaciones como eliminar los valores negativos consiguen mejoras notables en la clasificación.

La conclusión de estos experimentos es que, aunque pueden mejorarse los resultados de clasificación a base de optimizar los parámetros, con mejoras notables en algunos casos y apenas perceptibles en otros, estos ajustes no son generalizables y requieren el estudio del dominio del problema en cada caso.

Tabla 5.3: **Resultados del Método Híbrido.** Resultados de evolucionar una matriz diagonal con un máximo de 100 evaluaciones de la función de *fitness* y utilizar el resultado para inicializar una matriz completa (no tiene por qué tener ceros) que se evoluciona con un máximo de 900 evaluaciones.

	Aleatorio	Rectas45	Wine
Ratio Clasificacion	85.93(9.44)	98.60(2.75)	93.40(6.23)

5.3. Evolución con *fitness* Ponderada

Los resultados hasta ahora muestran que, en ciertos casos, una matriz de transformación diagonal consigue resultados notablemente mejores que los de una matriz completa, pero que en otros se necesitan transformaciones que sólo se consiguen con una matriz que pueda tener en cualquiera de sus posiciones un valor distinto de cero. Por todo esto, cabe preguntarse si se podría desarrollar un método que llevara hacia un tipo de matriz u otro dependiendo de los resultados que se obtengan durante la evolución.

En primer lugar se probó un método que evolucionaba una matriz diagonal con un número bajo de evaluaciones (100), utilizando después la matriz obtenida como punto de partida para la evolución de una matriz completa. Sin embargo, los resultados de esta técnica híbrida (tabla 5.3) mostraban tendencia al estancamiento en la primera matriz diagonal, sin conseguir mejoras después.

La segunda aproximación al problema fue tratar de maximizar el número de ceros en la matriz, con la intención de simplificar la búsqueda, eliminando también el posible ruido, producto de la forma en que CMA-ES obtiene las instancias de cada generación. Para ello se modificó la función de *fitness* de forma que calculara dos objetivos y diera el resultado ponderado de éstos mediante la fórmula 5.1 :

$$fitness = ErrorRate \times Peso + (1 - Peso)(1 - \frac{nCeros}{Dimension}) \qquad (5.1)$$

Donde:

- $ErrorRate$: Índice de errores de clasificación $\frac{Instancias\ incorrectamente\ clasificadas}{total}$
- $Peso$: Importancia entre 0 y 1 que se le da a la clasificación.

Tabla 5.4: **Resultados del método ponderado para Aleatorio (Clasificación).** Medias y desviaciones típicas (entre paréntesis) del porcentaje de éxito de clasificación para las combinaciones de umbrales y pesos indicadas en el dominio Aleatorio.

Peso	01	0.5	0.7	09	1
Umbral					
0	57.76 (15.96)	62.03 (15.08)	**63.13 (15.94)**	62.36 (16.35)	61.80 (16.92)
0.2	52.46 (11.99)	57.43 (15.93)	60.53 (17.32)	61.50 (16.20)	**62.76 (17.15)**
1	53.30 (09.64)	98.10 (03.85)	98.06 (02.88)	**99.23 (01.94)**	98.16 (06.50)
2	50.00 (00.00)	97.23 (03.25)	97.43 (02.75)	98.43 (02.97)	**99.03 (02.08)**
3	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)
5	50.00 (00.00)	50.10 (01.00)	50.16 (01.66)	50.00 (00.00)	50.00 (00.00)

- $nCeros$: Número de ceros en la matriz.
- $Dimension$: Número de elementos de la matriz.

Recordando el funcionamiento de CMA-ES, que trata de minimizar el valor de *fitness*, lo que se obtiene con esta fórmula es reducir el número de errores de clasificación, con una importancia $Peso$, y maximizar el número de ceros, normalizado respecto al tamaño de la matriz, con una importancia $1 - Peso$.

Como se trata de valores reales y no enteros, se redondearon todos los valores inferiores en valor absoluto a cierto umbral, igualándolos a cero a la hora de transformar los datos y de calcular la nueva generación del algoritmo evolutivo. Se probaron distintos valores de umbral para encontrar el mejor y comprobar si era distinto para cada dominio.

Con el dominio *Aleatorio*, que obtenía mejores resultados para la matriz diagonal que para la completa se obtuvieron los resultados de clasificación mostrados en la tabla 5.4 , con el número de ceros medio en la tabla 5.5 . Como contraste, se pueden comparar estos resultados con los obtenidos en la sección 5.1 (tabla 5.1).

La mejora en los resultados es evidente, llegando a superar al método que utilizaba 10 veces más evaluaciones. El caso base (umbral 0, peso 1) ofrece un porcentaje de aciertos en la clasificación del 61.80 %, mientras que el mejor encontrado (con umbral 1 y peso 90 %) es del 99.23 %. Se ha realizado la misma prueba para los dominios Wine y Rectas45, con dos objetivos: comprobar la validez del método y ver si los mejores resultados seguían consiguiéndose para los mismos valores de umbral y peso, con lo

Tabla 5.5: **Resultados del método ponderado para Aleatorio (Ceros).** Medias y desviaciones típicas (entre paréntesis) del número total de ceros en las matrices para las combinaciones de umbrales y pesos indicadas.

Peso	01	05	0.7	09	1
Umbral					
0	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)
0.2	8.87 (1.92)	7.93 (2.20)	6.61 (2.02)	2.59 (2.40)	1.46 (1.78)
1	15.64 (0.57)	13.89 (0.34)	13.69 (0.61)	12.57 (1.17)	10.00 (1.48)
2	16.00 (0.00)	14.00 (0.00)	13.99 (0.10)	13.61 (0.49)	12.04 (1.21)
3	16.00 (0.00)	16.00 (0.00)	16.00 (0.00)	16.00 (0.00)	16.00 (0.00)
5	16.00 (0.00)	15.99 (0.10)	15.99 (0.10)	16.00 (0.00)	16.00 (0.00)

Tabla 5.6: **Resultados del método ponderado para Wine (Clasificación).** Medias y desviaciones típicas (entre paréntesis) del porcentaje de éxito de clasificación para las combinaciones de umbrales y pesos indicadas para el dominio Wine.

Peso	01	05	0.7	09	1
Umbral					
0	**81.21 (09.92)**	80.90 (09.84)	80.20 (07.97)	80.97 (09.62)	80.96 (09.03)
0.2	73.03 (09.63)	79.46 (09.84)	**81.62 (08.68)**	81.14 (09.53)	81.10 (09.09)
1	75.74 (09.86)	84.99 (09.45)	**87.70 (08.24)**	84.56 (09.93)	85.08 (09.47)
2	71.54 (26.07)	91.40 (06.27)	90.99 (06.67)	**91.94 (06.57)**	91.66 (06.76)
3	34.53 (08.69)	35.41 (09.84)	33.93 (04.83)	**37.69 (14.56)**	36.30 (11.60)
5	35.59 (09.77)	35.35 (09.62)	35.28 (08.26)	**36.12 (10.29)**	35.75 (10.81)

que podrían considerarse independientes del dominio y utilizarse siempre los mismos.

Analizando los dominios por separado y comparando con resultados anteriores (tabla 5.1), se puede ver como para el dominio Wine se consigue una mejora notable (del 80.86 % al 91.94 % en el mejor caso) aunque sin llegar al resultado de la matriz diagonal. En el dominio Rectas45 el mejor resultado (98.95 %) se obtiene con el umbral en 0 y el peso de clasificación al 1 %, como es prácticamente imposible que se genere un 0 entero, al final todo el peso del *fitness* recae sobre el valor de clasificación y es equivalente a realizar el proceso sin considerar los ceros.

Con este método se han obtenido resultados cercanos a los que aportaba la matriz diagonal, con la diferencia de que también ha funcionado para un dominio explícitamente diseñado para requerir rotaciones en la transformación de los datos que no podían obtenerse con matrices de transformación diagonales. La única complicación se encuentra en la arbitrariedad de los

Tabla 5.7: **Resultados del método ponderado para Rectas45 (Clasificación).** Medias y desviaciones típicas (entre paréntesis) del porcentaje de éxito de clasificación para las combinaciones de umbrales y pesos indicadas para el dominio Rectas45.

Peso	01	05	0.7	09	1
Umbral					
0	**98.95 (02.16)**	98.60 (02.57)	98.90 (02.08)	98.90 (02.19)	98.70 (02.62)
0.2	44.85 (39.08)	96.90 (05.21)	96.70 (10.25)	**98.85 (02.34)**	97.90 (08.26)
1	49.50 (05.00)	51.40 (10.70)	75.95 (24.24)	87.85 (21.38)	**91.40 (18.02)**
2	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.50 (05.00)
3	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)
5	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)	50.00 (00.00)

parámetros *umbral* y *peso*, que tienen importancia en el resultado obtenido y son diferentes para cada dominio. Para poder utilizar este método de forma general, es necesario generalizar también la búsqueda de valores óptimos para estos parámetros.

5.4. Evolución Multiobjetivo

En la sección 5.3 se pudo ver cómo el buscar maximizar el número de ceros dentro de la matriz de transformación durante la evolución podía ayudar a mejorar los resultados finales de clasificación. El problema residía en encontrar los valores óptimos de umbral (qué se considera un cero) y el balance entre la importancia de la clasificación y el número de ceros.

Para resolver el primer problema se propuso añadir a los valores que optimiza el algoritmo el valor de umbral, con esto se evita tener que conocer de antemano un valor óptimo según el dominio o encontrarlo a base de prueba y error con un gran consumo de tiempo y recursos, con el posible riesgo de que la evolución sea más lenta.

Para incluir el umbral en el genotipo que evoluciona CMA-ES, basta con introducirlo en el vector que representa dicho genotipo, de esta forma se tendría:

$$INDIVIDUO_n = (umbral,\ m_{1,1},\ m_{1,2},\ \ldots,\ m_{k,k})$$

Con $m_{i,j}$ los distintos valores de la matriz de transformación y k la

Tabla 5.8: **Resultados evolucionando el umbral.** Media y desviación típica del ratio, en porcentaje, de clasificación correcta para el conjunto de test cuando se evolucionó con CMA-ES la matriz de transformación y el umbral que se le aplicaría a la vez.

Aleatorio	Rectas45	Wine
99.23 (01.94)	98.70 (02.20)	91.29 (06.88)

dimensión del dominio. Cuando se fuera a evaluar el individuo habría que aplicar el umbral:

$$m_{i,j} = \begin{cases} 0 & \text{si } m_{i,j} < umbral \\ m_{i,j} & \text{si } m_{i,j} >= umbral \end{cases} \tag{5.2}$$

En la tabla 5.8 se puede ver que los resultados de esto son excelentes comparados con el resto de pruebas que se han realizado anteriormente. Esto es así aún sin considerar la idea de maximizar el número de ceros. Se debe tener en cuenta ademas que se ha mantenido la configuración, en cuanto a número de evaluaciones y demás, que se utilizó en el resto de pruebas con CMA-ES.

Un ejemplo de matriz obtenida con este método es:

$$\begin{pmatrix} 0 & 0 & -3{,}7789 & 0 \\ 0 & 0 & 3{,}8901 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix},$$

Que tiene sobre el dominio aleatorio el efecto mostrado en la figura 5.3, Se muestran la representación de las 3 primeras dimensiones, pero la cuarta sufre el mismo efecto que las dos primeras, se quedan a 0 todos los valores.

Para tener en cuenta la idea de aumentar el número de ceros sin tener que establecer un nivel de importancia o peso para optimizar un objetivo u otro se decidió pasar a un método evolutivo multiobjetivo. Se utilizó la implementación de MATLAB de NSGA-II, estableciendo un límite de 2 minutos para la evaluación para poder realizar las pruebas completas de validación

Figura 5.3: **Dominio Aleatorio tras la Transformación.** Representación de las 3 primeras dimensiones del dominio Aleatorio tras aplicarle una transformación obtenida evolucionando el umbral. La dimensión que no se muestra (x_4) quedó también reducida toda a 0, por lo que queda sólo una dimensión representativa.

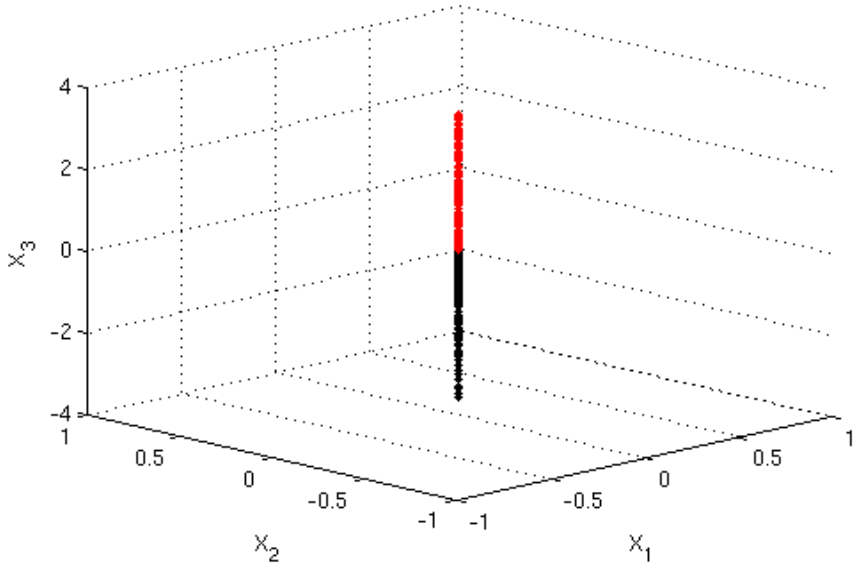


Tabla 5.9: **Resultados del método multiobjetivo.**
Resultados de clasificación en el conjunto de test para las matrices evolucionadas con los objetivos de minimizar el error de clasificación más los indicados en la tabla.

	Aleatorio	Rectas45	Wine
Ceros	**99.60 (01.66)**	**99.55 (01.60)**	91.25 (06.37)
Ceros,Umbral	97.40 (03.12)	99.10 (02.05)	**91.94 (06.07)**
1/Desviación	54.30 (11.69)	98.90 (02.52)	77.33 (09.39)
Media/Desviación	61.16 (10.61)	**99.55 (01.60)**	75.11 (09.37)
Ceros noD	76.86 (12.87)	95.45 (12.99)	91.30 (00.67)

cruzada en un tiempo razonable. Del frente que devuelve el algoritmo se eligió el individuo con mejores resultados de clasificación para el conjunto de entrenamiento.

En la tabla 5.9 se pueden ver los resultados para diferentes objetivos. A parte del de minimizar el error de clasificación, que es un objetivo en todos los casos, se probaron los siguientes:

- Ceros: Maximizar el número de ceros.
- Umbral: Minimizar el valor absoluto a partir del cual un valor de la matriz de transformación se considerará igual a cero.
- 1/Desviación: Se pretende maximizar la desviación típica para tener valores separados en la matriz.
- Media/Desviación: Para intentar que los valores sean o muy pequeños o muy grandes se intenta minimizar la media a la vez que se aumenta la desviación típica.
- Ceros noD: Sólo se aplica el umbral y se maximiza el número de ceros en las posiciones fuera de la diagonal. La intención es tratar de dar preferencia a las matrices diagonales sin eliminar la posibilidad de las completas.

Análisis de Resultados y Frentes Finales

A continuación se muestran ejemplos de frentes obtenidos para cada conjunto de objetivos junto con la conclusión del resultado de clasificación.

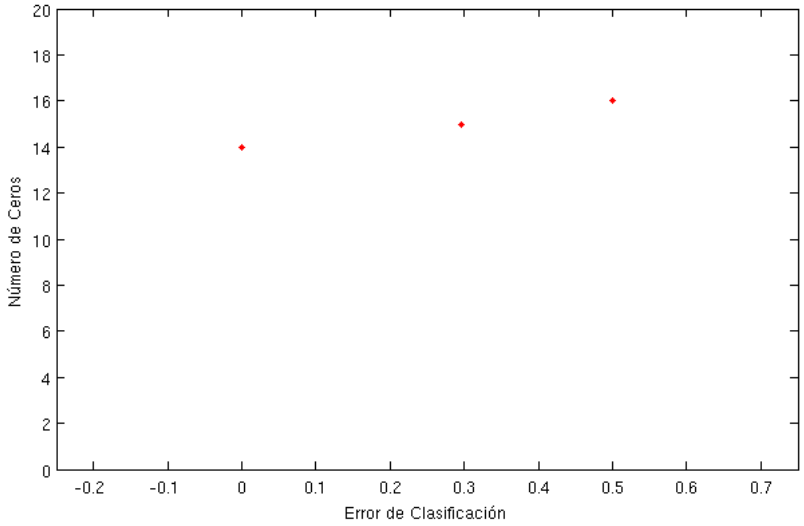
- Maximizar número de ceros: Figura 5.4. Los resultados para los dominios aleatorio y rectas45 son ligeramente superiores a todos los obtenidos hasta el momento aunque, si se tiene en cuenta la desviación, la diferencia no es notable respecto a otros métodos con resultados cercanos al 98 %. En el dominio Wine, sin embargo, se sigue sin alcanzar los ratios de clasificación de los métodos con diagonales de la sección 5.1. Aunque no se aprecia en la figura, el frente está formado por 89 puntos, 86 de ellos con un valor de umbral tan alto que deja a 0 todas las posiciones de la matriz y la clasificación se queda en el 50 %.

- Maximizar número de ceros, minimizar umbral: El frente tridimensional se muestra en la figura 5.7, con los perfiles en 5.8. Se puede ver tanto la dependencia entre el número de ceros y el umbral, como el rango de ceros en el que se obtienen los mejores valores de clasificación. El utilizar dos objetivos opuestos permite una exploración más detallada del frente en comparación con el resto de conjuntos de objetivos probados, sin embargo si lo que se busca es exclusivamente encontrar el mejor valor de clasificación en el menor tiempo posible, con el objetivo de minimizar el umbral parece que sólo se está reduciendo el rendimiento, ya que los resultados son muy similares a los del punto anterior. Por otra parte en las figuras correspondientes al dominio de las rectas rotadas (5.9) se ve como, aunque en este caso aumentar el número de ceros empeora el resultado de clasificación, se sigue explorando el espacio de búsqueda de forma que se encuentran buenas soluciones.

- Maximizar desviación: Figura 5.5. El valor de desviación no parece ofrecer ventajas en la búsqueda de la matriz de transformación.

- Minimizar $\frac{Media}{Desviacion}$: Figura 5.6. Los resultados son ligeramente mejores pero aún así no llegan a los de los demás métodos. Para comprobar que los valores de inicialización (por defecto en $[0, 1]$), se realizó una prueba con menos ejecuciones iniciando en los valores de la matriz en $[-100, 100]$, pero los resultados fueron similares. Esto es lógico según la teoría ya que se pueden conseguir transformaciones similares con

Figura 5.4: **Frente para Aleatorio: Maximizar Ceros.** Aunque el frente está formado por 89 puntos, casi todos ellos se acumulan en valores muy cercanos entre sí.



matrices proporcionales, p.ej. escalar una dimensión en $\frac{1}{10}$ es equivalente a multiplicar todas las demás por 10, o a escalarla por $\frac{1}{100}$ y a las demás por $\frac{1}{10}$, etc.

- Maximizar ceros fuera de la diagonal: Aunque el frente (figura 5.10) parece mostrar tendencia a mejorar los resultados cuanto menor es el número de ceros, los resultado están por debajo de los de otras pruebas pruebas.

El trabajo con la desviación, aunque no consiguiera buenos resultados, sí que muestra que, aún siendo los objetivos en teoría independientes, el resto de objetivos elegidos afecta a la búsqueda y al resultado de clasificación final.

Comparando con lo visto en la tabla 5.8 o con los mejores resultados del método ponderado en la sección 5.3, se comprueba que los resultados con los objetivos de minimizar los errores de clasificación y maximizar el número de ceros son cercanos a los mejores obtenidos hasta el momento. De estos experimentos se puede concluir que:

- El método multiobjetivo permite obtener de forma automatizada re-

Figura 5.5: **Frente para Aleatorio: Maximizar Desviación.**

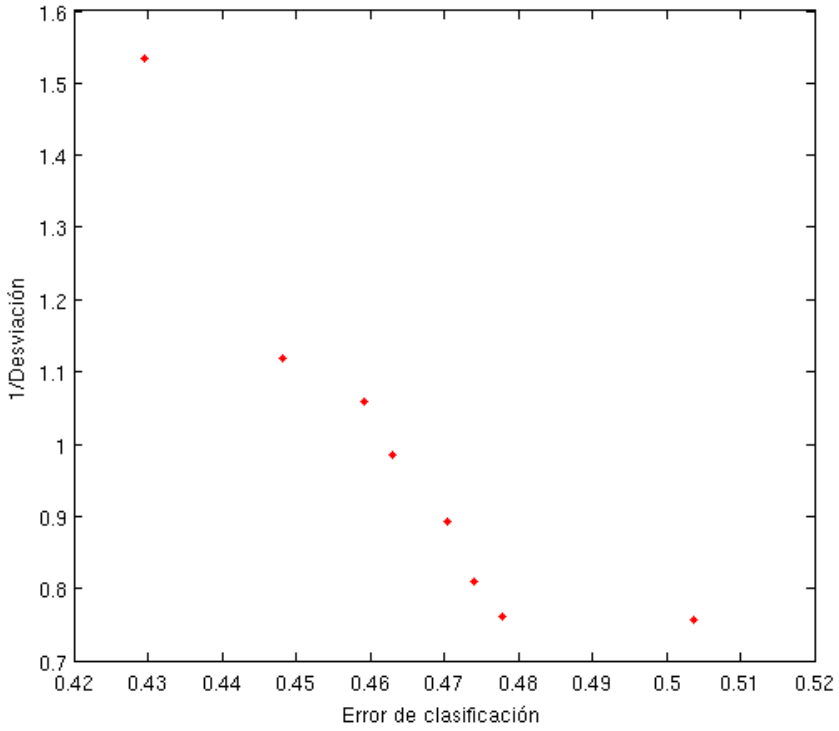


Figura 5.6: **Frente para Wine: Maximizar Desviación y Minimizar Media.**

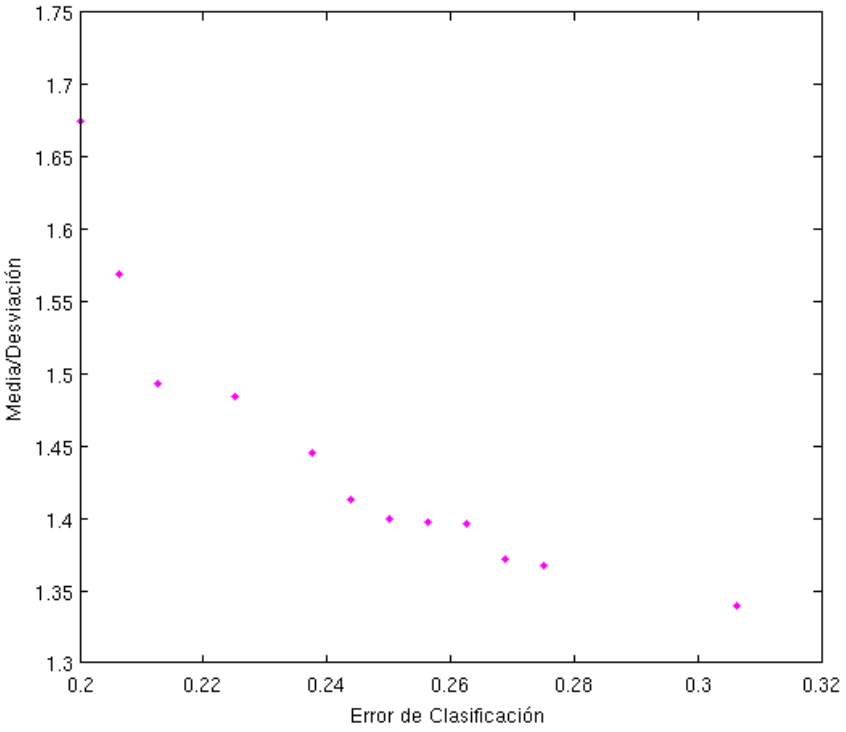


Figura 5.7: **Frente 3d para Wine: Minimizar Umbral y Maximizar Ceros.** Al introducir el objetivo de minimizar el umbral, opuesto al de maximizar el número de ceros, se consigue una exploración más amplia del frente, pero no por ello unos mejores resultados de clasificación.

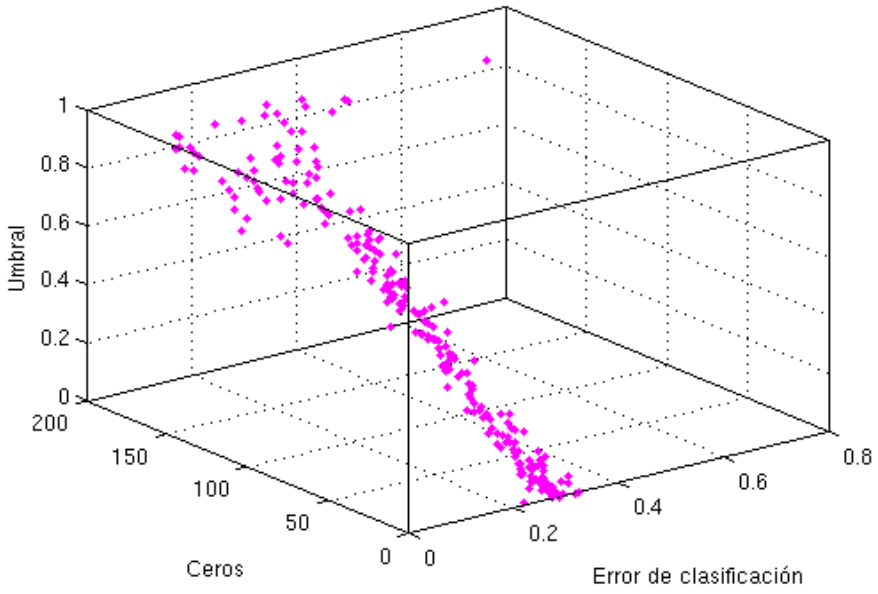


Figura 5.8: **Frentes 2d para Wine: Minimizar Umbral y Maximizar Ceros.** Se puede ver que en torno a los 130-170 ceros se consiguen los mejores resultados de clasificación.

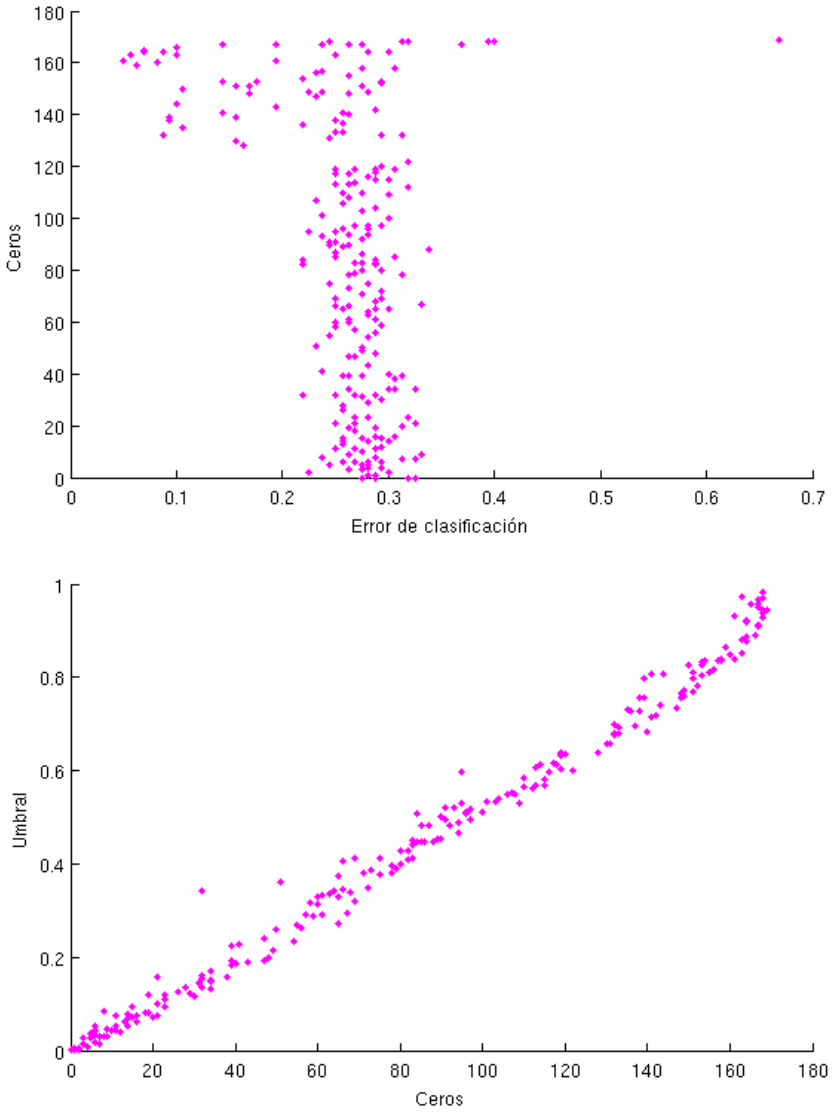


Figura 5.9: **Frente 3d para Rectas Rotadas: Minimizar Umbral y Maximizar Ceros.** Aunque se trate de objetivos opuestos en este dominio, se exploran las soluciones suficientes para encontrar buenos resultados para ambos objetivos.

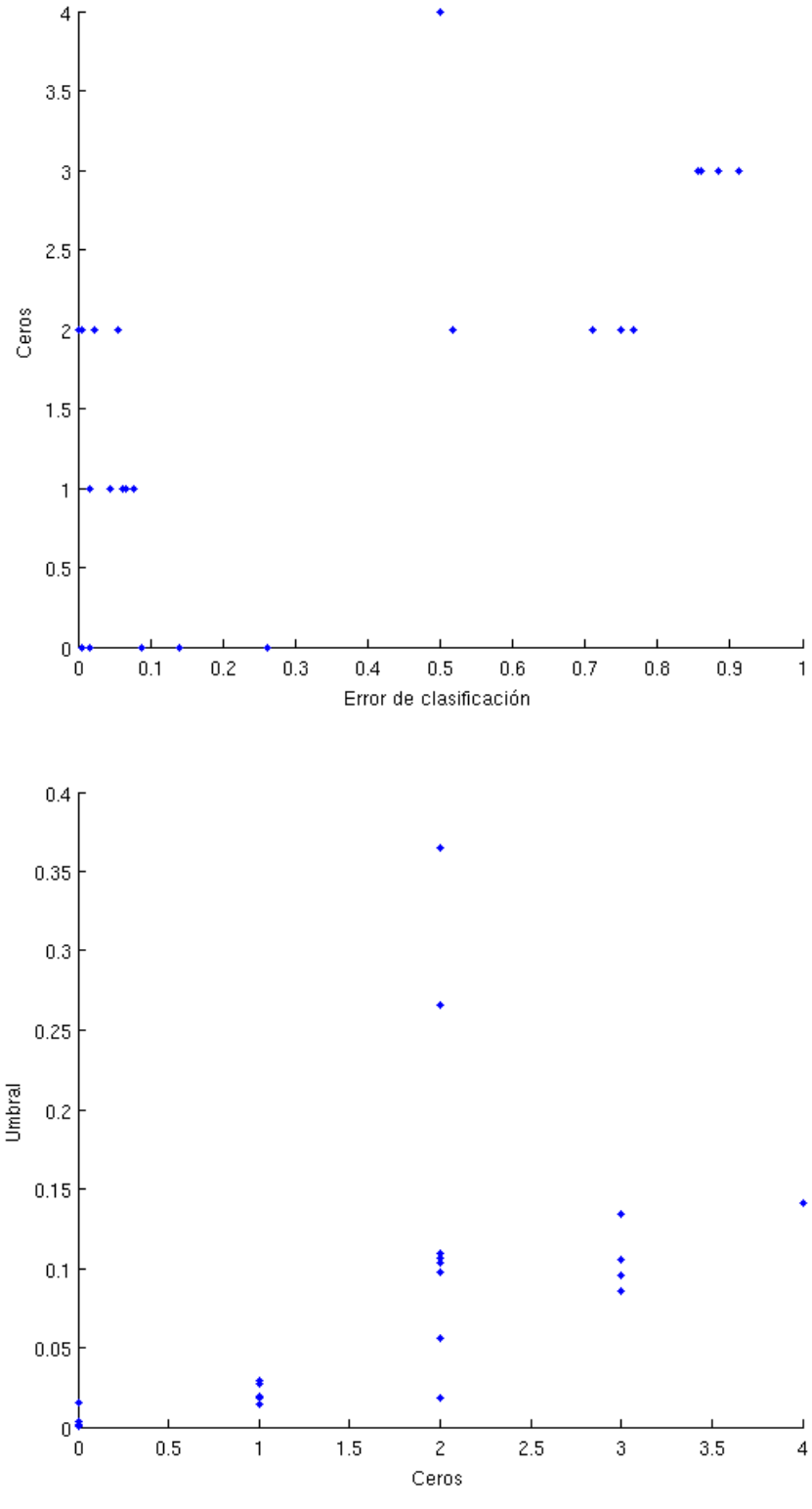
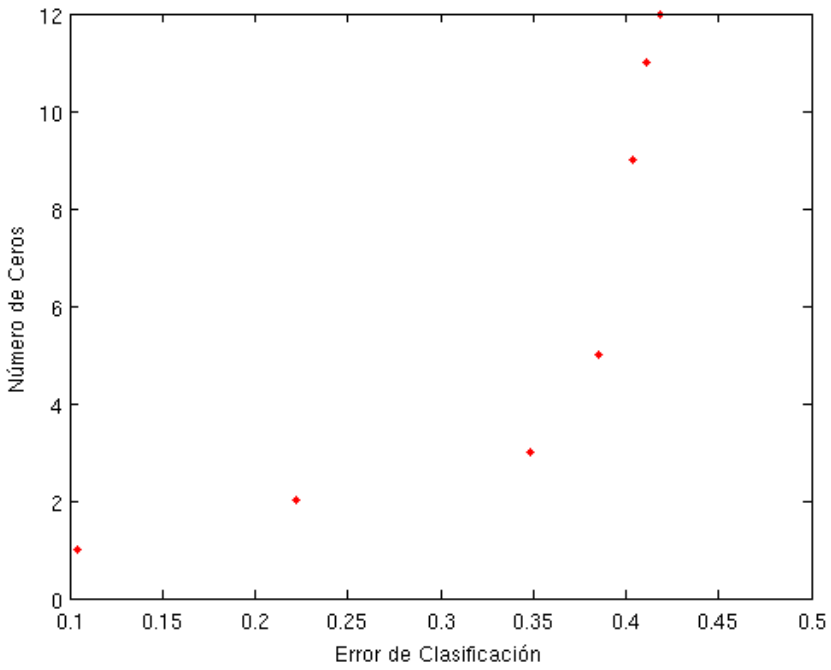


Figura 5.10: **Frente para Aleatorio: Maximizar Ceros Fuera de la Diagonal.**



sultados tan buenos como otras técnicas que requieren modificaciones en cuanto a metodología y optimización manual de parámetros.

- Aun así, teniendo en cuenta los resultados al incluir el valor de umbral en CMA-ES, puede ser posible encontrar métodos relativamente más simples con la misma capacidad de mejorar la clasificación.
- Hay que resaltar que resulta muy sencillo añadir nuevos objetivos o modificar los ya existentes a la implementación con NSGA-II, por lo que se podrían incluir metas como minimizar los falsos positivos para una clase o la confusión entre dos clases específicas. Con esto se obtendría un frente distinto que tuviera en cuenta estas características y se podría elegir el individuo que mantuviera el mejor compromiso posible entre todos los objetivos que se deseasen.

Por todo esto, la elección del método multiobjetivo se debe basar en una consideración entre rendimiento y flexibilidad.

5.5. Pruebas finales: comparativas de la evolución del umbral con otros métodos

El método que ha obtenido los mejores resultados de clasificación entre todos los probados ha sido el de incluir el umbral en el genotipo con CMAES, por lo que se ha realizado una batería de pruebas extendida con múltiples dominios para comprobar hasta qué punto es generalizable este método. Además, se ha utilizado el método descrito en [23], que básicamente consiste en construir una distribución normal con la diferencia entre los resultados de todos los *fold* de todas las ejecuciones externas y aplicarle un *test t de Student*, específicamente el *corrected replicated k-fold cross validation t-test* descrito en la referencia, comprobando así que las diferencias entre los resultados obtenidos son estadísticamente significativas. Si el valor de α es menor a 0.01, la diferencia se considera significativa.

Se ha comparado con los resultados sin realizar ninguna transformación y con los métodos que obtuvieron los mejores resultados en las primeras pruebas (sección 5.1):

- Euclídea: No se realiza transformación alguna de los datos.
- LS: Búsqueda Local.
- CMA: Evolución de la matriz de transformación con CMA-ES
- diag: la matriz de transformación es diagonal.
- comp: la matriz de transformación puede tener valores distintos de cero en todas sus posiciones.

Los dominios reales que se han añadido, todos extraídos del repositorio UCI, son:

- Bupa: Problemas de hígado en hombres adultos. 345 instancias de 7 atributos.
- Car: Agrupación de automóviles en distintos tipos. 1728 instancias con 6 variables.
- Diabetes: Diabetes en mujeres nativas americanas de más de 21 años. 768 instancias de 8 atributos.

- Ionosphere: Pulsos enviados mediante antenas de alta frecuencia en busca de estructuras reconocibles en la ionosfera. 351 instancias con 34 atributos.
- Yeast: Localización de proteínas. 1484 instancias con 8 atributos.

Tabla 5.10: **Comparación de mejoras y test de significación estadística.** Se compara la media del porcentaje de individuos clasificados correctamente para los conjuntos de test con la obtenida originalmente sin transformaciones.

Dominio	Resultado CMAES+Umbral	Resultado sin transformación	¿Diferencia significativa?
Aleatorio	99.23	49.40	**sí**
Bupa	63.18	62.11	no
Car	97.54	87.35	**sí**
Diabetes	67.18	67.53	no
Ionosphere	89.13	86.98	**sí**
Iris	94.93	96	no
Rectas45	98.7	6.95	**sí**
Ripley	88.56	88.71	no
Wine	91.30	76.14	**sí**
Yeast	51.64	51.97	no

En la comparación con los datos sin transformar (tabla 5.10), los resultados son significativamente mejores para el 50 % de los dominios. Estas diferencias son más grandes en el caso de los dominios artificiales, pero aún con esto en varios de los dominios reales hay una mejora notable.

Los valores de la tabla 5.11 muestran que en los experimentos en los que hubo diferencia significativa en los resultados entre los mejores métodos, ésta fue a favor del método que incluía el umbral en el genotipo, exceptuando el dominio *Wine*.

Tabla 5.11: **Comparación de los mejores resultados y test de significación estadística.** Comparación entre CMA-ES evolucionando matriz de transformación y umbral y el método, a parte de éste, que obtuvo los mejores resultados. Se muestran las medias de aciertos en clasificación obtenidas mediante validación cruzada, el método con el que se compara y si hay significación en la diferencia.

Dominio	Mejor Método Anterior	Resultado Anterior	Resultado CMAES+Umbral	¿Diferencia significativa?
Aleatorio	CMA-diag	95.70	99.23	**sí**
Bupa	CMA-comp	64.17	63.18	no
Car	CMA-comp	97.51	97.54	no
Diabetes	Euclídea	67.53	67.18	no
Ionosphere	CMA-comp	86.60	89.13	**sí**
Iris	Euclídea	96	94.93	no
Rectas45	CMA-Comp	98.7	98.7	no
Ripley	LS-diag	88.89	88.56	no
Wine	CMA-diag	94.17	91.30	**sí**
Yeast	Euclídea	51.97	51.64	no

Capítulo 6

Conclusiones

6.1. Resumen del Proyecto y Conclusiones

Se han realizado un amplio número de experimentos para comprobar de qué formas se podían mejorar los resultados que conseguía el algoritmo de clasificación *K-Vecinos* en diversos dominios.

Se realizó en primer lugar una batería de pruebas sencilla basada en utilizar matrices diagonales y completas evolucionadas mediante búsqueda local y CMAES con sus opciones por defecto. Al comprobar que en algunos dominios las matrices diagonales conseguían resultados mejores o mucho mejores que las completas y, sabiendo que en otros dominios son necesarias transformaciones que sólo pueden conseguirse con matrices completas para obtener buenos resultados; se han buscado métodos que simplificaran las matrices buscadas pero mantuvieran la posibilidad de tener valores distintos a cero en todas las posiciones.

Los métodos probados fueron:

- Evolucionar una matriz diagonal y después una completa a partir de ella. En éste caso sólo se modificaba la matriz en el segundo paso si los resultados del primero producían unos ratios de clasificación pobres, por lo que el proceso terminaba siendo equivalente a realizar las dos pruebas completas.
- Utilizar un valor prefijado como umbral de forma que las posiciones

cuyo valor absoluto fuera inferior, se quedaran a cero. Además se calculó la función de *fitness* como media ponderada entre el ratio de clasificación erróneo del conjunto de entrenamiento y el número de ceros inverso, de forma que al minimizar esta *fitness*, se trataba de minimizar el error y maximizar el número de ceros. Se consiguieron algunos resultados muy buenos, pero para ello se tuvieron que realizar muchas pruebas para encontrar valores de umbral y de peso para la media ponderada que obtuvieran dichos resultados.

Como se intentaba encontrar un método general que no requiriera la ejecución de múltiples pruebas y el ajuste manual de parámetros, se decidió buscar métodos que permitieran eliminar estos parámetros o automatizar su optimización.

En primer lugar se introdujo el umbral a partir del que considerar un elemento como 0 en el genotipo que evoluciona CMAES, de forma que ahora éste se compondría de la matriz de transformación y este valor de umbral y la función de *fitness* evaluara únicamente el resultado de clasificación. Los resultados obtenidos fueron excelentes, al menos comparados con todos los obtenidos hasta el momento.

El segundo paso fue utilizar un algoritmo evolutivo multiobjetivo en lugar de CMAES, de forma que se explorara el espacio de soluciones que combinara buenos resultados de clasificación y distintas cantidades de ceros en la matriz de transformación. Se realizaron pruebas con distintos grupos de objetivos, tanto para conseguir los mejores resultados como para comprobar que los objetivos que no eran directamente evaluar la clasificación afectaban a ésta. Se comprobó que esto sucedía, distintos conjuntos de objetivos, en los que en todos se incluía reducir el ratio de error de clasificación, llegaban a valores muy distintos de este ratio para los mismos dominios.

Con todo esto, los valores finales de clasificación correcta para el conjunto de test no superaban a los conseguidos con el método que utilizaba CMAES y el umbral. Por esto, si lo que se buscase fuera conseguir la mejor clasificación con el mínimo coste de recursos, este método sería más recomendable que los multiobjetivo. Éstos todavía podrían tener utilidad en algunos casos específicos en los que se tuvieran objetivos múltiples, como por ejemplo darle preferencia a falsos negativos para una clase sobre falsos positivos en otra.

Por último se realizaron unas pruebas finales para comprobar la validez

de la diferencia entre los resultados conseguidos con el método que parecía ser el mejor y las primeras pruebas realizadas. Se comprobó que, según el método utilizado, en aproximadamente el 30 % de los dominios esta diferencia era significativa y favorable al método con el umbral, quedando un método para el que el resultado era peor que los primeros que utilizaban directamente matrices diagonales.

Por todo esto, se puede decir que se han desarrollado métodos que consiguen mejorar los resultados de clasificación de *K-Vecinos* significativamente para múltiples dominios. Siguen existiendo dominios en los que las transformaciones no pueden mejorar la clasificación y, si verdaderamente se quiere encontrar la transformación óptima en cualquier dominio, se requiere un análisis más profundo que utilizar directamente el método estudiado en este proyecto que consigue los mejores resultados. Aún así no se empeora el resultado inicial sin transformación, por lo que en general el único argumento en contra de estos métodos serían los recursos necesarios para realizar el entrenamiento, mínimos en los dominios probados y sólo notables en dominios con un número de instancias para el entrenamiento muy alto: en dominios cercanos a las 2000 instancias, el entrenamiento en una ejecución tardó unos 30 minutos en un ordenador actual.

6.2. Consecución de Objetivos

Se ha conseguido el objetivo de mejorar la capacidad de clasificación de *K-Vecinos* (eficacia) para varios dominios (Generalidad). Además, el método desarrollado trabaja con rendimiento suficiente como para poder realizar la gran cantidad de ejecuciones necesarias (10 $Ejecuciones\ Externas \times$ 10 $Folds$) en un tiempo razonable (aproximadamente 2 horas para una batería de pruebas completa en el ordenador de pruebas)

Capítulo 7

Líneas Futuras

Al final del proyecto, aunque se hayan completado los objetivos, quedan muchos caminos por los cuales se podría continuar trabajando y extendiendo los métodos planteados y probados, con muy distintos niveles de dificultad y diferentes objetivos, como mejorar el rendimiento o aprovechar el trabajo realizado para otras tareas:

Prototipo. Se podría realizar un prototipo completo, bien mediante MATLAB, bien reescribiendo el programa en otro entorno, que permita comparar el método completo con alguna solución ya existente, en términos de eficacia, eficiencia, capacidad de generalización o reacción ante casos nuevos o extremos.

Optimización. Buscar e implementar optimizaciones, como mejorar la forma de calcular las distancias en *K-Vecinos* o eliminar partes de CMAES que no se estén utilizando.

Evolucionar otros parámetros. Debido al éxito, sin coste aparente en rendimiento, de introducir el umbral en el genotipo, se podrían introducir otros parámetros como la K que le indica a *K-Vecinos* el número de individuos más cercanos, de los que la clase de la mayoría será la que se adjudicará al individuo a clasificar. Durante todo el proyecto este valor se ha dejado como 1.

Otras transformaciones. Existen transformaciones más complejas que las que se pueden conseguir con una matriz como la utilizada en este

proyecto. Se podría intentar realizar tareas como desplazamientos en el espacio o rotaciones e inversiones sobre puntos que no fueran el origen de coordenadas y los ejes.

Transformaciones múltiples. Se podrían utilizar varias matrices de transformación distintas, de forma que afectaran a distintas zonas del espacio original o a distintas clases. Esto puede aumentar la versatilidad del método, aunque el coste en rendimiento al aumentar el espacio de búsqueda podría no merecer la pena salvo en contados casos.

Aproximaciones. Una forma de utilizar *K-Vecinos* es a la inversa, realizar aproximaciones de atributos desconocidos para individuos de clases conocidas. Dada una clase y un individuo del que se desconocen algunas características, se pueden aproximar sus valores mediante algún tipo de estimador que utilice los valores de ese atributo para los vecinos más cercanos. Para que esto se pudiera combinar con los métodos diseñados en este proyecto, habría que aproximar los valores desconocidos y después transformar el resultado al espacio euclídeo original. Esto no es trivial ya que requeriría que la matriz de transformación fuera invertible, lo que necesitaría cambios importantes en la evolución, de forma que no se tuvieran en cuenta las matrices no invertibles.

Bibliografía

[1] Duda, Richard O. *et Al Pattern Classification* John Wiley & Sons, 2001

[2] Pedro Isasi, Inés Galván *Redes de Neuronas Artificiales: Un Enfoque Práctico* Pearson Education, 2004

[3] Cover TM, Hart PE *Nearest neighbor pattern classification.* IEEE Transactions on Information Theory 13 (1): 21-27, 1967

[4] Mahalanobis, P.C. *On the generalised distance in statistics.* Proceedings of the National Institute of Sciences of India 2 (1): 49–55,1936

[5] Bentley, J.L. *Multidimensional Divide and Conquer.* Communications of the ACM, 23, 1980

[6] Lay, David C. *Álgebra Lineal y sus Aplicaciones.* Pearson Educación, 2007

[7] McCulloch, W. and Pitts, W. *A logical calculus of the ideas immanent in nervous activity.* Bulletin of Mathematical Biophysics, 1943

[8] Rosenblatt, Frank *The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.* Cornell Aeronautical Laboratory, Psychological Review, v65, No. 6, pp. 386-408. 1958

[9] L. J. Fogel, A. J. Owens y M. J. Walsh *Artificial Intelligence through Simulated Evolution.* New York: John Wiley, 1966.

[10] Ingo Rechenberg *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution.* Frommann-Holzboog, Stuttgart, 1973.

[11] Jonh Henry Holland *Adaptation in Natural and Artificial Systems.* University of Michigan Press, 1975.

[12] Koza, J.R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection* MIT Press, 1992

[13] Maurice Clerc *Particle Swarm Optimization.* ISTE, 2006.

[14] Hansen, N. and A. Ostermeier *Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation.* Proceedings of the 1996 IEEE International Conference on Evolutionary Computation, pp. 312-317, 1996

[15] Hansen, N. and A. Ostermeier *Convergence properties of evolution strategies with the derandomized covariance matrix adaptation: The* $(\mu/\mu_I, \lambda)$*-ES.* EUFIT'97, 5th Europ.Congr.on Intelligent Techniques and Soft Computing, Proceedings, Aachen, pp. 650-654. Verlag Mainz, Wissenschaftsverlag, 1997

[16] Hansen, N. and S. Kern *Evaluating the CMA Evolution Strategy on Multimodal Test Functions.* Eighth International Conference on Parallel Problem Solving from Nature PPSN VIII, Proceedings, pp. 282-291, Berlin: Springer, 2004

[17] Das, Indraneel and Dennis, John *Normal-Boundary Intersection: An Alternate Method For Generating Pareto Optimal Points In Multicriteria Optimization Problems* NASA, 1996

[18] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, T.Meyarivan *A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II.* IEEE Transacitions on Evolutionary Computation, Vol. 6, No.2, 2002

[19] Kohavi, Ron *A study of cross-validation and bootstrap for accuracy estimation and model selection* Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, 1995

[20] José M. Valls, Ricardo Aler. *Optimizing Data Transformations for Clasification Tasks.* 2007.

[21] Hamming, Richard W. *Error detecting and error correcting codes.* Bell System Technical Journal 26 (2): 147–160, 1950

[22] B.D. Ripley. *Pattern Recognition and Neural Networks.* Cambridge: Cambridge University Press, 1996.

[23] Remco R. Bouckaert, Eibe Frank. *Evaluating the Replicability of Significance Tests for Comparing Learning Algorithms.* Advances in Knowledge Discovery and Data Mining (PAKDD), 2004.

[24] Gantt, Henry Laurence *Work, wages, and profits* Engineering Magazine Co., 1913

Apéndice A

Manual del Programa

A continuación se incluye un breve manual que explica cómo utilizar el programa y cómo introducir modificaciones.

A.1. Manual de usuario

A.1.1. Uso del programa

La ejecución del programa requiere MATLAB. Un vez dentro de MATLAB, basta ir al directorio en el que se encuentre el archivo $clasificar.m$ y utilizar la función clasificar de la siguiente forma:

$clasificar(entrada, salida, metodo, tipoMatriz, nEjecuciones, nFolds)$

Con los siguientes parámetros:

- *entrada* Archivo con los valores de ejemplo del dominio de clasificación. Este archivo debe estar en el el subdirectorio *entradas* del directorio en el que se encuentre $clasificar.m$.
- *salida* Directorio en el que se deben guardar los resultados.
- *metodo* Cómo se va a calcular la transformación:
  - CMAES.

  - CMAES+umbral: Incluir el umbral en el genotipo de CMAES.
  - Híbrida: Comenzar evolucionando una matriz diagonal durante un límite del 10 % de las evaluaciones máximas para después pasar a una matriz completa el resto.
  - LS: Búsqueda local.
  - Ponderado: Utilizar CMAES con un *fitness* que sea media ponderada del número de ceros y error de clasificación. En la llamada a la función se deben incluir dos últimos parámetros $peso, umbral$ que indiquen el peso (sobre 1) que se le debe dar al error de clasificación y el umbral a partir del cual los valores inferiores se considerarán ceros.
  - Multiobjetivo: Evolución con múltiples objetivos utilizando la implementación de NSGA-II de MATLAB. Por defecto, los objetivos son reducir el error de clasificación, incrementar el número de ceros y minimizar el umbral.

- $tipoMatriz$ Diagonal o Completa.
- $nEjecuciones$ Ejecuciones externas completas de la validación cruzada para obtener una media de los resultados.
- $nFolds$ Número de *folds* en validación cruzada.

Para la ejecución de una batería completa de pruebas, se recomienda utilizar un *script* de MATLAB de la siguiente forma: preparar el árbol de directorios para los resultados y después utilizar un archivo con una lista de llamadas a clasificar con los parámetros convenientes.

A.1.2. Ficheros de entrada

El formato de los ficheros de entrada debe consistir simplemente en un individuo del dominio por línea con los valores de los atributos separados por espacios o tabulaciones, siendo la clase a la que pertenece el individuo el último atributo (un valor entero)

A.1.3. Resultados

En el directorio *salida*, indicado en la llamada a $clasificar$, el programa guarda varios ficheros:

- *Train.txt* : Resultados numerados de cada clasificación para cada conjunto de entrenamiento en cada ejecución externa.
- *Res.txt* : Resultados de clasificación para cada conjunto de test en cada ejecución externa y media final.Se incluye también el resultado de realizar la misma comparación sin realizar transformaciones y el número de ceros en la matriz de transformación. Por último, en la última línea se incluyen las medias y desviaciones típicas de los resultados.
- *Log.txt* : Además de los mismos resultados incluidos en *Res.txt*, se incluyen fecha, hora y dominio, número de ejecuciones externas y *folds*, matrices de transformación obtenidas para cada *fold* y tiempo total de ejecución, éste en la última línea del fichero.
- Frentes.txt: Si se trataba de un método multiobjetivo, se guardan los frentes finales correspondientes a cada *fold* de cada ejecución externa.

A.2. Manual de desarrollo

El programa es lo suficientemente sencillo para que cualquiera con unos conocimientos relativamente básicos de MATLAB y los suficientes conocimientos teóricos sobre los temas tratados en el proyecto pueda modificarlo o ampliarlo.

Como se explicó en el capítulo 4, *Desarrollo*, la aplicación se divide en tres partes fundamentales:

- Cuerpo principal, se podría dividir a su vez en:
  - Inicio: Se cargan los valores de los parámetros y se asignan valores por defecto, se cargan los datos del archivo de entrada.
  - Bucle de ejecuciones, en el que se realizan utiliza la función de optimización para conseguir la matriz de transformación y, una vez transformados los datos, se realiza la prueba sobre el conjunto de test. Esto se hace para cada ejecución externa. Los resultados se van escribiendo a los ficheros a mediada que se obtienen.
  - Final: Simplemente se calculan las medias y tiempos finales a partir de los datos almacenados y se cierran los flujos de escritura de datos,

La funcionar $clasificar$ realiza estas tareas, su cabecera es:

$clasificar(entrada, salida, metodo, tipoMatriz, nEjecuciones, nFolds, peso, umbral)$

La descripción de cada parámetro se puede encontrar en A.1.1

- Optimización: Prepara la ejecución de la técnica elegida para obtener la matriz de transformación. Técnicamente realiza una mejora y no una optimización, ya que no se puede asegurar que la transformación obtenida sea la óptima para el dominio. Se implementa en varias funciones $optimizar < Método > (parámetros)$, con diferentes parámetros según el método a utilizar. Devuelven todas una matriz de dos posiciones con la mejor transformación encontrada (aquella con menor ratio de error para su subconjunto de test) y su ratio de error. Las que utilizan métodos multiobjetivo devuelven además los miembros del frente final de la evolución y sus resultados de evaluación para cada objetivo.

- Cálculo de *fitness*: Distintos métodos pueden requerir distintas formas de evaluar los individuos en la técnica evolutiva, p.ej. si se evoluciona un vector que representa la diagonal de una matriz diagonal se deberá transformar primero en matriz para realizar la transformación. Por limitaciones del lenguaje las funciones de *fitness* son funciones internas dentro de la de optimización, mientras que las del resto de métodos se deben encontrar en archivos distintos. El formato es:

  $f = fitnessMetodo2(vector, datosTrain, claseTrain, dimensión)$

  $f = fitnessMetodo3(vector, dimensión, datosTrain, claseTrain, peso, umbral)$

  $f = fitnessMultiobjetivo1(vector)$

  $f = fitnessMultiobjetivo2(vector)$

  Como se muestra, las funciones que calculan el *fitness* del método multiobjetivo sólo pueden recibir el el vector a evaluar, mientras que las de los demás métodos pueden recibir más parámetros. Esto explica por qué las funciones de *fitness* para los método multiobjetivo son funciones internas, de esta forma se evita el uso de variables globales para compartir información entre las dos partes de la aplicación.

A.3. Contenidos del CD

El disco adjunto a esta memoria contiene todos los resultados de las pruebas realizadas cuyas estadísticas se han incluido en las tablas de la sección de resultados, además del código y varios de los archivos (figuras, hojas de cálculo utilizados para realizar esta memoria.

memoria El presente documento en formato PDF[1]

codigo Ficheros *clasificar.m* con el programa y *lanzar.m* con las pruebas.

entradas Dominios de entrada : *Aleatorio*100.*txt*, *bupa.txt*, *car.txt*, *diabetes.txt*, *ionosphere.txt*, *iris.txt*, *rectas*45.*txt*, *ripley.txt*, *wine.txt* y *yeast.txt*.

resultados Separados por directorios según método y dominio, además se incluye la hoja de cálculo *Resultados.txt* con los *t-test*, pruebas de significación estadística, utilizadas en la sección 5.5.

planificacion Archivos con el diagrama de Gantt con la planificación y hoja de cálculo con el presupuesto, incluidos también como archivo HTML[2] y documento PDF respectivamente.

[1] *Portable Documents Format* `http:\www.adobe.com`
[2] *HyperText Markup Language* `http:\www.w3.org`

Apéndice B

Planificación y Presupuesto

En este anexo se detallan los temas relativos a la organización del proyecto.

B.1. Planificación

Para la planificación del proyecto éste se dividió en 5 fases:

Estudio Previo. Establecimiento del contexto, estudio de técnicas disponibles y viabilidad del proyecto. Análisis del problema.

Desarrollo. Diseño e implementación del programa en MATLAB.

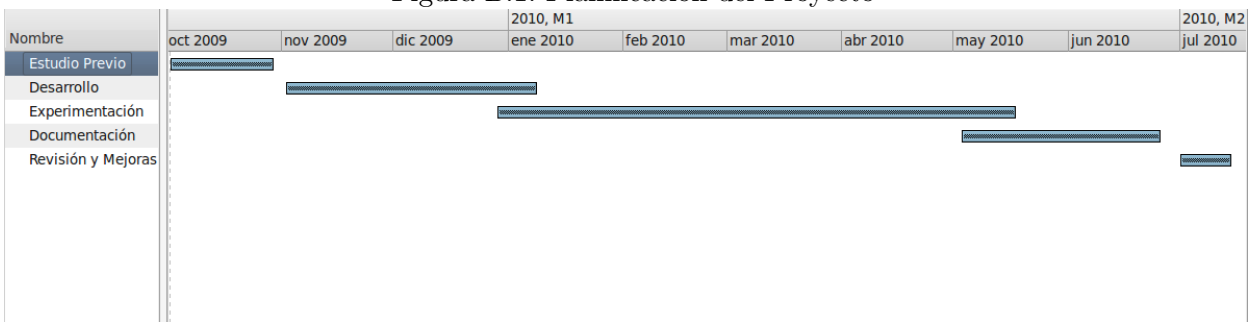
Experimentación. Selección de dominios, diseño y ejecución de las pruebas, refinamiento del código en base a éstas.

Documentación. Memoria del proyecto, revisión y mejora de la documentación del código.

Revisión y mejoras. Revisión general con los tutores del proyecto, correcciones.

El esquema seguido, teniendo en cuenta un horario de media jornada, se puede ver como gráfico Gantt [24] en B.1, las fechas concretas son:

Figura B.1: Planificación del Proyecto



Estudio Previo. Del 2 al 30 de octubre, 20 días laborables.

Desarrollo. Del 2 de Noviembre al 8 de enero, 50 días laborables.

Experimentación. 29 de diciembre al 27 de mayo, 100 días trabajados. Se solapa levemente con la fase de desarrollo mientras se ajustan posibles errores, tiempos de ejecución, formato de entrada/salida, etc.

Documentación. 3 de mayo a 25 de junio. De nuevo se solapa con los últimos días de la experimentación, mientras se realizan los últimos experimentos automáticamente ya se pueden documentar los resultados de los anteriores. 40 días.

Revisión y mejoras. Del 1 al 14 de julio. 10 días.

El total sería de 220 días trabajados a media jornada, el equivalente a 5 meses-persona.

B.2. Presupuesto

El proyecto ha requerido de unos materiales y software mínimos para su realización, por lo que al haber sido realizado por una sola persona cuenta con un presupuesto bastante sencillo y ajustado.

Los componentes del presupuesto son:

- Personal. Una persona trabajando media jornada durante 10 meses

según el coste por hora de un Ingeniero Superior en la Universidad Carlos III de Madrid sería de 21 447€.

- Ordenador de Pruebas.*Athlon X2* con 4 GB de RAM. 450€
- Portátil de Trabajo. Acer Aspire One, 200 €
- Licencia de MATLAB. El coste de una licencia de tipo académico es de 500€
- Licencias para bibliotecas de MATLAB. Durante el proyecto se necesitaron la *Bioinformatics Toolbox* y *Statistics Toolbox*, $400 \times 2 = 800$€.
- El resto de software utilizado durante el proyecto es libre y gratuito. P.ej: Ubuntu Linux, Openoffice.org, Planner, Inkscape.

El presupuesto total es de 26 127€ VEINTISÉIS MIL CIENTO VEINTISIETE EUROS

Este presupuesto se ha realizado siguiendo la rúbrica de la Universidad Carlos III de Madrid, en el CD adjunto al proyecto se incluye la hoja de cálculo con los datos exactos.