Universidad
Carlos III de Madrid

Departamento de INFORMÁTICA / CC E IA

PROYECTO FIN DE CARRERA

TÉCNICAS EVOLUTIVAS MULTIOBJETIVO PARA LA EVOLUCIÓN DE MATRICES DE PROYECCIÓN EN PROBLEMAS DE CLASIFICACIÓN

Autor: Moreno Garrido, Alfredo

Tutores: Aler Mur, Ricardo

Valls Ferrán, José María

Leganés, Octubre de 2012

Resumen

El trabajo presenta un estudio sobre la utilización de matrices obtenidas con técnicas de computación evolutiva con el fin de realizar proyecciones lineales sobre datos para obtener buenos resultados en el ámbito de la clasificación.

Se han realizado distintas experimentaciones, sobre varios dominios de datos, con matrices obtenidas mediante técnicas evolutivas como algoritmos genéticos y estrategias evolutivas, concretamente CMA-ES (Covariance Matrix Adaptation Evolution Strategy). La técnica de clasificación utilizada ha sido k-NN (K nearest neighbors).

Además, se han realizado experimentaciones sobre la proyección a dos dimensiones de los datos mediante el uso de matrices evolucionadas y se ha comparado con el uso de algoritmos bien conocidos como PCA y Sammon.

Por último, se ha realizado experimentación con algoritmos evolutivos multi-objetivo con el fin de encontrar matrices que proyectasen los datos a la menor dimensión posible aumentando los aciertos en clasificación respecto a los datos sin proyectar.

Etiquetas: computación evolutiva, clasificación, CMA-ES, k-NN, PCA, Sammon, reducción de dimensionalidad, extracción de características, transformaciones lineales, validación cruzada, estrategias evolutivas, algoritmos genéticos.

Abstract

The paper presents a study on the use of matrices obtained with evolutionary computation techniques to perform linear projections of data to obtain good results in the area of supervised classification.

Different experiments have been conducted on various domains of data with matrices obtained by evolutionary techniques like genetic algorithms and evolutionary strategies, specifically CMA-ES (Covariance Matrix Adaptation Evolution Strategy). Supervised classification technique used was k-NN (K nearest neighbors).

Furthermore, experiments have been performed on two-dimensional projection using evolved matrices and this method has been compared with the use of well-known algorithms such as PCA and Sammon.

Finally, there has been experimenting with multi-objective evolutionary algorithms to find the data matrices that project to the smallest possible dimension increasing hits on supervised classification respect to the data without projection.

Tags: evolutionary computation, classification, CMA-ES, k-NN, PCA, Sammon, dimensionality reduction, feature extraction, linear transformations, cross-validation, evolutionary strategies, genetic algorithms.

ÍNDICE DE CONTENIDOS

Introducción y objetivos 9
1.1 Introducción 9
1.2 Objetivos 9
1.3 Medios empleados 10
1.4 Estructura de la memoria 10
Estado del arte 12
2.1 Algoritmos de clasificación 12
2.1.1 Clasificadores basados en vecindad 14
2.1.2 Características de los datos 16
2.2 Transformaciones lineales sobre datos 17
2.3 Algoritmos de computación evolutiva 18
2.3.1 Estrategias evolutivas 19
2.3.1.1 CMA-ES 22
2.3.2 Algoritmos genéticos 24
2.3.2.1 Matlab genetic algorithm 28
2.3.3 Optimización multiobjetivo 29
2.4 Algoritmos de reducción de dimensionalidad 31
2.4.1 Análisis de componentes principales (PCA) 32
2.4.2 Método de Sammon 33
2.5 Validación cruzada 35
2.6 Implementación y experimentación ya existente 36
2.6.1 Algoritmo inicial 36
2.6.2 Dominios de datos 39

2.6.3 Experimentación existente 42
Experimentación 45
3.1 Estancamiento con matrices completas 45
3.1.1 Variación matriz inicial 46
3.1.2 Método de búsqueda alternativo: Algoritmo genético 47
3.1.3 Matrices simétricas con CMAES 50
3.2 Proyección a dos dimensiones 53
3.2.1 Representación gráfica de las proyecciones 55
3.3 Multiobjetivo 74
3.3.1 Vector de selección CMA-ES 74
3.3.2 Multiobjetivo con CMA-ES 76
3.3.3 Multiobjetivo con gamultiobj 77
3.3.3.1 Selección del punto del frente 78
3.3.3.2 Ejecuciones completas con gamultiobj 82
3.3.4 Resumen ejecuciones multiobjetivo 83
Conclusiones 85
Líneas futuras 86
Referencias Bibliográficas 87
APÉNDICE A: Manual del programa 89
Funciones principales 89
APÉNDICE B: Planificación y Presupuesto 96
B.1 Planificación 96
B.2 Presupuesto 97

ÍNDICE DE ILUSTRACIONES

Ilustración 1: Clasificador genérico 13
Ilustración 2: Clasificador k-NN 15
Ilustración 3: Codificación individuo algoritmo genético 24
Ilustración 4: Cruce de un punto 26
Ilustración 5: Cruce de dos puntos 26
Ilustración 6: Cruce uniforme 27
Ilustración 7: Frente de Pareto 30
Ilustración 8: Dominio rectas-45 40
Ilustración 9: Dominio rectas-0 41
Ilustración 10: Matriz Toeplitz 50
Ilustración 11: Matriz Toeplitz simétrica 50
Ilustración 12: Rectas0 2D CMA-ES 56
Ilustración 13: Rectas0 2D PCA 57
Ilustración 14: Rectas45 2D CMA-ES 58
Ilustración 15: Rectas45 2D CMA-ES 59
Ilustración 16: Bupa 2D CMA-ES 60
Ilustración 17: Bupa 2D PCA 61
Ilustración 18: Bupa 2D Sammon 61
Ilustración 19: Car 2D CMA-ES 62
Ilustración 20: Car 2D PCA 63
Ilustración 21: Car 2D Sammon 63
Ilustración 22: Iris 2D CMA-ES 64
Ilustración 23: Iris 2D PCA 65
Ilustración 24: Iris 2D Sammon 65
Ilustración 25: Ripley 2D CMA-ES 66
Ilustración 26: Ripley 2D PCA 67
Ilustración 27: Ripley 2D Sammon 67
Ilustración 28: Aleatorio100 2D CMA-ES 68
Ilustración 29: Aleatorio100 2D PCA 69
Ilustración 30: Aleatorio100 2D Sammon 69
Ilustración 31: Wine 2D CMA-ES 70
Ilustración 32: Wine 2D PCA 71
Ilustración 33: Wine 2D Sammon 71
Ilustración 34: Frente ejecución de gamultiobj 79
Ilustración 35: Planificación 96

ÍNDICE DE TABLAS

Tabla 1: Algoritmo de partida experimentación 37
Tabla 2: Resultados experimentación existente 43
Tabla 3: Estancamiento CMA-ES completa 45
Tabla 4: Variación matriz inicial completa CMA-ES 46
Tabla 5: Configuraciones algoritmo genético 48
Tabla 6: Resultados algoritmo genético 48
Tabla 7: Resultados matrices simétricas 51
Tabla 8: Resultados proyección 2D CMA-ES 54
Tabla 9: Tiempos ejecución 2D 73
Tabla 10: Multiobjetivo vector selección 75
Tabla 11: Valor fitness multiobjetivo CMA-ES 76
Tabla 12: Resultados multiobjetivo CMA-ES 77
Tabla 13: Resumen frentes de ejecuciones con gamultiobj 80
Tabla 14: Resultados gamultiobj 82
Tabla 15: Resumen ejecuciones multiobjetivo 83
Tabla 16: Tiempos ejecución multiobjetivo 84
Tabla 17: Nomenclatura tablas desarrollo 89
Tabla 18: Función prinGASimCMAES.m 90
Tabla 19: Función prinPCA.m 91
Tabla 20: Función prinSammon.m 91
Tabla 21: Función prin2DCMAES.m 92
Tabla 22: Función prin2DCMAES10ejec.m 93
Tabla 23: Función prinCMAESMultiobj.m 94
Tabla 24: Función prinGamultiobj.m 95

Capítulo 1

Introducción y objetivos

1.1 Introducción

Los problemas de clasificación sobre un dominio de datos han sido ampliamente estudiados en el campo del aprendizaje automático. En particular, los problemas de clasificación supervisada, en los que se cuenta con una base datos con muestras ya clasificadas, son especialmente útiles para construir modelos matemáticos de predicción que clasifiquen nuevas muestras de datos con gran probabilidad de acierto.

El algoritmo del vecino más cercano es una técnica de clasificación cuyo mecanismo de aprendizaje consiste simplemente en almacenar los datos de aprendizaje. Para clasificar nuevos datos, les asigna la clase del dato más cercano de entre los de aprendizaje. Uno de los problemas de esta técnica de clasificación es que es muy sensible a la función de distancia utilizada (típicamente la euclídea, pero no siempre esta es la más adecuada). Este problema se puede atacar modificando la función de distancia para adaptarla al problema. Una manera sencilla de modificar las distancias entre los datos es utilizar transformaciones lineales de los mismos, representadas por una matriz.

El presente trabajo se centra en el uso de técnicas evolutivas para la optimización de transformaciones lineales (representadas como matrices) sobre distintos dominios de datos, con el objetivo de conseguir mayor porcentaje de aciertos en problemas de clasificación. En concreto, se han utilizado las estrategias evolutivas (CMA-ES: Covariance Matrix Adaptation Evolution Strategy) y los algoritmos genéticos.

1.2 Objetivos

Los objetivos iniciales de proyecto se centran en el estudio de alternativas, en cuanto al algoritmo evolutivo de búsqueda utilizado y en torno al tipo de matriz completa empleada en la búsqueda, sobre algunos dominios de datos problemáticos. El propósito es el de intentar realizar una búsqueda lo más global posible que evite caer en mínimos locales.

El primero de los objetivos principales del trabajo es el de la utilización de matrices de proyección evolucionadas como método de visualización de distintos dominios de datos en dos dimensiones. Para ello, además se han contrastado los resultados obtenidos con dos métodos de reducción de dimensionalidad conocidos como son PCA y Sammon.

El segundo objetivo principal de este trabajo es el de explorar técnicas multiobjetivo que permitan reducir la dimensionalidad de un dominio de datos, con matrices de proyección evolucionadas, a la vez que se aumentan los aciertos en clasificación mediante un algoritmo de clasificación basado en el vecino más cercano.

1.3 Medios empleados

A lo largo del desarrollo de este proyecto se han empleado distintos recursos software y hardware que se exponen a continuación:

- Hardware:
  - Ordenador de sobremesa con procesador intel core i7, 6 GB de RAM y placa base ASUS Rampage II.
  - Ordenador portátil Acer Aspire con procesador intel core 2 Duo y 4 GB de RAM.
- Sofware:
  - Sistemas operativos:
    - ✓ Windows Vista 64 bits
    - ✓ Windows 7 Ultimate 32 bits
    - ✓ Ubuntu 10.04 32 bits
  - Matlab 7.8
  - Microsoft Office 2007
  - Open Office

1.4 Estructura de la memoria

En el capítulo 2 de la memoria se detallan de forma teórica algunos conceptos y algoritmos básicos relacionados con la inteligencia artificial.

El punto 2.1 se centra en la explicación general de los algoritmos de clasificación y en la descripción detallada de los algoritmos de clasificación basados en vecindad. También se comenta dentro del punto 2.1 algunas características típicas sobre los dominios de datos que influyen en su clasificación automática.

El apartado 2.2 describe en qué consisten las transformaciones lineales dentro del contexto del álgebra lineal.

A lo largo del apartado 2.3 se explica el funcionamiento de los algoritmos de computación evolutiva entrando en mayor detalle sobre los algoritmos genéticos, las estrategias evolutivas y la optimización multiobjetivo mediante algoritmos evolutivos.

En el punto 2.4 se realiza una exposición de los algoritmos de reducción de dimensionalidad, entrando en detalle sobre los algoritmos PCA y Sammon.

El apartado 2.5 explica en qué consiste el método de validación cruzada.

Por último, dentro del capítulo 2, en el apartado 2.6 se explica el algoritmo inicial utilizado en el proyecto, los dominios de datos sobre los que se realiza la experimentación y los resultados de experimentación existentes en el momento del comienzo de este trabajo.

El capítulo 3 está dedicado a la nueva experimentación realizada en este trabajo.

El punto 3.1 expone la experimentación realizada sobre dos dominios de datos sobre los cuales existe una problemática de estancamiento con la utilización de matrices completas (se explica en 2.6). Esta problemática se aborda variando la matriz de inicio en la búsqueda, variando el algoritmo de búsqueda y utilizando matrices simétricas.

El apartado 3.2 se centra en el estudio de la utilización de matrices de proyección evolucionadas como método de visualización de datos, contrastándolo con los métodos PCA y Sammon.

Por último dentro del capítulo 3.3 se realiza un estudio en profundidad sobre optimización multiobjetivo con la idea de realizar transformaciones sobre datos proyectándolos a la menor dimensión posible a la vez que se incrementan los porcentajes de aciertos en clasificación con un algoritmo de clasificación basado en vecindad.

En el capítulo 4 se exponen las conclusiones del trabajo.

Capítulo 2

Estado del arte

En primer lugar se hace una descripción general sobre algoritmos de clasificación, transformaciones lineales sobre datos, algoritmos de computación evolutiva y algoritmos de reducción de dimensionalidad de forma teórica, haciendo hincapié en aquellos algoritmos concretos utilizados a lo largo del proyecto. Esto comprende los apartados 2.1, 2.2, 2.3 y 2.4 respectivamente.

En el apartado 2.5 se explica detalladamente en qué consiste el método de validación cruzada utilizado en el proyecto.

Por último, en el apartado 2.6 se expone cual ha sido el punto de partida del proyecto en cuanto a la implementación y experimentación ya existentes.

2.1 Algoritmos de clasificación

Un algoritmo de clasificación [1, 2] cumple el objetivo de asignar, a una muestra de datos, una de las clases existentes. Estos algoritmos suelen estar formados por un conjunto de funciones de decisión, también llamadas funciones discriminantes, $G = \{g_1(x),...,g_N(x)\}$ que dividen el espacio de representación en N regiones.

Dentro del contexto descrito, un algoritmo de clasificación asigna un vector de características x a la clase c_i si:

$$g_i(x) > g_j(x)\ \forall\ i \neq j,\ 1 \leq i,j \leq N$$

Un clasificador podría expresarse de forma abstracta según la ilustración 1.

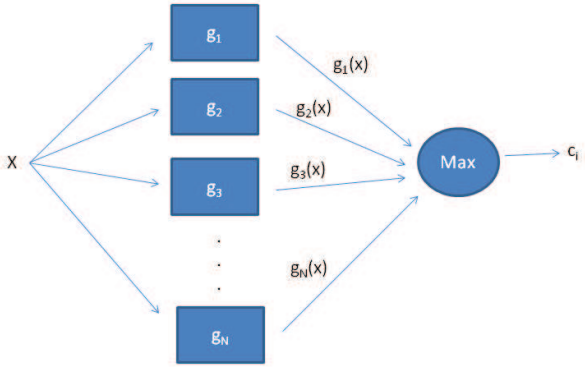

Ilustración 1: Clasificador genérico

La salida de cada una de las cajas son los valores de las funciones discriminantes y la salida del selector (máximo) es la etiqueta de la clase.

Dentro de las distintas técnicas de clasificación existen dos tipos principales, los métodos de clasificación clásicos y el agrupamiento o clustering.

En el agrupamiento o clustering no existe ninguna clase a priori sobre cada muestra, simplemente se establece el número de clases que queremos que haya y se deja que las defina algún tipo de procedimiento estadístico.

En la clasificación clásica se parte de la base de que se cuenta con un conjunto L de muestras clasificadas, $M = \{x_1, ..., x_L\}$, por tanto, la optimización de un criterio de aprendizaje podrá llevarse a cabo según alguno de los métodos siguientes:

- **Clasificación estadístico paramétrica**: Este método asume que las representaciones de las instancias en M siguen ciertas leyes estadísticas, cuyas formas funcionales son conocidas. Las funciones de clasificación se establecen en base a estas funciones y el aprendizaje consiste en la estimación de los parámetros, adoptando criterios como el de la máxima verosimilitud. Un ejemplo de este tipo de clasificadores es el de naïve Bayes.
- **Clasificación no estadístico paramétrica**: En este caso el problema se plantea como una decisión geométrica. Los parámetros de las funciones se determinan mediante técnicas de optimización, usando como criterio el error empírico de clasificación de la muestra de entrenamiento.

- **Clasificación basada en distancias**: Este es un caso particular del método no estadístico paramétrico en el cual las funciones discriminantes se establecen en base a medidas de disimilitud o distancias entre las representaciones de las instancias en *M*. Un ejemplo de este tipo de clasificadores es el de los *k* vecinos más cercanos.

A lo largo del proyecto se ha utilizado un clasificador del tipo regla del vecino más cercano que pasa a ser descrito en detalle en el siguiente sub-apartado.

2.1.1 Clasificadores basados en vecindad

El método de clasificación de los *k*-vecinos más cercanos [3, 4] (*k*-NN) (Fix y Hodges 1951) forma parte de la familia de técnicas de aprendizaje conocida como aprendizaje basado en ejemplos, es una técnica de clasificación basada en distancias. Este tipo de algoritmo almacena un conjunto de ejemplos de entrenamiento y etiquetan una nueva muestra en base a las instancias más próximas del grupo de entrenamiento, según una determinada medida de disimilitud.

Sea $T=(x_1,...,x_N)$ el conjunto de entrenamiento de muestras etiquetadas, tal que cada x_i tiene asociada una etiqueta de la clase c_i del conjunto de L clases, una clasificación por distancia mínima para una nueva instancia y puede definirse como:

$$y \in c_j \leftrightarrow \text{argmin}_i d(y, x_i)$$

donde d es la métrica definida en T, esta métrica debe cumplir los cuatro criterios siguientes:

1. No negatividad: $d(y, x) \geq 0$.
2. Identidad: $d(y, x) = 0 \leftrightarrow x = y$.
3. Simetría: $d(x, y) = d(y, x)$.
4. Desigualdad triangular: $d(y, x) \leq d(y, p) + d(p, x)$.

Una medida de similitud muy conocida para datos numéricos es la distancia Euclídea:

$$d(y, x) = \sqrt[2]{\sum_{j=1}^{n}(y_j - x_j)^2}$$

cuyo valor escalar se obtiene a partir de los atributos de las dos instancias.

Tomando k=1, una nueva muestra x se etiquetará en la clase asociada a su instancia más cercana. Para k ≥ 2, una nueva muestra x se asignará a la clase más representada entre las k instancias más próximas a la muestra.

En la ilustración 2 se muestra un ejemplo de aplicación de k-NN sobre un conjunto de datos de dos dimensiones:

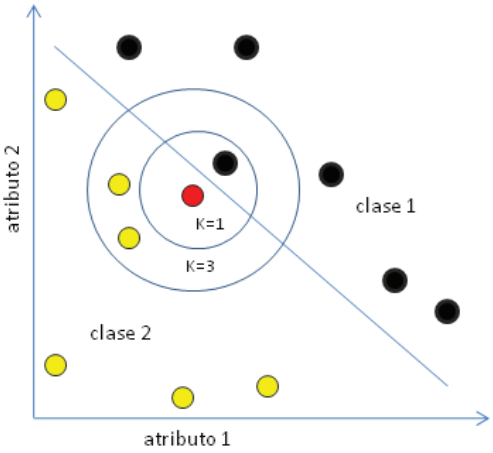

Ilustración 2: Clasificador k-NN

Matlab cuenta con una implementación del algoritmo k-NN[1] que ha sido utilizada a lo largo del proyecto, siempre con un valor k=1.

[1] http://www.mathworks.es/help/toolbox/bioinfo/ref/knnclassify.html

2.1.2 Características de los datos

El mayor o menor éxito de un método de clasificación [2] tiene una fuerte dependencia de la complejidad de los datos de entrenamiento utilizados. El conjunto de datos puede presentar características que dificulten el proceso de clasificación.

Algunas de estas características típicas son: el ruido, alta dimensionalidad y número de instancias, solapamiento entre clases y desbalanceo entre clases.

El ruido supone una anomalía para el conjunto de datos que puede estar originada por imprecisiones durante la recolección de los datos y por errores en el etiquetado de las muestras. La estrategia que debe llevarse a cabo para eliminar el ruido es la de filtrar las instancias poco confiables y descartarlas.

La alta dimensionalidad (gran número de atributos) y un elevado número de instancias hacen que el proceso de clasificación sea más costoso, requiriendo el algoritmo muchos recursos.

El solapamiento entre clases se define como la región donde, al menos, dos clases distintas presentan instancias que comparten información en algunos de sus atributos. Esto implica que dichas instancias son poco confiables ya que podría pertenecer a cualquiera de las clases implicadas.

Por último, tenemos el desbalanceo entre clases. En un problema de dos clases, el desbalanceo puede definirse como una situación en la cual existe un número de instancias significativamente más elevado de una clase (clase mayoritaria) que de la otra (clase minoritaria).

2.2 Transformaciones lineales sobre datos

Una aplicación o transformación [5] entre dos espacios vectoriales V y W,

$$f: V \rightarrow W$$

debe cumplir para que sea lineal:

$$f(u + v) = f(u) + f(v), \forall u, v \in V$$

$$f(\alpha v) = \alpha f(v), \forall v \in V, \forall \alpha \in K$$

Una transformación lineal puede representarse como una matriz que proyecte a un espacio de menor, mayor o igual dimensión que el original. Por tanto, pueden usarse matrices de proyección para realizar transformaciones sobre datos en problemas de clasificación. Cada muestra del conjunto original de datos se proyectaría de la siguiente forma:

$$muestra' = muestra \times M$$

La matriz de proyección *M* podrá tener distintas formas y en consecuencia podrá realizar distintos tipos de transformaciones como son el escalado y rotación de las muestras.

Para realizar una transformación de escalado básica basta con que la matriz sea diagonal, esto significa que la matriz debe ser cuadrada y que todas las posiciones de la matriz no correspondientes a la diagonal principal tendrán valor cero. Con una matriz diagonal solo podríamos, por tanto, realizar proyecciones sobre datos a la misma dimensión que la original. Cada atributo de la muestra se escalaría con el correspondiente valor de la diagonal principal de la matriz de proyección.

La transformación de rotación dependerá del espacio de datos original, según la dimensión que tengan los datos se podrá rotar sobre distintos ejes. En un espacio de dos dimensiones pueden rotarse las muestras un ángulo θ en dirección opuesta a las agujas del reloj con:

$$M = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

si se multiplica por la izquierda por un vector fila; o bien con:

$$M = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

si se multiplica por la derecha con un vector columna.

2.3 Algoritmos de computación evolutiva

Los algoritmos evolutivos [6] tratan de resolver problemas de búsqueda y optimización fundamentándose en el principio de "la supervivencia del más apto", postulado por Charles Darwin en su teoría de evolución de las especies.

El Darwinismo establece que la historia de la mayoría de la vida de nuestro planeta puede ser explicada a través de una serie de procesos estadísticos que actúan sobre y dentro de las poblaciones y especies [7]: reproducción, mutación, competencia y selección.

El término de computación evolutiva engloba una serie de técnicas de búsqueda y optimización estadística inspiradas en la teoría Darwiniana de la evolución natural. Se exponen a continuación algunos conceptos fundamentales que deben conocerse a la hora de utilizar cualquier algoritmo de computación evolutiva:

- Codificación del individuo: Un individuo debe quedar definido mediante una estructura de datos. El individuo típicamente representa una solución al problema que se está resolviendo.
- Función de aptitud o fitness: Tiene el propósito de valorar la aptitud del individuo, es decir, dirá como es de buena una solución para el problema que se esté tratando mediante el algoritmo evolutivo.
- Operadores genéticos: Afectan a los individuos de manera que los modifican de alguna forma, algunos de los operadores más típicos son mutación y cruce.
- Selección: Mecanismo de selección de individuos comparándolos a partir de la función de fitness.

Algunas de las técnicas de computación evolutiva más usadas son: estrategias evolutivas, algoritmos genéticos , programación genética [13] y optimización mediante colonias de hormigas [14]. Estos dos algoritmos evolutivos no se explican en detalle ya que no se han tenido en cuenta en el presente trabajo.

2.3.1 Estrategias evolutivas

Las estrategias evolutivas [8] fueron desarrolladas en los años 70 por Rechenberg y Schwefel. Este tipo de técnicas de computación evolutiva han sido ampliamente utilizadas en problemas de optimización numérica. Pueden clasificarse como métodos estocásticos de escalada con paso adaptativo, son en general bastante rápidos y una de sus principales características es que permiten la auto-adaptación de los parámetros de mutación. La representación de los individuos se realiza a través de vectores reales, la mutación se lleva a cabo mediante perturbación gaussiana.

Las estrategias evolutivas (EE) pueden clasificarse como simples o múltiples.

La nomenclatura utilizada para describir los distintos tipos de estrategias evolutivas es la siguiente:

- Población de μ individuos que generan una descendencia de λ individuos
- *(μ + λ):* Reemplazo por inclusión. Se juntan los λ descendientes con los μ progenitores en una única población y en ella se muestrean los μ individuos mejores.
- *(μ , λ):* Reemplazo por inserción. Se eligen a los μ mejores individuos de entre los λ descendientes. Permite corregir la convergencia hacia mínimos locales.

EE simples: (1+1) EE

En las estrategias evolutivas simples solo hay un individuo en cada generación, no se realiza cruce, solo mutación. El descendiente reemplaza a su progenitor solo si es más apto que éste; en caso contrario la población permanece sin cambios. Cada individuo viene representado por dos cromosomas, un cromosoma se utiliza para representar el espacio de búsqueda y el otro incluye las desviaciones típicas que se aplican en una normal de media cero para obtener el peso de cada mutación. La mutación solo se realiza sobre el cromosoma del espacio de búsqueda, el segundo se adapta en función del éxito que haya tenido la mutación del primero.

Algoritmo general:

1. Crear un nuevo individuo:
    - Vector α inicializado de forma aleatoria. Codificación de una solución al problema.
    - Vector inicial de desviaciones típicas σ. Indica lo alejada que se encuentra la solución del óptimo.

2. Mientras no se cumpla el criterio de terminación.
3. Obtener vector de tasa de cambio aplicando una distribución normal: $z = N(0, \sigma)$.
4. Obtener el nuevo individuo: $y = \alpha + z$.
5. El individuo nuevo pasa a ser el actual si es mayor su fitness.
6. Modificar desviaciones típicas según la regla de *1/k.*
7. Volver al punto 3.

Mecanismo de auto-adaptación

Si el coeficiente de éxito de las k (habitualmente 5) últimas generaciones es próximo a 1/k, la desviación típica se deja como está; si es mayor se baja un poco y si es menor se sube un poco:

$$\begin{cases} \text{si } p > 1/5 \text{ entonces } \sigma = \frac{\sigma}{c} \\ \text{si } p < 1/5 \text{ entonces } \sigma = \sigma * c \\ \text{si } p = 1/5 \text{ entonces } \sigma = \sigma \end{cases}$$

siendo p el porcentaje de mutaciones exitosas y c un valor que varía normalmente entre 0.8 y 1.

El objetivo de este mecanismo es, por tanto, el de modificar las desviaciones típicas de forma que obtengan valores grandes cuando se esté lejos de la solución y valores pequeños cuando se esté cerca del óptimo.

Mutación

Este es el principal mecanismo evolutivo de las EE. Realiza un cambio en los valores de las variables de los individuos añadiendo ruido aleatorio obtenido mediante una distribución normal. Las desviaciones típicas a aplicar se encuentran dentro del genoma del individuo y co-evolucionan con la solución.

Cruce

Se obtiene un único individuo a partir de los padres. Se aplica variable a variable, bien realizando la media de los valores de los padres para esa variable, o bien seleccionando uno de los valores de los padres.

Técnicas de reemplazo

El criterio ($\lambda + \mu$) se corresponde con una estrategia elitista, habitualmente suele preferirse el criterio (λ , μ) ya que es menos propenso a caer en mínimos locales. En las EE la presión selectiva es muy alta por lo que normalmente $\lambda \sim 7 * \mu$.

2.3.1.1 CMA-ES

CMA-ES (Covariance Matrix Adaptation Evolution Estrategy) [9, 10, 11, 29] es un algoritmo evolutivo de optimización para problemas difíciles no-lineales, no-convexos en dominio continuo. Suele utilizarse para realizar búsquedas en una dimensión espacial de entre tres y cien variables. Es similar a los métodos cuasi-Newton, aunque no está inspirado en ellos.

CMA-ES estima una distribución de probabilidad de las mejores muestras para guiar la búsqueda en torno a las regiones más prometedoras del espacio de búsqueda. Supongamos que deseamos optimizar una función de fitness $f(x) : \mathbf{R}^p \rightarrow \mathbf{R}$, donde p es la dimensionalidad del problema. El algoritmo básico de CMA-ES sigue una búsqueda aleatoria de caja negra, tal y como se muestra:

- Inicializa los parámetros θ de la distribución.
- Para cada generación (iteración) t=0,1,2,…:
    1. λ puntos de muestra de la distribución: $P(x|\theta^{(t)}) = x1, \ldots, x\lambda$
    2. Evaluar el fitness de cada uno de los puntos.
    3. Actualizar los parámetros θ de la distribución según los mejores puntos, aquellos que obtuvieron mejor valor en la función de fitness.

En CMA-ES, la distribución de probabilidad a ser estimada es una distribución normal multivariante $N(m, \delta^2 C)$,cuyos parámetros θ son la media m y la matriz de covarianza $\delta 2\, C$. La media representa la localización actual de búsqueda y se moverá en torno a mejores localizaciones a medida que la búsqueda progrese. La matriz de covarianza controla las mutaciones y se usa para guiar la búsqueda.

Es necesario señalar que CMA-ES descompone la matriz de covarianza total en una matriz de covarianza C y la varianza global δ^2, llamado control de tamaño de paso (step-size control). Los diseñadores del algoritmo encontraron útil controlar el tamaño de los pasos de la mutación δ^2 independientemente de la dirección de búsqueda C. La razón es que estimar C requiere muchas muestras y, por tanto, lleva más tiempo ser adaptada, mientras δ^2 puede ser adaptada en marcos de tiempo más cortos. Lo que se persigue con esto es intentar reducir el número de generaciones para llegar a buenos individuos, mientras se evita caer en mínimos locales. Se controla el tamaño de los pasos comparándolo con el que tendrían si se eligiese un camino aleatorio. Si el camino es más corto significa que los pasos son muy largos y se producen ciclos, en cambio si el camino es más largo significa que se dan varios pasos en la misma dirección, por lo que podrían reducirse si los pasos fuesen más largos.

La media, el step-size y la matriz de covarianza se actualizan en cada generación t.

La implementación de CMA-ES usada en el proyecto [2]puede configurarse según diversos parámetros. Los parámetros por defecto más importantes son:

Restarts: Indica el número de reinicios del algoritmo cada vez que se alcance alguna de las condiciones de parada, aumentará en cada reinicio el tamaño de la población según un parámetro. El valor por defecto es cero.

StopOnStagnation: Se detiene el algoritmo si tras un gran número de evaluaciones no hay cambios significativos en el valor del fitness. El valor por defecto es desactivado.

DiagonalOnly: El algoritmo hace uso de una matriz de covarianzas diagonal durante un número determinado de generaciones que se le especifique. Se recomienda su uso para dominios con un número elevado de dimensiones, cercano o superior a 100.

StopFitness: Permite especificar un valor de fitness de parada para evitar más evaluaciones de las necesarias. Esto puede ser útil si, por ejemplo, el algoritmo está devolviendo valores fitness negativos, cuando realmente el mínimo posible fuese cero.

LBound, UBound: Cotas inferior y superior respectivamente, para los valores que puede tomar la variable buscada, por debajo o por encima de estas cotas se establece una penalización a la búsqueda de la solución.

DiffMinChange, DiffMaxChange: Cambio mínimo y máximo que se puede realizar a un valor de una generación a otra.

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

2.3.2 Algoritmos genéticos

Los algoritmos genéticos [12, 26, 27] fueron desarrollados por John H. Holland a principios de los años 60.

Este tipo de algoritmos evolutivos se apoyan fundamentalmente en el uso del operador de cruce, también utilizan la mutación como operador secundario. La selección de individuos es probabilística.

El esquema de ejecución típica de este tipo de algoritmos tiene la siguiente estructura general básica:

1. Generar aleatoriamente una población de individuos inicial
2. Evaluación de cada individuo de la población a partir de la función de fitness.
3. Finalizar si se ha encontrado la solución satisfactoria o pasar al punto cuatro.
4. Selección probabilística de los mejores individuos.
5. Aplicación de operadores genéticos (cruce, mutación...).
6. Evaluación de los nuevos individuos.
7. Selección probabilística de los nuevos individuos que formarán la nueva generación.
8. Volver al primer paso hasta ejecutar la cantidad de generaciones deseadas o encontrar un determinado individuo que satisfaga como solución .

Codificación de un individuo

La codificación más típica de los individuos de este tipo de algoritmos se realiza mediante una cadena binaria, esta cadena es denominada el cromosoma del individuo. El bloque de bits que codifica una variable (uno o varios bits) se le denomina gen y el valor de cada posición del cromosoma se le denomina alelo.

En la ilustración 3 puede verse un ejemplo de la codificación de un individuo:

0	1	1	0	1	1	0	1	0	0	1	1	1	0	0	1

Ilustración 3: Codificación individuo algoritmo genético

Estrategia de selección

La fase de selección se encargará de elegir de forma probabilística cuales son los individuos más aptos a partir de su valor de fitness. La selección se realiza de forma

probabilística debido a que no interesa seleccionar directamente a todos los mejores individuos, debe darse oportunidad a individuos que son peores pero que proporcionan variabilidad genética a la población. Esto permite que sea más complicado que la población acabe siendo homogénea en las primeras generaciones y que el algoritmo se estanque en mínimos locales.

Dos métodos de selección ampliamente conocidos son el método de la ruleta y la selección por torneo.

En la selección mediante la técnica de la ruleta a cada uno de los individuos de la población se le asigna una porción de una ruleta de forma proporcional a su valor de fitness, de forma que la suma de todas las porciones sea la unidad. De esta manera los individuos más aptos y por tanto con un mayor valor de fitness recibirán una mayor parte de la ruleta. Para realizar la selección de un individuo se genera un número aleatorio entre cero y uno, y se devuelve el individuo situado en esa porción de la ruleta. Este procedimiento deberá repetirse hasta completar el tamaño de la nueva población.

El método de la ruleta es bastante sencillo de implementar, sin embargo, su complejidad es de $O(n^2)$ a medida que aumenta el tamaño de población. Otro inconveniente de este método es que el peor individuo de la población puede ser seleccionado varias veces.

El método de selección por torneo se basa en realizar comparaciones directas entre individuos (Blickle y Thiele, 1995).

Una forma de realizar la selección por torneo es seleccionando al azar un número *n* (se fija como parámetro del algoritmo genético) de individuos. De entre los individuos elegidos para competir, se seleccionará el más apto (mayor valor fitness) para la nueva generación o población intermedia, habrá que realizar tantos torneos como individuos haya en la población. Variando el número de individuos que participan en el torneo se modifica la presión de selección, cuantos más participen mayor será la presión y los peores individuos tendrá menos posibilidades de pasar a la siguiente población.

Operación de cruce

Como ya se ha comentado con anterioridad, el cruce es la operación principal de estos algoritmos de búsqueda. Esta operación puede llevarse a cabo de varias formas, continuando con el tipo de codificación del ejemplo anterior algunos tipos de cruce más usados son los siguientes:

❖ Cruce de un punto:

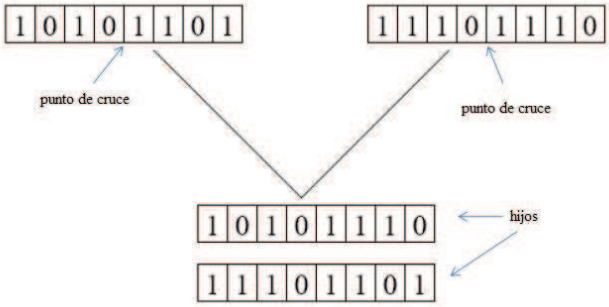

Ilustración 4: Cruce de un punto

❖ Cruce de dos puntos:

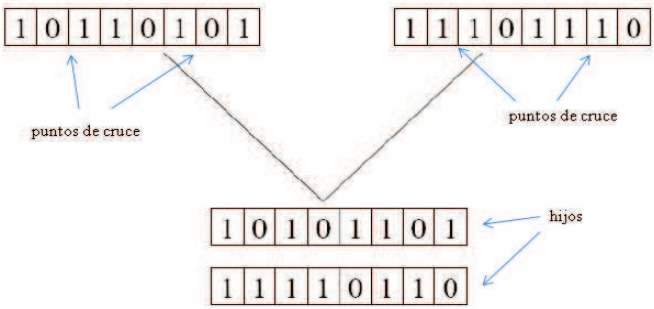

Ilustración 5: Cruce de dos puntos

❖ Cruce uniforme con probabilidad del 50%:

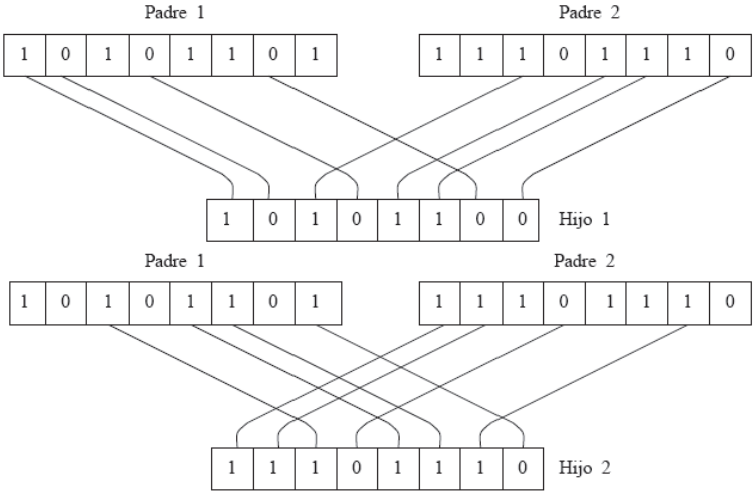

Ilustración 6: Cruce uniforme

Operación de mutación

La operación de mutación consiste en modificar directamente el valor de un gen con una determinada probabilidad. En caso de tener una codificación binaria simplemente bastaría con poner el valor contrario del bit, en caso por ejemplo de una codificación real podría generarse un número aleatorio que sustituyese al valor de la posición genética seleccionada probabilísticamente para mutar.

Elitismo

Otra técnica que suele emplearse muy a menudo en este tipo de algoritmos es el elitismo. Esta técnica permite que un tanto por ciento de los mejores individuos de una generación pasen directamente a la siguiente. Mediante el uso del elitismo se consigue conservar muy buenas soluciones que podrían perderse al efectuar selección, cruce y mutación.

2.3.2.1 Matlab genetic algorithm

Matlab cuenta con un algoritmo genético[3] mono-objetivo que ha sido utilizado a lo largo del presente trabajo.

Valores por defecto de los parámetros (estructura de opciones) del algoritmo genético '*ga*' que viene implementado en el toolbox de Matlab:

- Tamaño de población *'PopulationSize'*: El tamaño de la población por defecto será de 20 individuos.
- Representación de individuos *'PopulationType'*: Cada individuo será representado con un vector de números reales.
- Elitismo *'EliteCount'*: Los dos mejores individuos, con mejor fitness, de la generación actual pasarán directamente a la siguiente generación.
- Cruce *'Crossover Fraction'*: El número de individuos que serán sometidos a cruce serán el 80% de la población, obviando los individuos de élite, redondeando a número par. A los individuos restantes, tras hacer elitismo y cruce, se les aplicará mutación.
- Función de mutación por defecto *'MutationFcn'*: Para cada gen de los individuos a los que se les aplique mutación, se seleccionará aleatoriamente un número de una distribución Gaussiana. Se puede controlar además que la cantidad de mutación que se aplica sobre los individuos decrezca a medida que pasan las generaciones.
- Generaciones *'Generations'*: El número de generaciones del algoritmo será de 100 por defecto.

Por tanto, por defecto, tendremos una población de 20 individuos. De esos 20 individuos, los dos mejores pasarán directamente a la siguiente generación. Sobre los 18 individuos restantes se aplicará cruce al 80% redondeando, es decir, 0.8*18=14.4 → 14. Sobre los cuatro individuos que quedan se aplicará mutación a todos sus genes.

[3] http://www.mathworks.es/help/toolbox/gads/ga.html.

2.3.3 Optimización multiobjetivo

La mayor parte de los problemas de optimización del mundo real que se intentar abordar suelen tener una naturaleza multiobjetivo. Es decir, tienen más de una función objetivo que debe satisfacerse a la vez y que además, en gran número de ocasiones, estas funciones se encuentran en conflicto entre sí.

Este tipo de problemas suelen modelarse como problemas mono-objetivo. Esto puede hacerse fusionando todas las funciones en una sola, o bien puede usarse solo una de las funciones originales y utilizar el resto como restricciones.

En general, la optimización multi-objetivo [15, 16] no se restringe a la búsqueda de una sola solución sino que se buscan múltiples soluciones conocidas como soluciones no-dominadas. Cada una de las soluciones encontradas se denomina óptimo de Pareto, y la representación en el espacio de los valores de las funciones objetivo conforma el frente de Pareto. Por tanto, dado un problema de optimización multi-objetivo, la finalidad última es la de obtener el frente de Pareto, es decir, obtener un conjunto de puntos en los que no se puede mejorar un objetivo sin perjudicar al otro.

Podemos ver en la ilustración 7[4] una demostración del frente de Pareto, en un problema de minimización, donde cada estrella rosa (punto del frente) representa una solución del problema de búsqueda. Se puede observar que todas las soluciones son mejores que cualquiera de las otras en al menos uno de los objetivos y que mejorar en uno de los objetivos conlleva empeorar en el otro.

[4] http://www.mathworks.com/help/toolbox/gads/brin0vx.html

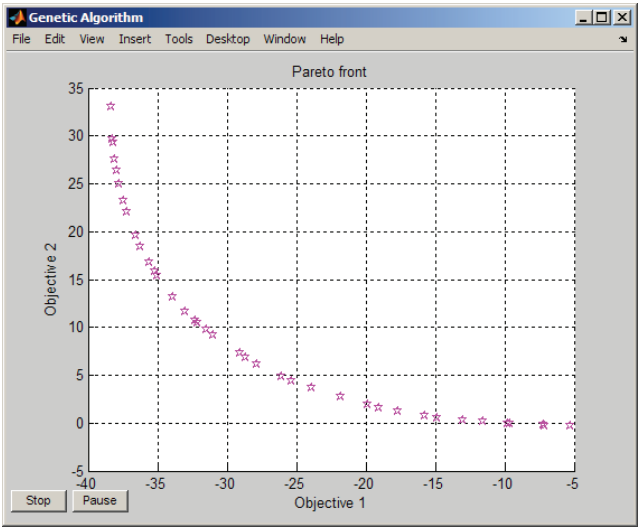

Ilustración 7: Frente de Pareto

Existen diversas técnicas para la obtención del frente de Pareto: enumerativas, deterministas y estocásticas. Los algoritmos evolutivos, en los cuales se centra el proyecto, se encuadran dentro de las técnicas estocásticas [16, 17, 18, 19]. Los métodos estocásticos no garantizan la obtención de la solución óptima pero ofrecen soluciones aceptables en un amplio rango de problemas de optimización en los que cuales los métodos deterministas encuentran dificultades. La búsqueda enumerativa es un método determinista en el cual no se emplea ninguna heurística, se trata de de una estrategia de optimización conceptualmente simple que está basada en la evaluación de cada posible solución dado un espacio de búsqueda finito. El principal inconveniente de la búsqueda enumerativa es su ineficiencia, puede llegar a ser muy costosa computacionalmente a medida que el espacio de búsqueda se incrementa.

A lo largo del proyecto se ha hecho uso del algoritmo evolutivo multiobjetivo[5], incluido en el toolbox de Matlab.

[5] http://www.mathworks.es/help/toolbox/gads/gamultiobj.html

2.4 Algoritmos de reducción de dimensionalidad

Estos algoritmos realizan, sobre un conjunto de datos, una recombinación de los atributos con el objetivo de obtener un menor número de ellos [20]. Esto puede suponer una ayuda importante en problemas de clasificación, además de poder usarse con el fin de visualizar en tres o dos dimensiones un determinado conjunto de datos con una dimensión superior.

Puede realizarse una clasificación de este tipo de algoritmos según diversos criterios, se exponen a continuación dos de ellos:

- Lineales vs no lineales: En función de que los nuevos atributos surjan a partir de combinaciones lineales de los atributos originales o no.
- Supervisados vs no supervisados: En función de que a la hora de realizar las combinaciones se tenga en cuenta o no la clase a la que pertenecen cada una de las instancias del conjunto de datos.

Algunos de los algoritmos lineales más usados son: análisis de componentes principales (PCA), análisis discriminante lineal (LDA), análisis de componentes independientes (ICA).

Entre los algoritmos no lineales más conocidos, que pueden usarse con el fin de reducir la dimensionalidad, tenemos: método de Sammon, mapas auto-organizativos de Kohonen (SOM), wavelets, extracción de características fractal, redes de neuronas, algoritmos genéticos.

En este trabajo se utiliza PCA y Sammon, que son explicados en detalle en los puntos siguientes.

2.4.1 Análisis de componentes principales (PCA)

Este método reduce la dimensionalidad a través de proyecciones lineales, estas proyecciones son elegidas de forma que la varianza total de las instancias en la nueva representación sea máxima [21]. Se trata de un método no supervisado debido a que no tiene en cuenta la clase a la que pertenecen las instancias del conjunto de datos. El objetivo de PCA es encontrar un nuevo conjunto de atributos no correlacionados linealmente y ordenados por la varianza explicada. El número de atributos se puede reducir descartando aquellos atributos que explican muy poca de la varianza de los datos.

El algoritmo funciona del siguiente modo: sean *N* instancias, en el conjunto de datos, representadas por los vectores de atributos $\{x_1, x_2,...,x_N\}$ tales que x_i pertenezca a R^n , consideremos una transformación lineal que nos transforme el espacio original *n* dimensional en *m* dimensional, siendo $m<n$. Las nuevas representaciones yi se calculan del siguiente modo:

$$y_i = W^T x_i \qquad i = 1,2,..., N$$

siendo las m columnas de $W \in R^{n\,x\,m}$ ortonormales.

Sea Sc la matriz de covarianzas, de los vectores originales, definida como:

$$S_c = \frac{1}{N} \sum_{i=1}^{N} (x_i - \mu)\ (x_i - \mu)^T$$

donde $\mu \in R^n$ representa la medida de los vectores originales. S_c tiene dimensión $n\ x\ n$ y es simétrica, por tanto tendrá *n* vectores propios $\{\phi_1,..., \phi_n\}$ y *n* valores propios $\{\lambda_1,..., \lambda_n\}$, de tal forma que cumplen la siguiente relación:

$$S_c\, \Phi = \Phi\, \Lambda$$

donde Φ es la matriz de vectores propios y Λ es la matriz de valores propios.

Los ϕ_i son ortonormales entre si, por tanto la matriz Φ define una transformación lineal ortogonal:

$$y_i = \Phi^T x_i$$

tras realizar esta transformación, los vectores y_i tienen como matriz de covarianzas:

$$\Phi^T S_c \Phi$$

dado que Φ es ortogonal ($\Phi^T \Phi = I$), la nueva matriz de covarianzas es Λ. Con esta transformación se consigue decorrelar los valores de las nuevas dimensiones. Además cada nueva dimensión, generada por un vector propio, tendrá como varianza el valor propio correspondiente.

Definiendo $W = [\phi_1,..., \phi_n]$, siendo ϕ_i los m vectores propios de mayor valor propio, se tiene el sub-espacio lineal de m dimensiones que más cantidad de la varianza original de los datos explica.

Matlab cuenta con una implementación de este algoritmo[6], que ha sido utilizado a lo largo de este proyecto.

2.4.2 Método de Sammon

Este es un algoritmo de mapeo no supervisado y además no lineal. Aplicando esta técnica, sobre un conjunto de datos n dimensional, se pasa a un espacio de dos o tres dimensiones.

El objetivo fundamental del método de Sammon [22] es el de conservar la estructura de los datos en el nuevo espacio, para ello intenta mantener las distancias existentes entre las instancias de datos en el espacio n-dimensional original.

Dado un conjunto de vectores n dimensionales $\{x_i: i=1,2...,N\}$, Sammon pretende pasar a un conjunto de vectores m dimensionales $\{y_i: i=1,2...,N\}$. Se define

$$d_{ij} = dist\ (x_i, x_j)$$

como la distancia entre dos puntos en el espacio original y

$$d^*_{ij} = dist\ (y_i, y_j)$$

como la distancia entre dos puntos en el espacio de destino. Tras inicializar aleatoriamente los vectores y_i podemos calcular la función de error E como:

$$E = \frac{1}{\sum_{i<j}(d^*_{ij})} \sum_{i<j} \frac{(d^*_{ij} - d_{ij})^2}{d^*_{ij}}$$

[6] http://www.mathworks.es/help/toolbox/stats/princomp.html

utilizando técnicas de descenso por gradiente se van modificando los valores de los vectores y_i hasta obtener un nivel de error aceptable.

Esta técnica tiene algunas desventajas:

- Debido a que usa técnicas de descenso por gradiente puede caer en mínimos locales.
- Coste computacional de $O(n^2)$.
- El mapeo es dependiente de los datos, un nuevo dato en el espacio original obliga a recalcular el algoritmo.

Matlab no cuenta en su toolbox con una implementación de este algoritmo, pero existe una implementación para Matlab[7] que ha sido utilizada en este proyecto.

[7] http://www.cis.hut.fi/somtoolbox/

2.5 Validación cruzada

Esta técnica, ampliamente utilizada en aprendizaje automático, nace de la necesidad de probar un algoritmo concreto sobre múltiples agrupaciones de un mismo conjunto de datos con el fin de evitar una mala o una muy buena ejecución aislada, es decir, evitar sesgos en las evaluaciones estadísticas de los algoritmos. De esta forma, pueden extraerse conclusiones estadísticas generales sobre el buen o mal funcionamiento de un algoritmo de aprendizaje automático sobre un determinado conjunto de datos.

El método de validación cruzada [23] utilizado en este proyecto es el comúnmente llamado K-folds. Este método consiste en dividir un conjunto completo de datos en K subconjuntos llamados *folds*. Se aplica el algoritmo de aprendizaje automático tantas veces como *folds* haya, de tal forma que en cada ejecución será un *fold* diferente el que asuma el papel de conjunto de test o validación y, el resto de *folds* formarán el conjunto de entrenamiento. Se obtendrá por tanto varios resultados de validación sobre los cuales se podrán aplicar técnicas de estadística descriptiva (media, desviación típica...) para obtener una conclusión general del funcionamiento del algoritmo de aprendizaje automático sobre el conjunto de datos tratado.

2.6 Implementación y experimentación ya existente

Para este trabajo se ha contado con una implementación y experimentación inicial existentes detalladas en [24]. En los puntos siguientes se comenta cual era el algoritmo de inicio sobre el cual se ha trabajado realizando modificaciones para la nueva experimentación, también se explican cuales son los dominios de datos utilizados a lo largo de la experimentación del proyecto y por último se exponen cuales eran los resultados de experimentación existentes y los problemas que se debían abordar en un inicio.

2.6.1 Algoritmo inicial

Se expone a continuación, en la tabla 1, un pseudo-código del algoritmo, implementado en Matlab, con el que se ha contado desde un inicio en el desarrollo de este trabajo. El objetivo de este algoritmo es utilizar CMA-ES para encontrar la amtriz de transformación óptima que maximiza el porcentaje de aciertos del algoritmo de clasificación k-NN (k=1).

```
1 inicio programa
2       desde 1 hasta 10 // bucle de ejecuciones externas
3               distribuir el conjunto de datos aleatoriamente en 10 grupos homogéneos
4               desde 1 hasta 10 // bucle de validación cruzada, folds externos
5                       asignar nuevos conjuntos de test y entrenamiento.
6                       matriz_evolucionada = CMA-ES(matriz_identidad, datos_train_original )
7                       datos_train_proyectados = datos_train_original * matriz_evolucionada
8                       datos_test_proyectados = datos_test_original * matriz_evolucionada
9                       knn(datos_train_original, datos_test_original)
10                      knn(datos_train_proyectados, datos_test_proyectados)
11                      acumular porcentaje de aciertos de test
12              fin
13              acumular porcentaje de aciertos de test
14      fin
15      media de los aciertos de test
16 fin programa
```

Tabla 1: Algoritmo de partida experimentación

Como puede observarse en el pseudo-código anterior, al inicio del bucle de ejecuciones externas, en la línea 3, se desordenan los datos de forma aleatoria distribuyéndolos en 10 grupos , cada uno de estos grupos contendrá aproximadamente la misma cantidad de instancias.

Dentro del bucle externo hay un bucle de validación cruzada en cual, en cada iteración, se tomará un distinto conjunto de test, es decir, una sola de las agrupaciones obtenidas anteriormente será el conjunto de test en cada iteración. Para el conjunto de entrenamiento se tomarán las nueve agrupaciones de datos restantes.

Dentro de cada iteración de validación cruzada se obtiene una matriz de proyección óptima obtenida mediante la estrategia evolutiva CMA-ES, esta matriz se usa para proyectar linealmente los datos tanto de entrenamiento como de test, y a partir de los datos proyectados obtener el porcentaje de aciertos en clasificación con k-NN (k=1) sobre el conjunto de test. Esto significa que en una ejecución completa del algoritmo se obtienen cien matrices óptimas con las cien proyecciones correspondientes.

Con los datos de todas las proyecciones se saca la media y la desviación típica de aciertos en test.

La función de aptitud o fitness utilizada por CMA-ES trata de minimizar el porcentaje de error en clasificación con k-NN (k=1). Para ello realiza un proceso de validación cruzada con los datos de entrenamiento proyectados con la matriz que se está evaluando. Esta validación cruzada es distinta de la validación cruzada de la línea 4 del algoritmo, esta última se utilizaba para calcular el porcentaje de aciertos en test, mientras que la validación cruzada que se usa en la función de validación es para calcular el fitness de cada individuo de CMA-ES.

Para realizar el procedimiento de validación cruzada K-folds, se agrupan los datos de entrenamiento en cinco grupos homogéneos. El resultado final de fitness es la media del porcentaje de fallos en clasificación de los cinco conjuntos.

2.6.2 Dominios de datos

Se han utilizado varios dominios de datos a lo largo del proyecto, sobre los cuales ya existe una experimentación previa en [24] con la que poder realizar comparaciones.

Estos dominios pueden dividirse en dos grupos, por un lado tenemos dominios artificiales y por otro lado dominios de datos correspondientes a problemas del mundo real. Todos los dominios pertenecen a problemas de clasificación y poseen atributos numéricos.

- Dominios de datos artificiales:
  - Ripley [25]: En este dominio cada muestra viene representada por dos atributos reales clasificados como '0' o '1'. Cada una de las clases se corresponde con una distribución bimodal, que es una composición equilibrada de dos distribuciones normales. Las matrices de covarianza son iguales para todas las distribuciones y los centros son diferentes. Una de las características que hace interesante este dominio es el gran solapamiento que existe entre las dos clases.
  - Aleatorio100 [24]: Este dominio consta de 4 atributos, x_1, x_2, x_3, x_4 y posee dos clases posibles para cada instancia. Los ejemplos se han proporcionado siguiendo una distribución uniforme en el intervalo [0, 1] para los atributos x_1 y x_2, y en el intervalo [0, 100] para los atributos x_3 y x_4. Si $x_1 < x_2$, el ejemplo se etiqueta dentro de la clase '1', en caso contrario se clasifica como clase '0', por tanto los atributos x3 y x4 son irrelevantes. Debido a que el rango de los atributos irrelevantes es mucho mayor, el porcentaje de acierto de k-NN es muy malo, en torno al 50%. El conjunto total de datos está compuesto por 300 ejemplos, 150 de cada clase.
  - Rectas45 [24]: Dominio de dos clases y con dos atributos. Fueron generados primero 100 ejemplos de la clase 1, localizados como intervalos regulares en línea recta pasando por el punto de origen (0,0) y formando un ángulo de 45 grados respecto al eje horizontal. La distancia entre dos puntos consecutivos es 1. Se generaron también 100 ejemplos de la clase 0 de la misma forma en una recta paralela pasando por el punto (0, -1), de forma que dado cualquier punto, el más cercano es de la clase opuesta. Además los puntos han sido perturbados añadiendo a cada coordenada un número aleatorio uniformemente distribuido en el intervado [-0.5, 0.5].

La idea de este dominios es que k-NN obtendrá muy malos resultados porque la mayor parte de las veces el vecino más cercano a un punto dado pertenece a la clase opuesta. Sin embargo, haciendo ciertas transformaciones de rotación y escalado de coordenadas deberían permitir que k-NN ofreciese un buen porcentaje de aciertos en clasificación.

En la siguiente ilustración podemos ver un subconjunto de puntos del dominio.

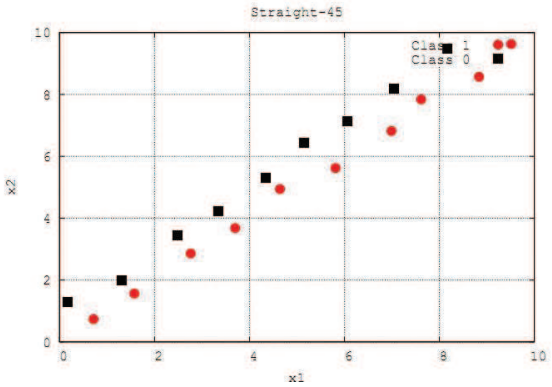

Ilustración 8: Dominio rectas-45

- Rectas0 [24]: Este dominio es similar a rectas45, la principal diferencia es que todos los puntos han sido rotados en sentido anti horario 45 grados. Por tanto, este dominio solo requiere una operación de escalado para poder obtener buen porcentaje de aciertos en clasificación.

En la siguiente figura podemos ver un subconjunto de puntos del dominio.

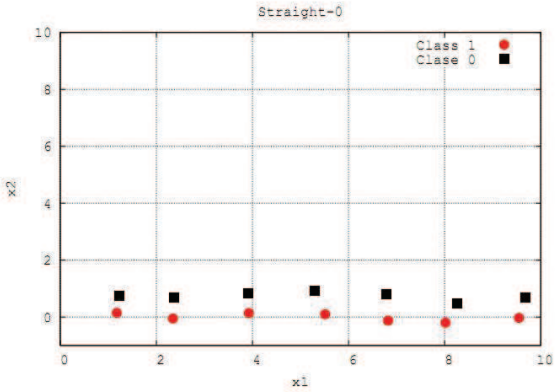

Ilustración 9: Dominio rectas-0

- Dominios de datos del mundo real:
  - Iris: Datos sobre plantas Iris. 150 instancias, 4 atributos y 3 clases.
  - Car: Automóviles de distintos tipos. 1728 instancias, 6 atributos, 4 clases.
  - Bupa: Problemas de hígado en hombres adultos. 345 instancias, 7 atributos, 2 clases.
  - Wine: Calidad sobre distintos vinos. 178 instancias, 13 atributos, 3 clases.
  - Ionosphere: Pulsos enviados mediante antenas de alta frecuencia en busca de estructuras reconocibles en la ionosfera. 351 instancias, 34 atributos, 2 clases.

Los dominios de datos del mundo real se han obtenido del UCI Machine Learning Repository [28]

2.6.3 Experimentación existente

En este punto se mostrarán los resultados obtenidos en [24] con la utilización del algoritmo expuesto en 2.6.1, sobre algunos de los dominios explicados en 2.6.2.

En la experimentación se utilizaron tanto matrices completas como matrices diagonales en la proyección de los datos originales. El número de filas y columnas de las matrices es igual a la dimensión del dominio de datos a proyectar en cada caso. En las experimentaciones con matrices completas, en todos los casos se partió desde la matriz identidad como matriz inicial en la búsqueda.

La utilización de matrices diagonales equivale a realizar un escalado de las coordenadas de datos originales. En el caso de las matrices completas no existen restricciones, realizándose una transformación lineal de los datos de forma general. Es importante recalcar que en el caso de las matrices completas siempre se ha proyectado los datos a la misma dimensión del espacio original.

Los parámetros de CMA-ES han sido establecidos de la siguiente forma en todas las experimentaciones: la desviación estándar inicial ha sido inicializada a 1.0, el número máximo de evaluaciones de fitness se ha establecido en 3000 y el resto de parámetros se han dejado por defecto.

En la siguiente tabla se muestran los resultados de experimentación obtenidos en cuanto a porcentaje de aciertos (media y desviación estándar) con la utilización de *k*-NN (k=1) haciendo uso de diez procesos de validación cruzada según se explica en el punto 2.6.1 del presente trabajo.

Dominios	Datos originales	CMA-ES Diagonal	CMA-ES Completa
Rectas-0	10.50 ± 1.99	**100.00** ± 0	99.70 ± 0.35
Rectas-45	8.45 ± 2.07	8.55 ± 2.05	**98.70** ± 0.54
Aleatorio100	50.03 ± 1.48	**95.33** ± 2.21	81.13 ± 4.62
Ripley	88.60 ± 0.49	**88.84** ± 0.35	88.43 ± 0.54
Car	87.47 ± 0.15	95.82 ± 0.30	**97.39** ± 0.25
Iris	95.87 ± 0.28	94.67 ± 0.99	**96.07** ± 0.80
Bupa	62.20 ± 1.29	60.52 ± 1.29	**65.33** ± 2.39
Wine	76.38 ± 1.42	**93.63** ± 0.9	85.61 ± 2.33

Tabla 2: Resultados experimentación existente

En la primera columna de la tabla 2 se muestra el dominio sobre el que se realizó la experimentación, en la segunda columna se muestran los resultados obtenidos sobre los datos originales sin proyectar, en las columnas tres y cuatro se muestran los resultados obtenidos en clasificación de los datos proyectados con matrices diagonales y completas evolucionadas con CMA-ES. Se muestran resaltados en negrita los mejores resultados obtenidos para cada dominio.

Para el dominio *rectas0* con una simple compresión de la coordenada x_1 pueden alcanzarse buenos resultados en clasificación ya que los puntos de la misma clase se acercan lo suficiente. La transformación puede conseguirse a través de la proyección lineal con una matriz diagonal o una completa. Como se puede observar en la tabla 2 de resultados, tanto con la proyección con matrices diagonales como completas se alcanzan excelentes resultados, 100% y 99.7% respectivamente. El ligeramente menor porcentaje de aciertos para la matriz completa es debido a que el número de parámetros de ajuste, en la evolución de la matriz, es mayor.

En el dominio *rectas45* se requiere una transformación más compleja de los datos para conseguir buenos aciertos en clasificación, no es suficiente con escalar las coordenadas si no que también hay que rotar los puntos. Esta transformación lineal no puede ser obtenida mediante una matriz diagonal. Puede observarse como se obtienen pésimos resultados en la clasificación con k-NN de los datos originales y de los datos proyectados mediante una matriz diagonal evolucionada, en torno al 8.5%. Sin embargo con una matriz completa se obtienen resultados cercanos al 100%.

El dominio *aleatorio100* consta de dos atributos irrelevantes cuyo rango numérico es mucho más amplio que el rango de los dos atributos relevantes en clasificación. Esta es la razón por la que k-NN ofrece un porcentaje del 50% en cuanto a la clasificación de los datos originales. Escalando los atributos irrelevantes debería ser suficiente para obtener un porcentaje de aciertos mayor, por tanto, tanto las matrices diagonales como completas deberían ofrecer buenas soluciones. Sin embargo puede observarse que el porcentaje de aciertos de las matrices diagonales ronda el 95% mientras que para las matrices completas ronda el 81%, esto es debido al mayor número de elementos que deben ajustarse en las matrices completas.

En el dominio *wine* se observan unos resultados similares a los ocurridos con *aleatorio100*, existiendo diferencias significativas entre los resultados obtenidos con matrices diagonales y completas, siendo los de las primeras bastante mejores.

Sobre los dominios *iris*, *bupa y ripley* se consigue una ligera mejora al proyectar los datos tanto con matrices diagonales como completas aunque la mejora no es significativa.

Por último, sobre el dominio *car* se consigue una mejora significativa en clasificación proyectando los datos, especialmente con matrices completas.

También existe en [24] experimentación con la utilización de un algoritmo de búsqueda alternativo llamado Local Search (LS) que no ha sido considerado en el presente trabajo por no resultar relevante respecto a CMA-ES.

Capítulo 3

Experimentación

3.1 Estancamiento con matrices completas

El punto de partida de la experimentación se sitúa en torno a los problemas de estancamiento producidos en los dominios *wine* y *aleatorio100* para matrices completas, partiendo desde la identidad, y usando el algoritmo de búsqueda CMA-ES. La cuestión es que al partir de una matriz diagonal no se produce este estancamiento. En la tabla resumen 3 puede observarse la problemática, según los experimentos expuestos en [24] y explicados en el punto 2.6.3 del presente trabajo.

Dominio	Datos originales	CMA-ES Diagonal	CMA-ES Completa
Aleatorio100	50.03 ± 1.48	95.33 ± 2.21	81.13 ± 4.62
Wine	76.38 ± 1.42	93.63 ± 0.9	85.61 ± 2.33

Tabla 3: Estancamiento CMA-ES completa

Como puede observarse en la tabla 3, las matrices completas no consiguen aproximarse a los buenos resultados obtenidos con las matrices diagonales. Esto puede ser debido fundamentalmente a que las matrices completas constan de un mayor número de variables que debe considerar el algoritmo a la hora de buscar una buena solución.

Esta problemática se ha abordado realizando tres tipos de experimentación diferente: variando la matriz de inicio, cambiando el algoritmo de búsqueda y utilizando matrices simétricas. El algoritmo de clasificación k-NN se configuró con k=1 para todas las experimentaciones de los puntos siguientes.

3.1.1 Variación matriz inicial

Lo primero que se hizo fue modificar la matriz inicial de partida para el caso de CMA-ES completa. La matriz de partida que se había probado había sido siempre la matriz identidad, por tanto se realizó la prueba partiendo de una matriz completa aleatoria con valores en sus celdas comprendidos entre -10 y 10. Los valores de configuración de CMA-ES fueron los mismos: desviación inicial: 1.0 y número de evaluaciones: 3000. Los resultados obtenidos se muestran en la tabla 4.

Dominio	Datos originales	CMA-ES completa identidad	CMA-ES diagonal	CMA-ES completa aleatoria
Aleatorio100	50.03 ± 1.48	81.13 ± 4.62	95.33 ± 2.21	**78.76 ± 4.22**
Wine	76.38 ± 1.42	85.61 ± 2.33	93.63 ± 0.9	**85.50 ± 2.43**
Rectas45	8.45 ± 2.07	98.70 ± 0.54	8.55 ± 2.05	**98.85 ± 0.5**

Tabla 4: Variación matriz inicial completa CMA-ES

Se encuentra resaltada en negrita la última columna, que contiene los nuevos resultados de experimentación obtenidos en este punto.

Como puede observarse en la tabla 4, los resultados obtenidos partiendo de una matriz completa aleatoria con CMA-ES no son satisfactorios, son similares a los obtenidos partiendo desde la matriz identidad.

Es interesante además comentar los tiempos de ejecución aproximados del algoritmo sobre los dominios *wine* y *aleatorio100*. Para el dominio *aleatorio100* el algoritmo ha tardado en ejecutarse ~ 88 minutos. Para el dominio *wine* el algoritmo ha tardado en ejecutarse ~ 77 minutos.

3.1.2 Método de búsqueda alternativo: Algoritmo genético

En este punto se muestran los resultados obtenidos al aplicar un algoritmo genético en la búsqueda de matrices de proyección para los problemas de clasificación. Hasta el momento solo se había aplicado el algoritmo de búsqueda CMA-ES.

El motivo de la utilización de algoritmos genéticos es que, en principio, la búsqueda realizada puede explotar con mayor eficacia la aparición de buenos individuos en la población al realizar cruzamiento entre ellos.

La función de fitness utilizada en todos los experimentos de este punto es la misma, la que ya había implementada para CMA-ES.

El algoritmo genético mono-objetivo utilizado ha sido el disponible en el toolbox de matlab 7.8.0, explicado en el punto 2.3.2.1 del presente documento.

Para poder realizar una comparación entre ambos algoritmos de búsqueda, CMA-ES vs GA (algoritmo genético), se han establecido los parámetros del genético de forma equivalente a los establecidos en CMA-ES. El número de evaluaciones de CMA-ES fue de 3000 por lo que en el genético el tamaño de población multiplicado por el número de generaciones deberá ser de 3000.

En la tabla siguiente se muestran las dos configuraciones para las cuales se ha aplicado el algoritmo genético:

	Parámetros
Configuración 1	❖ Población inicial: 30. ❖ Generaciones: 100. ❖ Valores de cada individuo inicial (vector): aleatorios en el intervalo [-10, 10]. ❖ Resto de parámetros por defecto.
Configuración 2	❖ Población inicial: 60. ❖ Generaciones: 50. ❖ Valores de cada individuo inicial (vector): aleatorios en el intervalo [-10, 10]. ❖ Resto de parámetros por defecto.

Tabla 5: Configuraciones algoritmo genético

Los dominios para los cuales se ha aplicado el genético son los mismos que en el punto 3.1.1 (*aleatorio100, wine* y *rectas45*), el motivo es exactamente el mismo, se quiere averiguar porque se atascan los algoritmos de búsqueda para matrices de proyección completas.

Se muestra en la tabla 6 los resultados con los porcentajes de aciertos en clasificación con los datos originales, los datos proyectados con CMA-ES [24] y los resultados de la nueva experimentación con GA:

Dominio	Datos originales	CMA-ES diagonal	CMA-ES completa	GA configuración 1	GA configuración 2
Aleatorio100	50.03 ± 1.48	95.33 ± 2.21	81.13 ± 4.62	**85.26 ± 4.07**	**86.96 ± 3.37**
Wine	76.38 ± 1.42	93.63 ± 0.9	85.61 ± 2.33	**87.13 ± 2.3**	**87.13 ± 3.43**
Rectas45	8.45 ± 2.07	8.55 ± 2.05	98.70 ± 0.54	**99.40 ± 1.4**	**99.30 ± 0.8**

Tabla 6: Resultados algoritmo genético

Se encuentran resaltadas en negrita las dos últimas columnas, que contienen los nuevos resultados de experimentación obtenidos en este punto.

Como puede observarse en la tabla 6, el algoritmo genético en sus dos configuraciones consigue superar ligeramente a CMA-ES completa para los tres dominios. Sin embargo sigue quedándose lejos de los resultados obtenidos con CMA-ES diagonal. Algo importante es que queda probado que el genético no se queda estancado en el dominio de control *rectas45*. A priori parece que la búsqueda con el genético es algo más global que la de CMA-ES y por tanto menos propensa a estancarse en mínimos locales.

Para el dominio *aleatorio100* el algoritmo ha tardado en ejecutarse ~ 85 minutos, tiempo similar al conseguido con CMA-ES. Para el dominio *wine* el algoritmo ha tardado en ejecutarse ~ 68 minutos que es un valor algo menor que el conseguido con CMA-ES sobre este mismo dominio.

Por último, comentar dentro de este punto que también se ha realizado una ejecución con el dominio *aleatorio100* dándole un mayor margen de búsqueda al genético, para comprobar si se queda o no estancado en mínimos locales, estableciendo los parámetros de la siguiente forma:

- Población inicial: 100.
- Generaciones: 500.
- Valores de cada individuo inicial (vector): aleatorios en el intervalo [-10, 10].
- Resto de parámetros por defecto.

El resultado obtenido en esta ejecución es de: **92.46 ± 2.8**. Este resultado se acerca considerablemente al obtenido con CMA-ES diagonal y está bastante por encima del obtenido con CMA-ES completa. Aunque el resultado es bueno y vemos que el genético no se queda atascado, la carga computacional es demasiado grande y esta misma configuración no sería viable con dominios reales con un mayor número de dimensiones. Recordemos que el dominio *aleatorio100* es un dominio artificial de tan solo cuatro dimensiones y cien instancias.

3.1.3 Matrices simétricas con CMAES

Se ha experimentado utilizando matrices simétricas sobre los dominios: *wine, aleatorio100 y rectas45*, para comprobar si la búsqueda con CMAES se sigue atascando como pasaba con las matrices completas aleatorias.

Antes de pasar a comentar la experimentación es importante aclarar brevemente en qué consisten las matrices simétricas Toeplitz que no son más que un caso particular de las matrices de Toeplizt generales.

Las matrices de Toeplitz generales son matrices cuadradas con la siguiente forma:

$$\begin{pmatrix} a & b & c & d & k \\ f & a & b & c & d \\ g & f & a & b & c \\ h & g & f & a & b \\ j & h & g & f & a \end{pmatrix}$$

Ilustración 10: Matriz Toeplitz

Por tanto, una matriz de Toeplitz general puede determinarse a partir de la primera fila y la primera columna, es decir, de dos vectores dados.

Las matrices simétricas de Toeplitz pueden determinarse a partir únicamente de la primera fila, es decir, de un vector dado. Tienen la siguiente forma:

$$\begin{pmatrix} A & B & C & D & E \\ B & A & B & C & D \\ C & B & A & B & C \\ D & C & B & A & B \\ E & D & C & B & A \end{pmatrix}$$

Ilustración 11: Matriz Toeplitz simétrica

Los tipos de matrices simétricas utilizadas en este bloque de experimentación han sido dos:

- **Matrices simétricas de Toeplitz (diagonal constante):** Matlab ya cuenta con un comando que permite formar estas matrices fácilmente a partir de un vector inicial dado. En este caso, se le ha proporcionado un vector inicial aleatorio (con

dimensión según el dominio de datos) con valores iniciales comprendidos en el intervalo [-10. 10].

- **Matrices simétricas normales (diagonal no constante):** Para la creación de estas matrices y del vector inicial he tenido que programarlo desde cero. El vector inicial aleatorio (con valores iniciales comprendidos en el intervalo [-10. 10]) el cual se le pasará a CMAES tiene el siguiente tamaño:

  *(Dimension*2-1)+numerosExtra*

  donde *(Dimension*2-1)* define la diagonal de la matriz y la primera fila de la misma, y *numerosExtra* define el resto de números, por encima de la diagonal principal que no corresponden a la primera fila. En la función de fitness, al recibir este vector, se construye la matriz simétrica para realizar cada evaluación. También se reconstruye la matriz simétrica al finalizar CMAES, para devolver la matriz mejor.

El motivo principal de probar con las matrices simétricas de Toeplitz respecto a las normales es que el vector que necesita manejar el algoritmo CMAES es considerablemente menor, por lo que el coste computacional será mucho más pequeño.

La configuración de CMA-ES para los dos casos ha sido la misma que se viene utilizando, 3000 evaluaciones y desviación estándar inicial 1.0, para poder realizar la comparación con la experimentación existente.

Sobre cada dominio, con cada tipo de matriz simétrica, se han hecho una ejecución del algoritmo descrito en 2.6.1, 10 validaciones cruzadas.

Una vez descritos los tipos de matrices utilizados en este bloque , se muestran los resultados obtenidos en la tabla 7.

Dominio	Datos originales	Completa CMA-ES	Diagonal CMA-ES	Toeplitz CMA-ES	Simétrica normal CMA-ES
Rectas45	8.45 ± 2.07	98.70 ± 0.54	8.55 ± 2.05	**92.85± 4.65**	**20.45± 9.92**
Aleatorio100	50.03 ± 1.48	81.13 ± 4.62	95.33 ± 2.21	**51.63± 1.95**	**50.56± 3.76**
Wine	76.38 ± 1.42	85.61 ± 2.33	93.63 ± 0.9	**74.21± 1.69**	**78.53± 2.6**

Tabla 7: Resultados matrices simétricas

Se encuentra resaltadas en negrita las dos últimas columnas, que contienen los nuevos resultados de experimentación obtenidos en este punto.

Como puede observarse los resultados no son buenos, se quedan muy lejos tanto de las matrices completas como de las matrices diagonales. Para el caso de las matrices simétricas normales el algoritmo no consigue mejorar ni si quiera los datos originales sin proyectar. Para el caso de las matrices simétricas Toeplitz, el único dominio en el que no se atasca es en el de control, *rectas45*.

Como conclusión final al problema de estancamiento, no se ha conseguido solucionar el problema con un número de evaluaciones razonable por lo que no parece tratarse de un problema en la inicialización de la matriz ni en el método de búsqueda utilizado.

3.2 Proyección a dos dimensiones

La utilización de matrices de proyección evolucionadas puede usarse como un método supervisado para la reducción de la dimensionalidad de un conjunto de datos con la intención de poder visualizarlos.

En esta experimentación se han proyectado a dos dimensiones cada uno de los dominios. La proyección a dos dimensiones resulta de gran interés ya que permite la representación gráfica de los dominios proyectados, permitiendo de esta forma una visualización de los mismos. El objetivo por tanto será el de encontrar matrices de proyección a dos dimensiones de tal forma que una vez proyectados los datos se aumente el número de aciertos en la clasificación.

En esta primera parte de la experimentación se ha lanzado el algoritmo completo de diez validaciones cruzadas, descrito en 2.6.1, con la intención de observar que tal funciona de forma exhaustiva la búsqueda con CMA-ES de buenas matrices de proyección a dos dimensiones.

El algoritmo evolutivo utilizado para obtener las matrices de proyección ha sido CMA-ES con 3000 evaluaciones y desviación estándar inicial de 1.0, es decir, los mismos parámetros usados en experimentos anteriores con el motivo de poder realizar una comparativa. El algoritmo de clasificación k-NN se configuró con k=1 para todas las experimentaciones de los puntos siguientes.

Al proyectar a dos dimensiones, las matrices no podían ser cuadradas para la mayoría de los dominios (al tener solo dos columnas), por lo que no había matriz identidad de partida para la mayoría de los casos.

Se han utilizado matrices completas y se ha experimentado partiendo desde matrices aleatorias con valores comprendidos entre -10 y 10.

En la tabla siguiente se muestran los resultados obtenidos sobre cada uno de los dominios.

Dominio	Datos originales	CMA-ES Diagonal	CMA-ES Completa Dimensión original	CMA-ES Completa Dos dimensiones
Rectas-0	10.50 ± 1.99	100.00 ± 0	99.70 ± 0.35	**99.80 ±0.45**
Rectas-45	8.45 ± 2.07	8.55 ± 2.05	98.70 ± 0.54	**98.70 ± 0.5**
Aleatorio100	50.03 ± 1.48	95.33 ± 2.21	81.13 ± 4.62	**70.76± 6.94**
Ripley	88.60 ± 0.49	88.84 ± 0.35	88.43 ± 0.54	**88.68 ± 0.45**
Car	87.47 ± 0.15	95.82 ± 0.30	97.39 ± 0.25	**96.77 ± 0.74**
Iris	95.87 ± 0.28	94.67 ± 0.99	96.07 ± 0.80	**95.60 ± 0.95**
Bupa	62.20 ± 1.29	60.52 ± 1.29	65.33 ± 2.39	**60.11 ± 2.43**
Wine	76.38 ± 1.42	93.63 ± 0.9	85.61 ± 2.33	**87.64 ± 1.99**

Tabla 8: Resultados proyección 2D CMA-ES

Se encuentra resaltada en negrita la última columna, que contiene los nuevos resultados de experimentación a dos dimensiones obtenidos en este punto.

Como se puede observar en la tabla, la proyección a dos dimensiones con CMA-ES ofrece unos resultados similares a los obtenidos con CMA-ES completa proyectando a la misma dimensión que tenían los datos originales [24] . Solo en el caso del dominio *aleatorio100* los resultados son especialmente peores en la proyección a dos dimensiones.

3.2.1 Representación gráfica de las proyecciones

Para cada uno de los dominios anteriores, se ha evolucionado una matriz completa con CMA-ES para realizar la proyección y visualización de los datos. Para la visualización se ha hecho uso de la función de Matlab *gscatter*[8]. En las gráficas de proyección se han representado tanto los datos del conjunto de entrenamiento como los datos del conjunto de test, además todos los datos aparecen clasificados por colores según el grupo al que pertenezcan.

También se han representado gráficamente los datos de los distintos dominios haciendo uso de otros dos métodos de visualización: PCA (explicado en 2.4.1) y Sammon (explicado en 2.4.2). Esto permite comparar los resultados ofrecidos con dos algoritmos ampliamente usados con la nueva experimentación obtenida con la evolución de matrices de proyección en problemas de visualización.

Para el caso de los dominios *rectas0* y *rectas45* no se ha aplicado el algoritmo de Sammon debido a que estos dominios tienen dimensión dos. Sammon lo que pretende es conservar la estructura de los datos (distancias euclídeas) de la dimensión original en la de destino, y en este caso ambas dimensiones coinciden, por lo que no tiene sentido utilizarlo.

[8] http://www.mathworks.es/help/toolbox/stats/gscatter.html

Dominio rectas0:

Proyección mediante una matriz óptima obtenida con CMA-ES:

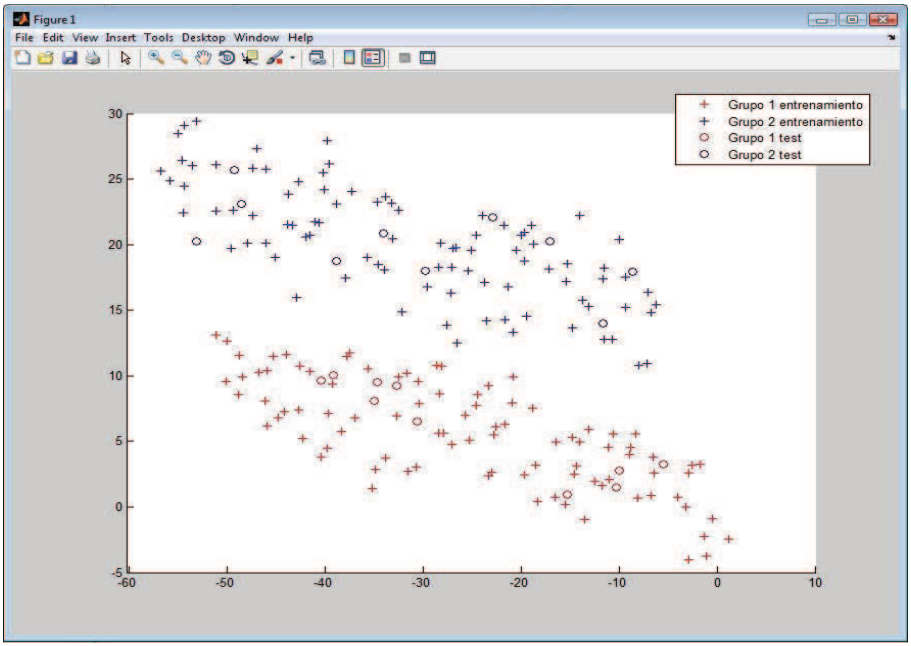

Ilustración 12: Rectas0 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 5%.
- Tasa de aciertos para datos proyectados: 100%.

Proyección obtenida mediante PCA:

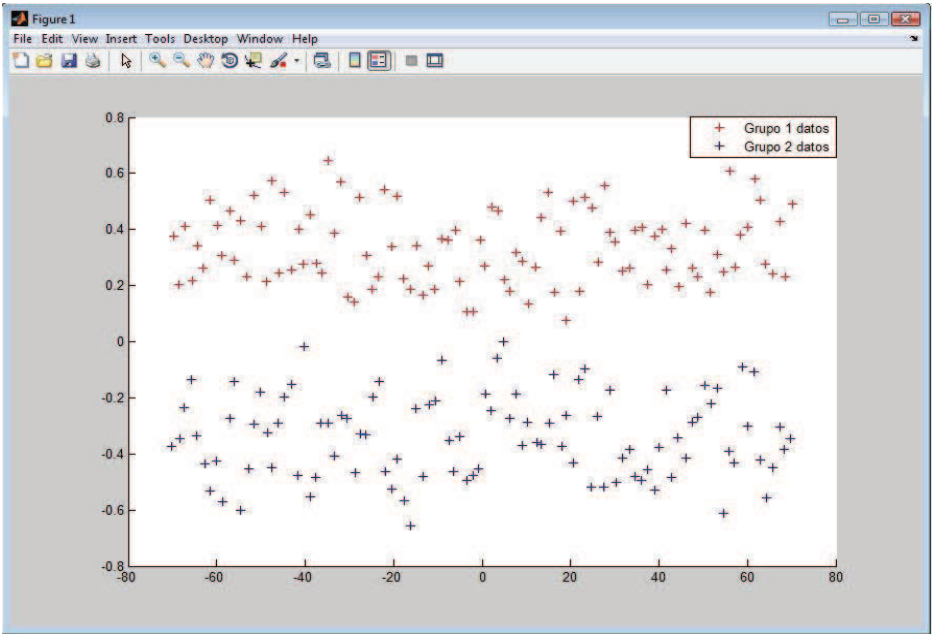

Ilustración 13: Rectas0 2D PCA

Dominio rectas45:

Proyección mediante una matriz óptima obtenida con CMA-ES:

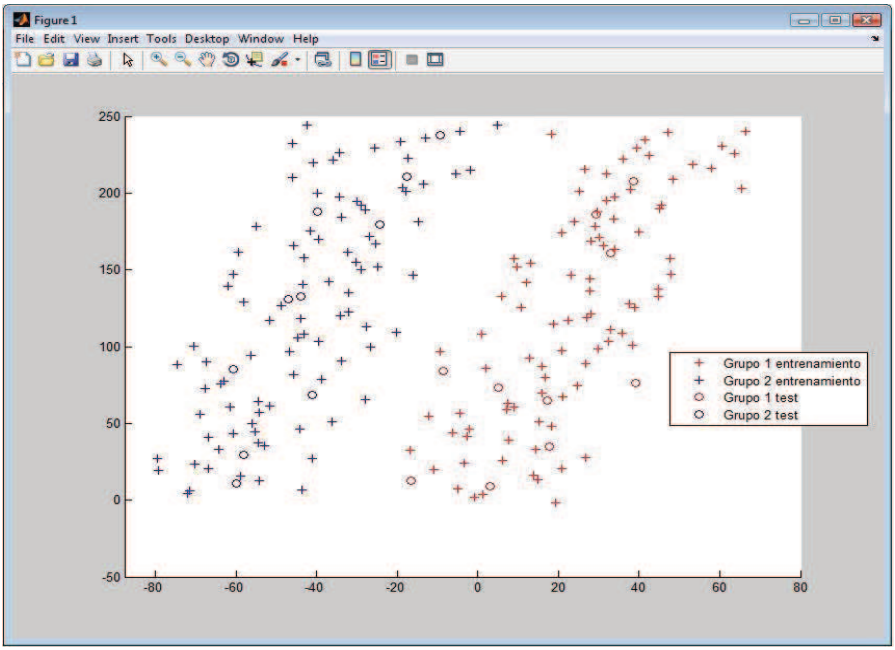

Ilustración 14: Rectas45 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 20%.
- Tasa de aciertos para datos proyectados: 100%.

Proyección obtenida mediante PCA:

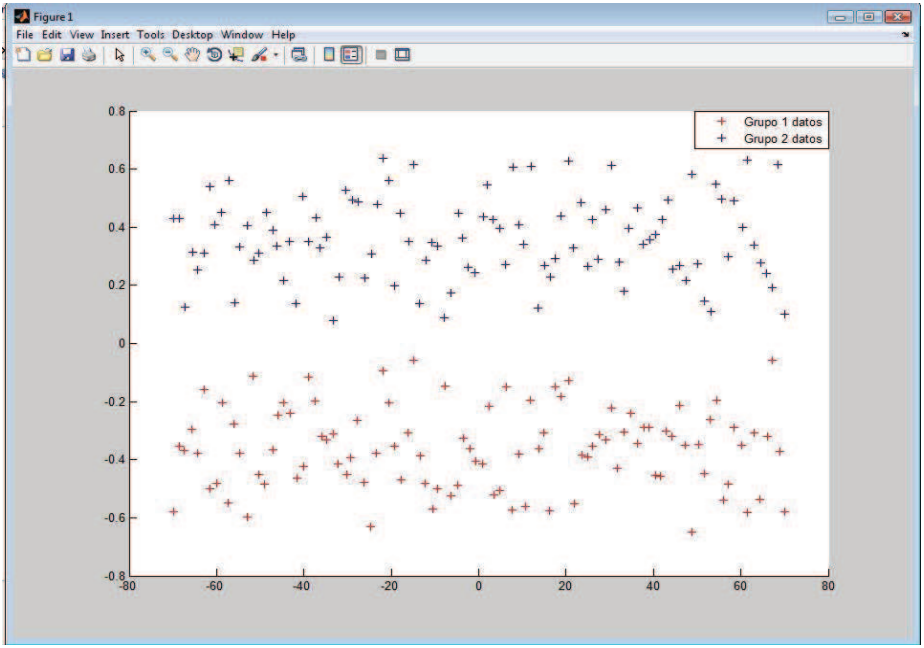

Ilustración 15: Rectas45 2D CMA-ES

Dominio bupa:

Proyección mediante una matriz óptima obtenida con CMA-ES:

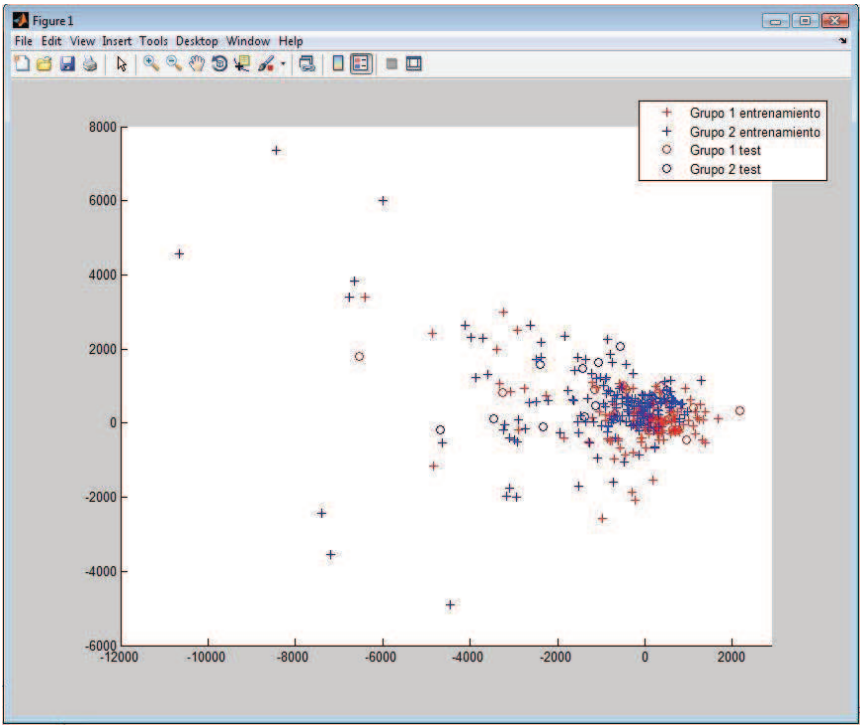

Ilustración 16: Bupa 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar:54,29%.
- Tasa de aciertos para datos proyectados: 65,71%.

Proyección obtenida mediante PCA:

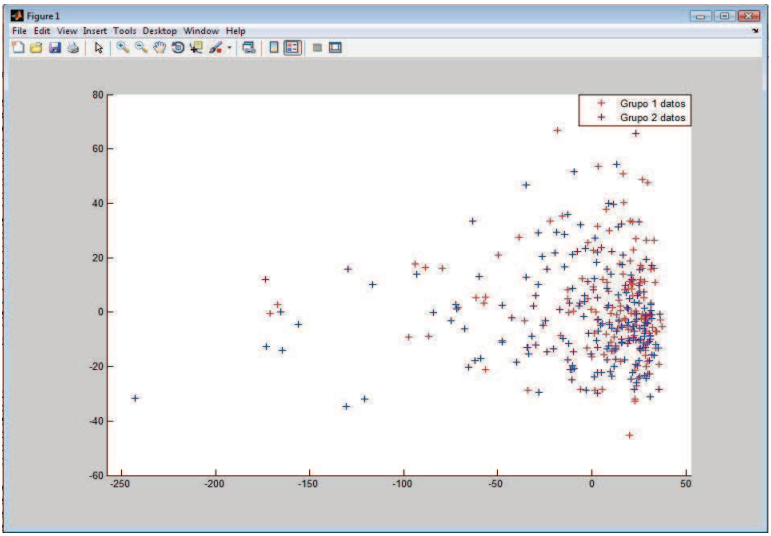

Ilustración 17: Bupa 2D PCA

Proyección obtenida mediante SAMMON:

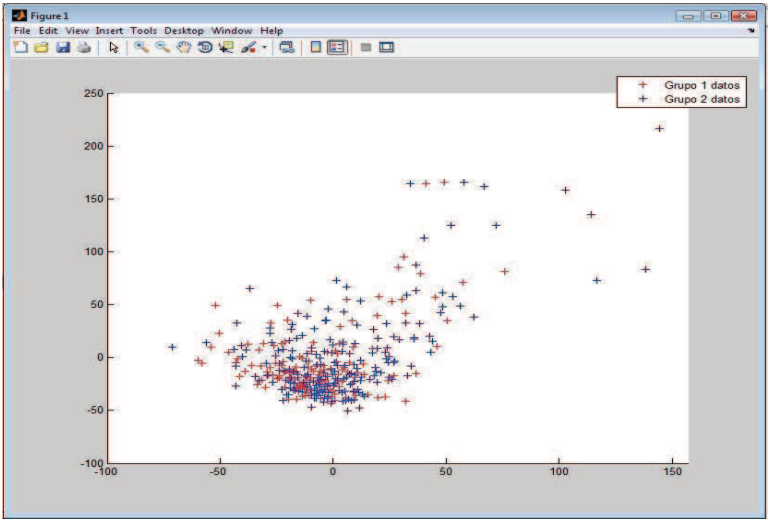

Ilustración 18: Bupa 2D Sammon

Domino car:

Proyección mediante una matriz óptima obtenida con CMA-ES:

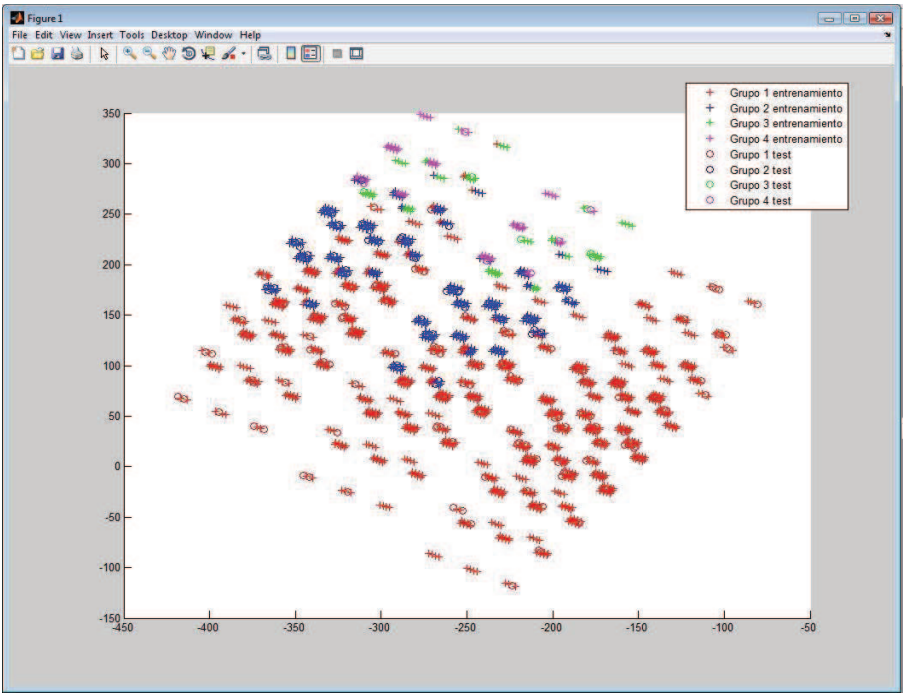

Ilustración 19: Car 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 88,37%.
- Tasa de aciertos para datos proyectados: 92,44%.

Proyección obtenida mediante PCA:

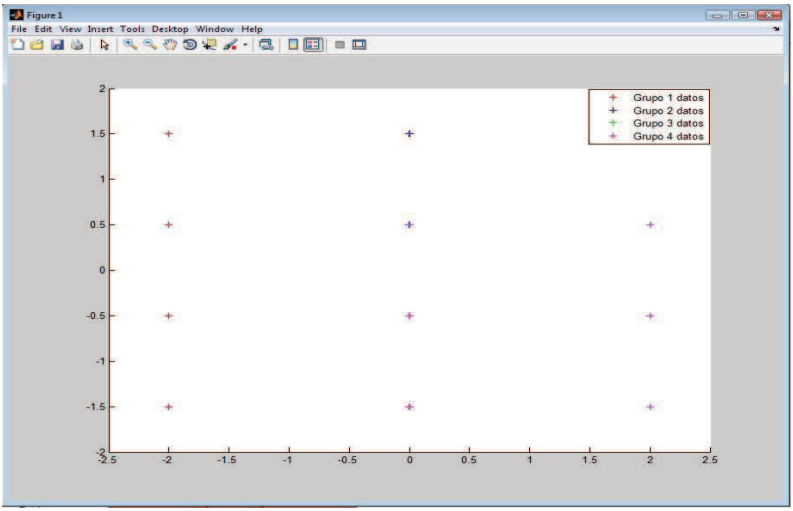

Ilustración 20: Car 2D PCA

Proyección obtenida mediante SAMMON:

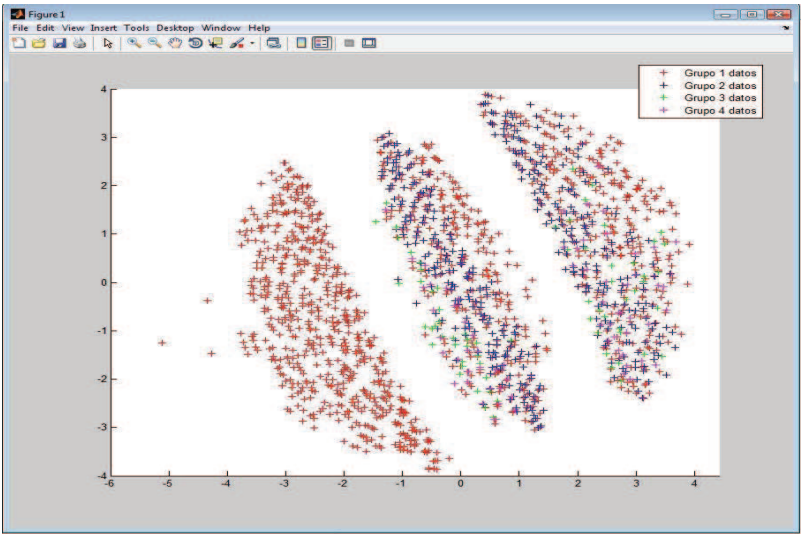

Ilustración 21: Car 2D Sammon

Dominio iris:

Proyección mediante una matriz óptima obtenida con CMA-ES:

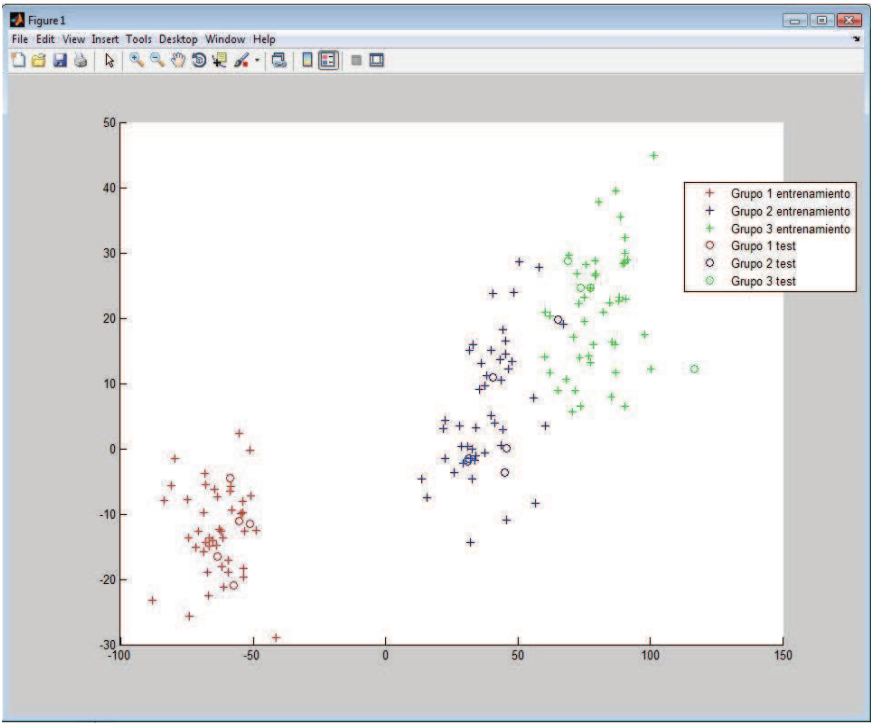

Ilustración 22: Iris 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 100%.
- Tasa de aciertos para datos proyectados: 93,33%.

Proyección obtenida mediante PCA:

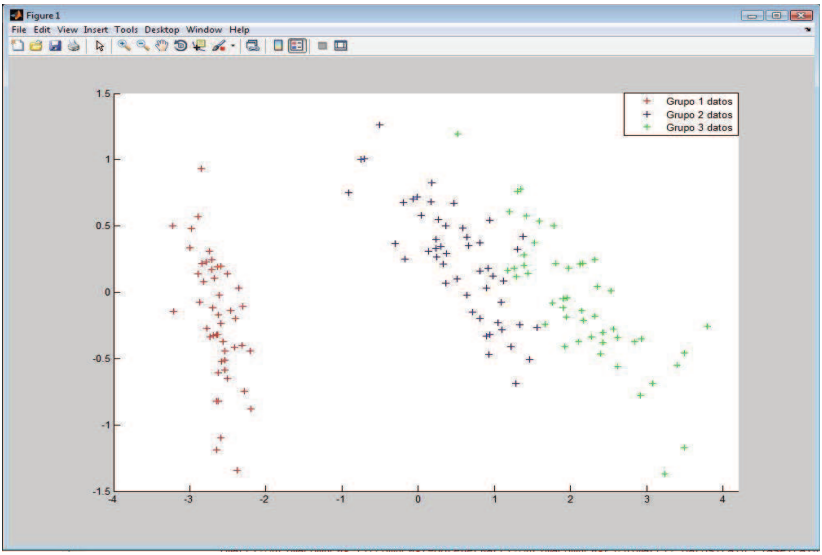

Ilustración 23: Iris 2D PCA

Proyección obtenida mediante SAMMON:

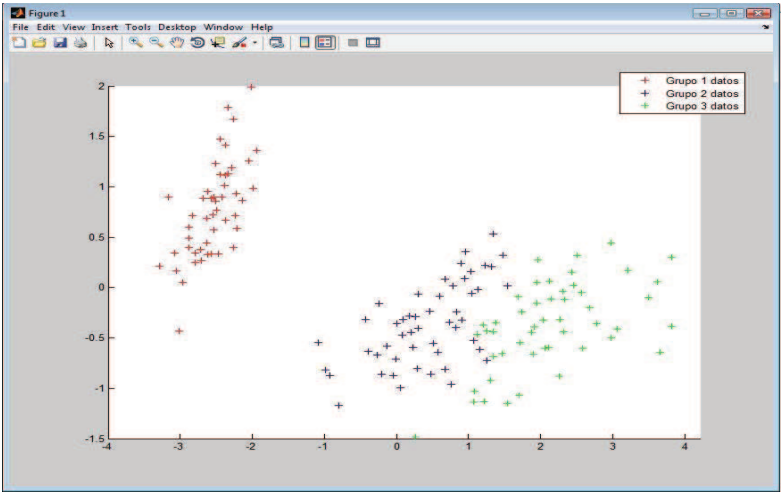

Ilustración 24: Iris 2D Sammon

Dominio ripley:

Proyección mediante una matriz óptima obtenida con CMA-ES:

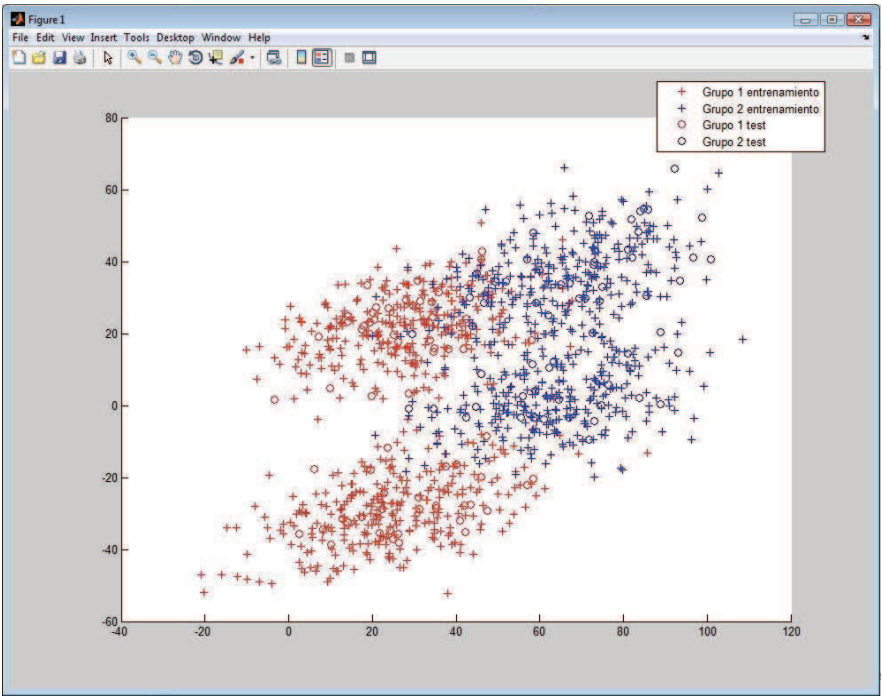

Ilustración 25: Ripley 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 88,10%.
- Tasa de aciertos para datos proyectados: 89,68%.

Proyección obtenida mediante PCA:

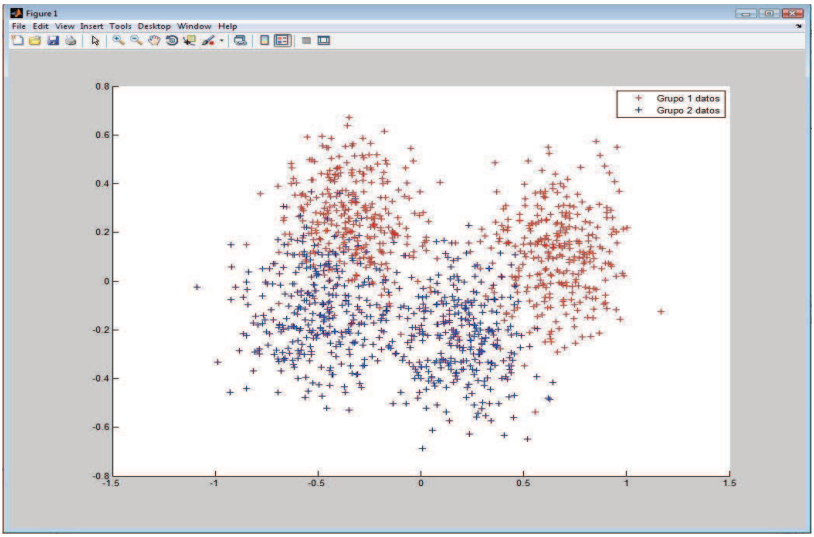

Ilustración 26: Ripley 2D PCA

Proyección obtenida mediante SAMMON:

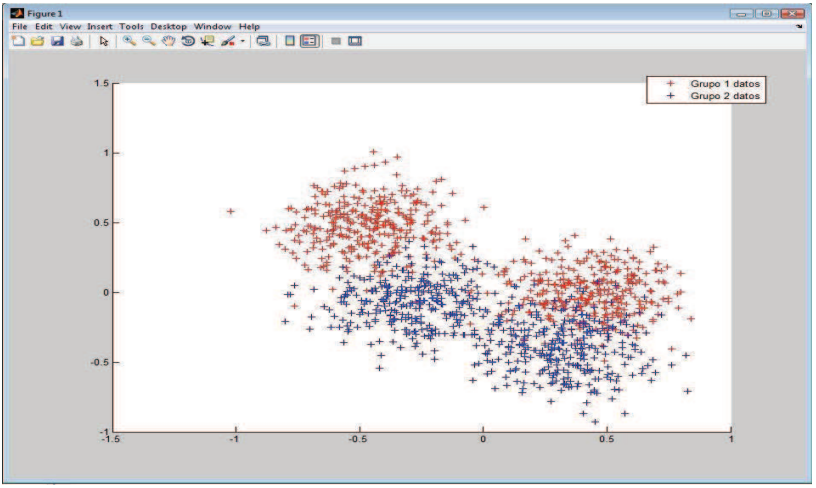

Ilustración 27: Ripley 2D Sammon

Dominio aleatorio100:

Proyección mediante una matriz óptima obtenida con CMA-ES:

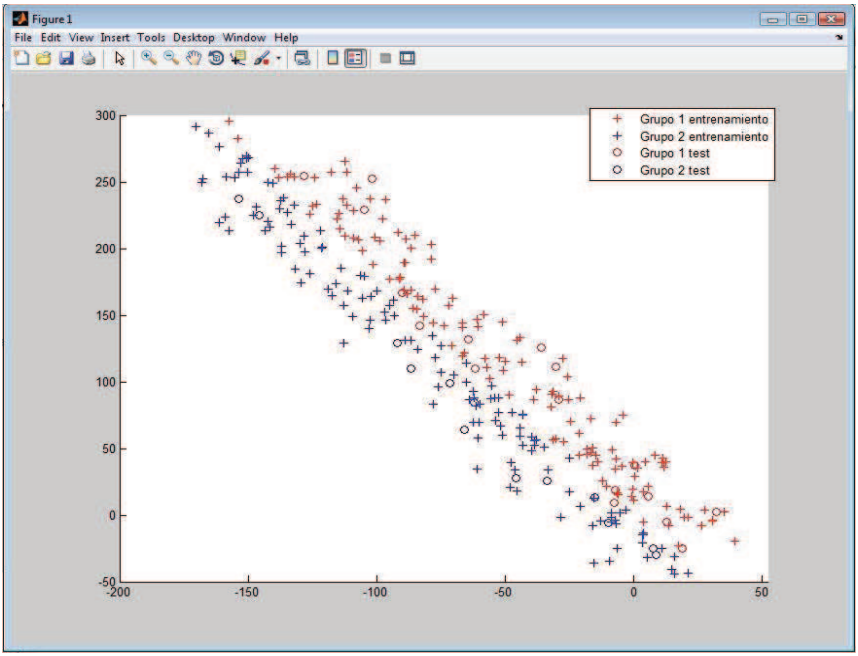

Ilustración 28: Aleatorio100 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 40%.
- Tasa de aciertos para datos proyectados: 93,33%.

Proyección obtenida mediante PCA:

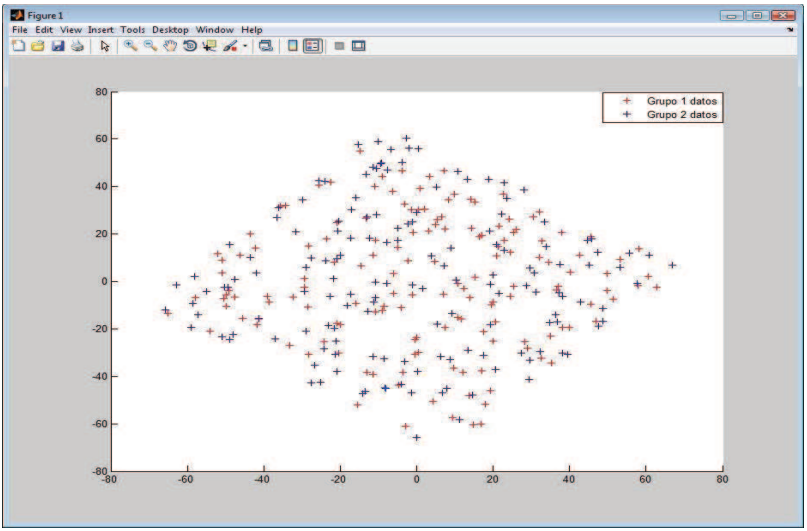

Ilustración 29: Aleatorio100 2D PCA

Proyección obtenida mediante SAMMON:

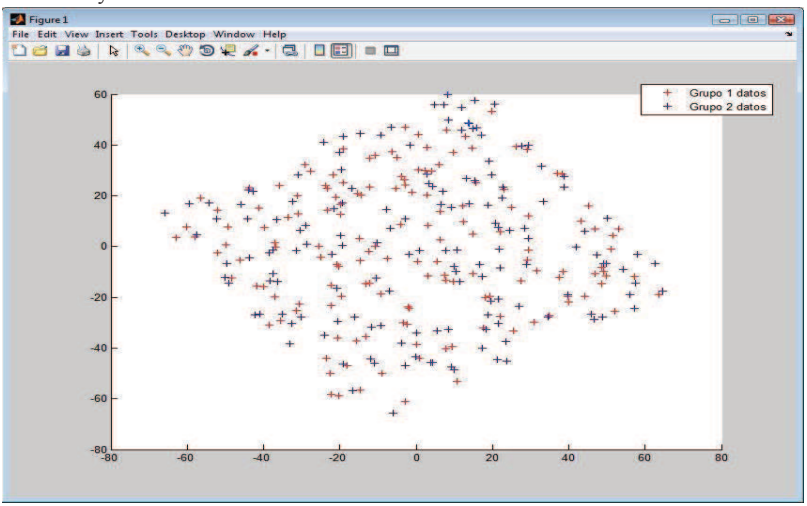

Ilustración 30: Aleatorio100 2D Sammon

Dominio wine:

Proyección mediante una matriz óptima obtenida con CMA-ES:

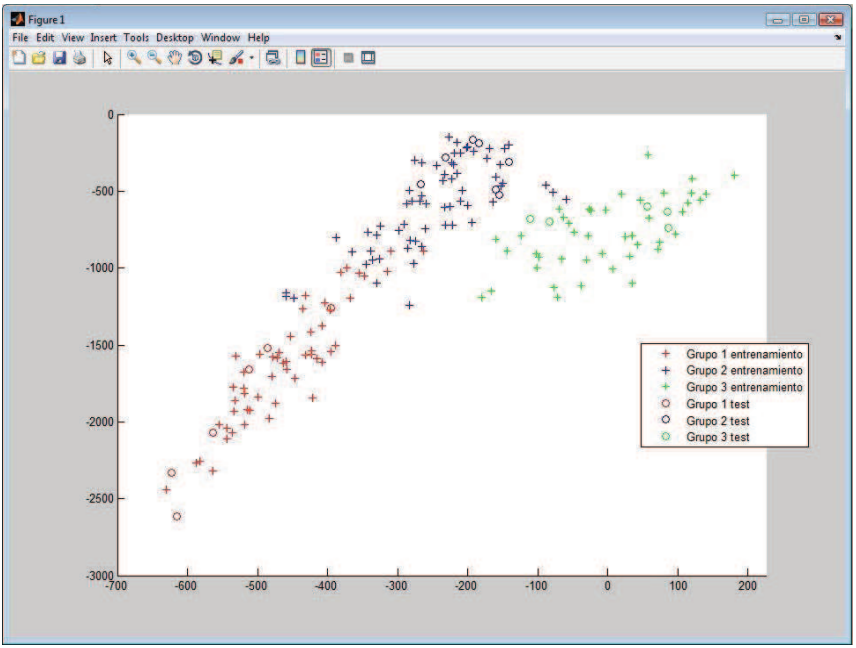

Ilustración 31: Wine 2D CMA-ES

Los resultados de clasificación obtenidos en esta experimentación son los siguientes:

- Tasa de aciertos para datos sin proyectar: 88,89%.
- Tasa de aciertos para datos proyectados: 100%.

Proyección obtenida mediante PCA:

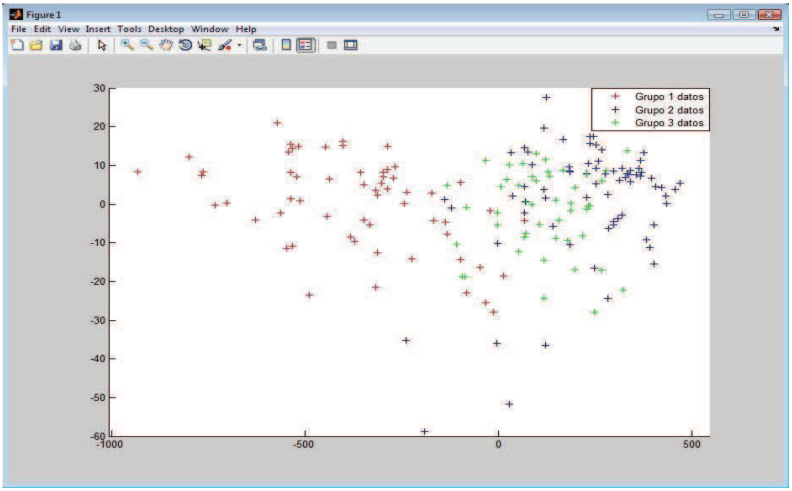

Ilustración 32: Wine 2D PCA

Proyección obtenida mediante SAMMON:

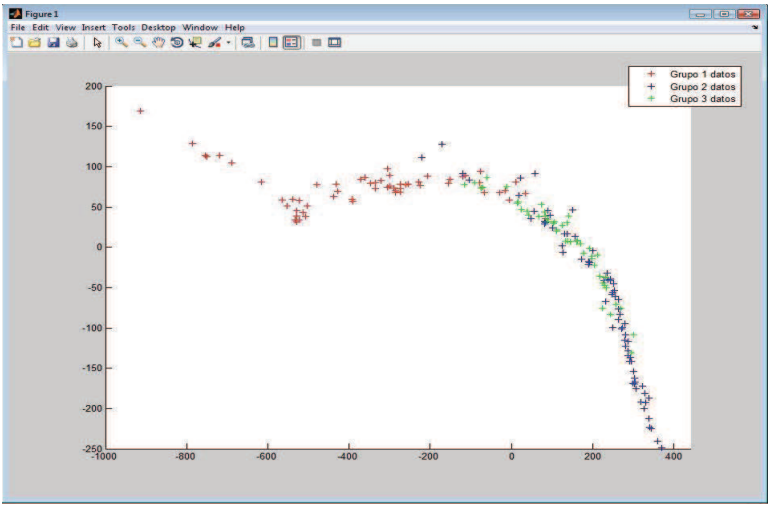

Ilustración 33: Wine 2D Sammon

Para los dominios *rectas0* y *rectas45* tanto la matriz de proyección evolucionada con CMA-ES como PCA consiguen separar perfectamente los datos según el grupo al que pertenecen tal y como puede observarse en las gráficas.

Para el dominio *bupa* ninguno de los métodos de visualización consigue buenos resultados, quedando ambas clases de datos bastante mezcladas.

Sobre el dominio *car* los tres métodos de proyección guardan cierta similitud distribuyendo los datos en tres bloques claramente diferenciables, aunque en cada método esos bloques tengan una apariencia completamente distinta, mediante matrices de proyección se consigue una separación algo más clara en la gráfica.

Los tres métodos de visualización tanto para el dominio *iris* como para el dominio *ripley* consiguen unos resultados bastante parecidos en su forma de distribuir los datos de cada grupo.

Para el dominio *aleatorio100* solamente mediante el uso de matrices de proyección consiguen visualizarse en la gráfica ambas clases de datos perfectamente separados. Para PCA y Sammon quedan completamente mezclados los dos grupos, estos dos métodos consiguen una solución muy similar.

Por último, para el dominio *wine*, los tres métodos de proyección consiguen separar las clases de datos de una forma bastante razonable aunque la matriz de proyección destaca sobre PCA y Sammon.

En conclusión, puede decirse que la utilización de matrices de proyección evolucionadas como método de visualización de datos es muy válido, y puede llegar a ofrecer, en según qué dominios, mejores resultados que algoritmos tan extendidos como PCA o Sammon. Es importante destacar que en ninguno de los dominios se ha observado que destaquen PCA o Sammon sobre las matrices de proyección.

Se muestra a continuación una tabla con los tiempos de ejecución en segundos de cada algoritmo de proyección sobre cada dominio de datos. En la columna de CMA-ES se representa el tiempo que ha tardado en realizar la evolución de una sola matriz sobre cada dominio.

Dominio	PCA	SAMMON	CMA-ES Completa
Rectas-0	~0,5 s	~2 s	~4 s
Rectas-45	~0,5 s	~2 s	~4 s
Aleatorio100	~0,5 s	~3 s	~47 s
Ripley	~0,5 s	~36 s	~200 s
Car	~0,5 s	~67 s	~240 s
Iris	~0,5 s	~1 s	~3 s
Bupa	~0,5 s	~4 s	~60 s
Wine	~0,5 s	~1 s	~21 s

Tabla 9: Tiempos ejecución 2D

El algoritmo que ha marcado más eficiencia en cuanto a tiempo de ejecución ha sido PCA, sin embargo al ser un algoritmo ya incluido en el propio toolbox de matlab posiblemente tenga optimizado su rendimiento internamente.

Los dominios de datos con mayor número de instancias, como *Ripley* y *Car*, son los que más han penalizado los tiempos de ejecución. CMA-ES ha demostrado ser el algoritmo más costoso computacionalmente para estos dominios.

3.3 Multiobjetivo

Esta experimentación tiene como objetivo el estudio del problema en el cual lo que se busca es maximizar el número de aciertos en test de los datos proyectados y a la vez minimizar el número de columnas de la matriz de proyección.

Esto resulta interesante para dominios de datos que tienen un número de atributos elevado. Habrá que considerar un equilibrio entre la maximización en el porcentaje de aciertos y la dimensión (menor a la original) a la que se quiere proyectar los datos. Toda la experimentación multiobjetivo se ha realizado sobre el dominio *ionosphere,* de dimensión 34.

El problema se ha abordado a través de tres fases diferenciadas. En primer lugar se ha experimentado con CMA-ES, añadiendo a los individuos un vector de selección que decida que columnas de la matriz de proyección se usan, manteniendo la función de fitness original explicada en 2.6.1. En segundo lugar, además de la utilización del vector de selección, se ha modificado la función de fitness de forma que se ha abordado el problema multiobjetivo a través de una única función objetivo y la utilización de CMA-ES. Por último, se ha hecho uso de un algoritmo multiobjetivo, *gamultiobj*, que intenta minimizar dos funciones objetivo: el número de columnas seleccionadas y el porcentaje de fallos en clasificación del conjunto de entrenamiento.

El algoritmo de clasificación k-NN se configuró con k=1 para todas las experimentaciones de los puntos siguientes, tal y como se ha venido haciendo a lo largo de toda la experimentación del trabajo.

3.3.1 Vector de selección CMA-ES

Se ha añadido al vector de entrada (matriz de proyección) del algoritmo CMA-ES una serie de números al principio (vector de selección), uno por cada columna de la matriz de proyección. Cada uno de esos números podrá tener el valor cero o uno, e indican si la columna correspondiente de la matriz de proyección será seleccionada en la transformación de los datos o no. El número uno indica que si será utilizada y el cero indica que la columna no será utilizada.

Por tanto, ahora cada individuo del algoritmo evolutivo no será una matriz de proyección sino una matriz junto con su vector de selección.

A lo largo de la ejecución de CMA-ES, será el propio algoritmo el que vaya seleccionando las columnas apropiadas de la matriz de proyección para minimizar los fallos en clasificación del conjunto de entrenamiento. Es importante señalar que no se ha modificado la función de fitness, por lo que lo único que se sigue teniendo en cuenta a la hora de valorar un individuo es el porcentaje de fallos en la clasificación de los datos de entrenamiento proyectados.

Respecto a la implementación, deben aclararse un par de puntos sobre decisiones tomadas. Debido a que es posible que en algún momento en el transcurso del algoritmo se intente evaluar un individuo cuyos valores de selección de columnas sea todo ceros, es decir, no se seleccione ninguna columna, se ha tomado la decisión de que si esto ocurre se utilice la matriz identidad para proyectar los datos, por lo que los datos quedarían igual. Otra decisión que se ha tomado es que los valores iniciales del vector que indica las columnas seleccionadas se inicializa aleatoriamente a unos y ceros.

Los parámetros de ejecución del programa han sido los mismos que se vienen utilizando en ejecuciones anteriores, 3000 evaluaciones. La matriz de proyección inicial es la identidad y el tipo de matriz utilizado es completa. Se ha realizado una ejecución completa de diez validaciones cruzadas según lo explicado en 2.6.1.

Los resultados obtenidos se muestran en la tabla siguiente.

	Aciertos test	Número medio columnas seleccionadas	Dominio
Datos originales	86.58 ± 0.39	34	ionosphere
CMA-ES completa fitness original	88.06 ± 1.28	17.63 (34)	ionosphere

Tabla 10: Multiobjetivo vector selección

El número medio de columnas seleccionadas ha sido de 17.63, de un máximo posible de 34. Ese número medio se encuentra muy cerca de la mitad de columnas de la matriz completa, es decir, la dimensión del dominio *ionosphere* (34). El vector de selección de columnas toma inicialmente valores aleatorios, por tanto, aproximadamente la mitad serán ceros y la otra mitad unos desde un inicio. Por lo que se observa, CMA-ES en su ejecución no se aleja demasiado de ese valor inicial en la mayoría de las ejecuciones.

3.3.2 Multiobjetivo con CMA-ES

En este punto, además de la utilización del vector de selección, se ha modificado la función de fitness para que contemplase el número de columnas de la matriz de proyección al obtener el valor de fitness del individuo. Es decir, se ha utilizado una sola función para abordar el problema multiobjetivo.

La nueva función de fitness tiene la siguiente forma:

$valor\_nuevo\_fitness = valor\_original\_fitness + k * número\_columnas$

Tabla 11: Valor fitness multiobjetivo CMA-ES

Donde *valor_original_fitness* es el valor de fitness original explicado en el punto 2.6.1 de este trabajo, *número_columnas* es el número de columnas de la matriz de proyección y *k* es una constante que penaliza *valor_nuevo_fitness* según el número de columnas de la matriz seleccionadas. Para un mayor valor de *k* la penalización del valor de fitness será mayor.

Los individuos del algoritmo CMA-ES tienen la misma forma que en el apartado 3.3.1; estarán compuestos por una serie de números al principio (vector de selección), uno por cada columna de la matriz de proyección, y la propia matriz de proyección. El número de evaluaciones se ha establecido en todas las ejecuciones en 3000, la matriz inicial es la matriz identidad y el vector de selección inicial es aleatorio compuesto de unos y ceros (aproximadamente la mitad serán unos y la otra mitad ceros). Se ha realizado varias ejecuciones completas de 10 validaciones cruzadas según el algoritmo explicado en 2.6.1 para varios valores de *k*.

En la tabla siguiente se muestra un resumen de los resultados obtenidos:

	k	Aciertos test	Número medio columnas seleccionadas	Función fitness
CMA-ES completa fitness multiobjetivo	0.005	88.14 ± 1.26	9.98 (34)	Multiobjetivo
	0.05	86.03 ± 1.39	6.93 (34)	
	0.2	86.80 ± 1.55	6.82 (34)	

Tabla 12: Resultados multiobjetivo CMA-ES

Tras la experimentación realizada sobre el dominio *ionosphere* se ha podido comprobar que alrededor de un valor de *k* cercano a 0.05 se produce un punto de inflexión. Para valores menores que ~0.05 el número de columnas medio resultante en la ejecución del algoritmo crece y para valores mayores que ~0.05 el número medio de columnas resultante en la ejecución se estanca y decrece muy poco a poco.

Los tiempos de ejecución del algoritmo sobre el dominio *ionosphere* han sido ~ 203 minutos ~ 3 horas y 38 minutos.

Resulta por tanto decisivo, a la hora de proyectar un dominio de datos con muchos atributos, elegir un valor de *k* adecuado para que el número medio de columnas de la matriz decrezca a la vez que se mantiene (o incrementa) el porcentaje de aciertos en test.

3.3.3 Multiobjetivo con gamultiobj

La experimentación multiobjetivo con *gamultiobj*[9] se ha dividido en dos fases. En la primera fase se explica cómo se ha estudiado la forma de selección de un punto del frente devuelto por el algoritmo, en la segunda fase se han realizado varias ejecuciones completas de 10 validaciones cruzadas, usando *gamultiobj* como algoritmo de búsqueda.

[9] http://www.mathworks.es/help/toolbox/gads/gamultiobj.html

3.3.3.1 Selección del punto del frente

Experimentación utilizando el dominio *ionosphera,* ejecutando el algoritmo de búsqueda *gamultiobj* sobre matrices completas con vector de selección de columnas.

La idea de este primer punto de experimentación con *gamultiobj* es la de decidir como escoger el punto del frente devuelto en las sucesivas ejecuciones multiobjetivo.

Se han realizado cinco ejecuciones independientes de *gamultiobj*, en ellas aparece como *objetivo 2* el número de columnas seleccionadas de la matriz de proyección, y como *objetivo1* el porcentaje de fallos en clasificación con el conjunto de entrenamiento. Ambos objetivos pretenden minimizarse.

Para todas las ejecuciones la configuración ha sido la misma: población de 60 individuos y 50 generaciones, el resto de parámetros por defecto.

Se muestra a continuación, en la ilustración siguiente, una de las ejecuciones realizadas.

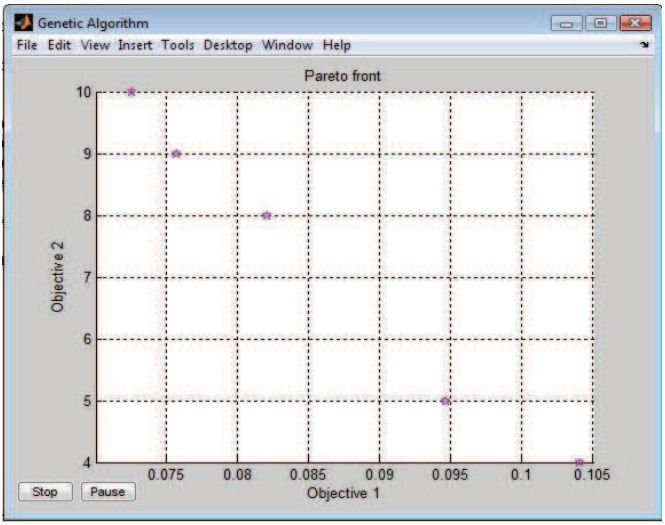

Ilustración 34: Frente ejecución de gamultiobj

En las cinco ejecuciones se forma una curva parecida con los puntos del frente, donde el punto que muestra mayor equilibrio entre ambos objetivos es fácilmente localizable.

Se muestra en la tabla siguiente un resumen de las cinco ejecuciones independientes, con el valor correspondiente del punto más equilibrado y los dos puntos más extremos del frente.

		Ejecución 1	Ejecución 2	Ejecución 3	Ejecución 4	Ejecución 5
Punto más equilibrado	Objetivo 1	9.5%	17%	16%	14%	8.7%
	Objetivo 2	5	10	2	2	5
Punto extremo 1	Objetivo 1	7.2%	13%	8%	5%	7%
	Objetivo 2	10	18	5	4	8
Punto extremo 2	Objetivo 1	10%	19%	24%	25.5%	11%
	Objetivo 2	4	6	1	1	4
	Puntos del frente	7	7	7	7	6

Tabla 13: Resumen frentes de ejecuciones con gamultiobj

Recordemos que el objetivo 2 busca minimizar el número de columnas de la matriz y el objetivo 1 busca minimizar el porcentaje de fallos en entrenamiento.

Al obtener varios puntos en el frente para cada ejecución del algoritmo, debe elegirse uno de ellos como mejor solución que los demás. Recordemos que, en el frente de pareto, cada solución es siempre mejor que todas las demás en uno de los objetivos, por tanto, salvo que obtuviésemos un único punto en el frente, no existe una solución que sea mejor que todas las demás en todos los objetivos.

Por tanto, para escoger el punto del frente que devuelve el algoritmo de búsqueda en cada ejecución se ha decidido tener en cuenta los resultados sobre el conjunto de test de cada punto. Este es el algoritmo de selección del punto del frente:

- Se ejecuta *gamultiobj* con los datos de entrenamiento y se obtiene un conjunto de soluciones (frente de puntos), cada punto del frente es una matriz de proyección. Las dos funciones que pretende minimizar son: número de columnas de la matriz (unos en el vector de selección) y porcentajes de fallos sobre el conjunto de entrenamiento.

- Se evalúan todos los puntos del frente con el conjunto de datos de entrenamiento.

- El punto del frente seleccionado será aquel que obtenga el mayor valor en la siguiente fórmula: *aciertos_sobre_conjunto_entrenamiento - (k*num_columnas_matriz).*
- El valor *k* penaliza el número de columnas que ofrece la solución.

3.3.3.2 Ejecuciones completas con gamultiobj

Se han hecho tres ejecuciones completas, diez validaciones cruzadas, utilizando *gamultiobj*.

La configuración ha sido: población de 60 individuos y 50 generaciones, el resto de parámetros por defecto. El valor *k* está explicado en el punto 3.3.3.1.

La tabla 13 se reunen los resultados de experimentación obtenidos con *gamultiobj* para distintos valores de *k*.

	k	Aciertos test	Número medio columnas seleccionadas
gamultiobj completa, dos funciones objetivo	0.005	88.49 ± 1.48	5.88
	0.02	85.44 ± 1.6	5.3
	0.08	82.65± 1.75	4.85

Tabla 14: Resultados gamultiobj

Para un valor de *k* comprendido entre [0.005, 0.02] y la utilización de *gamultiobj* se encuentran las mejores soluciones buscadas en esta experimentación multiobjetivo, ofreciendo unos aciertos de clasificación en test ligeramente superiores a los datos originales y con proyecciones a dimensiones más bajas que en experimentaciones anteriores.

Los tiempos de ejecución del algoritmo sobre el dominio *ionosphere* han sido ~ 192 minutos ~ 3 horas y 20 minutos.

3.3.4 Resumen ejecuciones multiobjetivo

	k	Aciertos test	Número medio columnas seleccionadas
Datos originales	-	86.58 ± 0.39	34
CMA-ES completa, fitness original	-	88.06 ± 1.28	17.63
CMA-ES completa, fitness multiobjetivo	0.005	88.14 ± 1.26	9.98
	0.05	86.03 ± 1.39	6.93
	0.2	86.80 ± 1.55	6.82
gamultiobj completa, dos funciones objetivo	0.005	88.49 ± 1.48	5.88
	0.02	85.44 ± 1.6	5.3
	0.08	82.65± 1.75	4.85

Tabla 15: Resumen ejecuciones multiobjetivo

En la primera fila se muestra el porcentaje de aciertos en test sobre los datos sin proyectar.

En la segunda fila se muestran los resultados de los datos proyectados añadiendo el vector de selección en la búsqueda; usando la función de fitness original (explicada en 2.6.1). Esta experimentación está explicada en el punto 3.3.1.

En la tercera fila se muestran los resultados que se obtuvieron utilizando la función de fitness adaptada para resolver el problema multiobjetivo con una única función que minimizase el número de columnas y el porcentaje de fallos sobre el conjunto de entrenamiento. Experimentación explicada en 3.3.2.

En la última fila aparecen los nuevos últimos resultados de experimentación, en los que se ha hecho uso de *gamultiobj*. Experimentación explicada en 3.3.3.

Como puede observarse en la tabla, no existe una solución óptima que mejore ambos objetivos sobre el resto de soluciones. Sin embargo, la solución ofrecida por *gamultiobj* con un valor de *k* de 0.005 es posiblemente la solución más interesante, puesto que mantiene el porcentaje de acierto y consigue reducir el número de columnas frente a los resultados de CMA-ES.

El tiempo de ejecución del algoritmo usando el genético multiobjetivo ha sido además algo menor que con la utilización de CMA-ES:

Dominio	Tiempo con CMA-ES	Tiempo con gamultiobj
ionosphere	~ 203 minutos	~ 192 minutos

Tabla 16: Tiempos ejecución multiobjetivo

Capítulo 4

Conclusiones

La experimentación del trabajo ha ido orientada a conseguir, sobre distintos dominios, aumentar el porcentaje de aciertos en clasificación con *k*-NN mediante la realización de transformaciones lineales usando matrices de proyección evolucionadas. Todo esto ubicado dentro de un proceso de diez validaciones cruzadas sobre las que pueden extraerse conclusiones estadísticas fiables.

Sobre los resultados obtenidos pueden extraerse varias conclusiones interesantes.

En primer lugar, la utilización de un algoritmo genético como alternativa a CMA-ES, en la búsqueda de matrices de proyección que aumenten el porcentaje de aciertos en clasificación supervisada, no mejora la búsqueda, siendo ambos métodos bastante parecidos, logrando soluciones similares. Además, la utilización de matrices simétricas evolucionadas no son un buen método para proyectar dominios de datos para conseguir mejorar aciertos en clasificación con *k*-NN, las matrices simétricas se quedan muy lejos de los resultados obtenidos con matrices diagonales y matrices completas generales.

En segundo lugar, ha quedado demostrado que la utilización de matrices de proyección evolucionadas para la realización de transformaciones lineales sobre distintos dominios de datos, con el fin de proyectarlos a dos dimensiones, es una técnica muy interesante para su visualización. Para todos los dominios utilizados, esta técnica ha conseguido iguales o mejores resultados que dos métodos de visualización ampliamente conocidos como son PCA y Sammon. Eso sí, el coste computacional con la utilización de matrices evolucionadas con CMA-ES ha sido notablemente más elevado que con la utilización de PCA y Sammon, sobre todo para los dominios de datos que incluían un mayor número de instancias.

Por último, en lo referente a la búsqueda multiobjetivo, se consiguió encontrar buenas matrices de proyección a dimensiones bajas aumentando ligeramente el porcentaje de aciertos en clasificación sobre el dominio ionosphere, de dimensión 34. Entre los dos métodos utilizados para abordar la búsqueda multiobjetivo, destaca ligeramente la utilización de gamultiobj sobre la utilización de CMA-ES con una única función de fitness adaptada.

Líneas futuras

El estudio futuro puede ir orientado en varias direcciones.

En primer lugar, podría implementarse el algoritmo general sobre un esquema de computación paralela, para explotar al máximo los núcleos que ofrecen los PCs actuales. Esto podría permitir aumentar el número de evaluaciones de CMA-ES sin que supusiese un coste en tiempo de computación demasiado elevado.

Otro punto interesante sería el de continuar la línea de desarrollo del problema multiobjetivo. Podrían utilizarse otros dominios del mundo real, que tuviesen dimensiones elevadas, con el objetivo de afinar el ajuste de los parámetros multiobjetivo utilizados en este trabajo.

Referencias Bibliográficas

[1] Duda, R.O.;Hart, P.E. y Stork, D.G.: *Pattern Classication (2nd edition.)* Wiley-Interscience, 2001.

[2] García V.: *Distribuciones de clases no balanceadas: Métricas, Análisis de Complejidad y Algoritmos de Aprendizaje*. Tesis doctoral, Universitat Jaume I, 2010.

[3] Cover, T.M.; Hart P.E.: *Nearest neighbor pattern classication.* IEEE Transactions on Information Theory 13 (1): 21-27, 1967.

[4] Barandela, R.; Cortés, N. y Palacios, A.: *The nearest neighbor rule and the reduction of the training sample size*. En: Proceedings of the 9th Spanish Symposium on Pattern Recognition and Image Analysis I, pp. 103 108, 2001

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

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

[7] A. Hoffmann, *Arguments on Evolution: A Paleontologist's Perspective*, Oxford University Press, New York, 1989.

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

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

[10] 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

[11] Hansen, N. and S. Kern: *Evaluating the CMA Evolution Strategy on Multimodal Test Functions*. Eighth International Conference on Parallel Problem Solving from

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

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

[14] Dorigo M.; Maniezzo V. and Colorni A.: *The ant system: an autocatalytic optimizing process.* Technical report TR91-016, Politecnico de Millano, 1991.

[15] A. Osyczka: *Multicriteria Optimization for Engineering Design*. Academic Press, 1985.

[16] Nebro A.J.; Alba E.; Luna F.: *Optimización Multiobjetivo y Computación Grid*. Soft Computing, Vol 11, No 6, pp 531-540, 2007.

[17] C.A. Coello, D.A. Van Veldhuizen, and G.B. Lamont: *Evolutionary Algorithms for Solving MultiObjective Problems*. Genetic Algorithms and Evolutionary Computation. Kluwer Academic Publishers, 2002.

[18] C.A. Coello: *A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques*. Knowledge and Information Systems. An International Journal, vol. 1, No. 3, pp. 269-308, 1999.

[19] C. M. Fonseca and P. J. Flemming: *An Overview of Evolutionary Algorithms in Multiobjective Optimization*. Evolutionary Computation, vol. 3, no. 1, pp. 1-16, 1995.

[20] Pérez A.J: *Extracción de características empleando técnicas de optimización aleatoria*. Tesis doctoral, Universidad Politécnica de Valencia, 2003.

[21] K. Fukunaga: *Statistical Pattern Recognition*. Academic Press, second edition edition, 1990.

[22] J.W. Sammon: *A non-linear mapping for data structure analysis*. IEEE Transactions on Computers, C-18(5), 1969.

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

[24] José M. Valls, Ricardo Aler. *Optimizing Linear And Quadratic Data Transformations for Clasification Tasks*. Ninth International Conference on Intelligent Systems Design and Aplications, 2009.

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

[26] D.E. Goldberg: *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[27] M. Mitchell: *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, MA,

1996.

[28] C.L. Blake and C.J. Merz: *UCI repository of machine learning databases*, 1998. http://archive.ics.uci.edu/ml/, University of California, Irvine.

[29] R. Aler, I. M. Galván, and J. M. Valls: *Applying Evolution Strategies to preprocessing EEG signals for Brain Computer Interfaces*.

APÉNDICE A: Manual del programa

Dentro de la carpeta "Codigo_entrega", en el CD adjunto, se encuentran todas las funciones (archivos *.m*) programadas y utilizadas a lo largo del proyecto.

La subcarpeta "Entrada" contiene los archivos de datos utilizados en la experimentación. En la subcarpeta "resultados" se escriben los ficheros de salida de las funciones implementadas. En la subcarpeta "somtoolbox" se encuentra toda al api de funciones necesaria para ejecutar el algoritmo Sammon.

Cada uno de los programas implementados se describe mediante una tabla según la siguiente nomenclatura:

Nombre	**Nombre de la función principal, que se corresponderá con un archivo *.m*.**	
Descripción	Explicación detallada del funcionamiento del programa.	
Dependencias principales	**Funciones implementadas**	Funciones fundamentales implementadas a las que hace referencia el programa.
	Otras funciones	Funciones más destacables del propio toolbox de Matlab (o externas) que han sido utilizadas para desarrollar este programa.

Tabla 17: Nomenclatura tablas desarrollo

Funciones principales

Funciones que pueden ejecutarse directamente desde la línea de comandos de Matlab, sin pasarles ningún parámetro. En el cuerpo de estas funciones se configura los parámetros fundamentales de ejecución del programa y se realiza la llamada a la función de optimización correspondiente.

Nombre	**prinGASimCMAES.*m*.**	

Descripción	Realiza 10 validaciones cruzadas de 10 folds cada una, con los datos del fichero cargado.
	Para cada fold se obtendrán tres matrices de proyección optimizadas mediante un algoritmo genético (matriz completa), CMA-ES (matriz completa) y CMA-ES (matriz simétrica), que maximicen los aciertos con k-NN (k=1).
	Para cada conjunto de test se proyectarán los datos con las matrices obtenidas para clasificarlos con k-NN (k=1), además también se calculan los resultados de k-NN (k=1) de los datos sin proyectar.
	Escribirá dentro de la carpeta "resultados" (dentro de Codigo_Entrega) en un fichero los resultados completos de test además de sacarlos por pantalla.
	
	Desde la función principal se inicializan los parámetros fundamentales de ejecución del algoritmo:
	
	• Nombre del fichero de datos.
	• Parámetros de CMA-ES:
	  ○ Número máximo de evaluaciones
	  ○ Desviación inicial.
	• Parámetros del algoritmo genético:
	  ○ Tamaño de población.
	  ○ Número de generaciones.	

Dependencias principales	**Funciones implementadas**	• optimGASimCAMES.m • obtenerMatrizOptimaCompGA.m • obtenerMatrizOptimaCompCMAES.m • obtenerMatrizOptimaSimCMAES • fitnessCMAESComp.m • fitnessCMAESSim.m
	Otras funciones	• knnclassify.m • ga.m • cmaes.m

Tabla 18: Función prinGASimCMAES.m

Nombre	**prinPCA.*m***	
Descripción	Este programa carga los datos del fichero especificado y realiza la llamada a la función princomp.m que lanza el algoritmo PCA para proyectar los datos a dos dimensiones. Una vez obtenidos los datos de PCA, se representan gráficamente utilizando las funciones gscatter.m y legend.m, los parámetros de dichas funciones deberán modificarse manualmente en base al número de clases en las cuales se pueden clasificar las instancias del conjunto de datos.	
Dependencias principales	**Funciones implementadas**	-
	Otras funciones	• princomp.m • gscatter.m

Tabla 19: Función prinPCA.m

Nombre	**prinSammon.*m***	
Descripción	Este programa carga los datos del fichero especificado y realiza la llamada a la función sammon.m que lanza el algoritmo SAMMON para proyectar los datos a dos dimensiones. Una vez obtenidos los datos de SAMMON, se representan gráficamente utilizando las funciones gscatter.m y legend.m, los parámetros de dichas funciones deberán modificarse manualmente en base al número de clases en las cuales se pueden clasificar las instancias del conjunto de datos.	
Dependencias principales	**Funciones implementadas**	-
	Otras funciones	• sammon.m • gscatter.m

Tabla 20: Función prinSammon.m

Nombre	**prin2DCMAES.*m***	

Descripción	Este programa carga los datos del fichero especificado y obtiene una matriz de proyección a dos dimensiones optimizada mediante CMA-ES que maximice los aciertos con k-NN (k=1) usando datos de entrenamiento.
	
	Con dicha matriz proyectarán los datos de test a dos dimensiones y realizará una representación gráfica de los mismos. Dentro de la función optim2D.m se deberá modificar manualmente los parámetros de las fuciones gscatter.m y legend.m en base al número de clases en las cuales se pueden clasificar las instancias del conjunto de datos.
	
	Desde la función principal se inicializan los parámetros fundamentales de ejecución del algoritmo:
	
	• Nombre del fichero de datos.
	• Numero de dimensiones a las que se proyecta.
	• Parámetros de CMA-ES:
	  ○ Número máximo de evaluaciones
	  ○ Desviación inicial.	

Dependencias principales	**Funciones implementadas**	• optim2D.m • obtenerMatrizOptimaCompCMAES2D.m • fitnessCMAESComp2D.m
	Otras funciones	• knnclassify.m • cmaes.m • gscatter.m

Tabla 21: Función prin2DCMAES.m

Nombre	**prin2DCMAES10ejec.*m*.**	

Descripción	Realiza 10 validaciones cruzadas de 10 folds cada una, con los datos del fichero cargado.
	Para cada fold se obtendrá una matriz de proyección a dos dimensiones optimizada mediante CMA-ES (matriz completa) que maximice los aciertos con k-NN (k=1).
	Para cada conjunto de test se proyectarán los datos con las matrices obtenidas para clasificarlos con k-NN (k=1), además también se calculan los resultados de k-NN (k=1) de los datos sin proyectar.
	Escribirá dentro de la carpeta "resultados" (dentro de Codigo_Entrega) en un fichero los resultados completos de test además de sacarlos por pantalla.
	
	Desde la función principal se inicializan los parámetros fundamentales de ejecución del algoritmo:
	• Nombre del fichero de datos.
	• Numero de dimensiones a las que se proyecta.
	• Parámetros de CMA-ES:
	  ○ Número máximo de evaluaciones
	  ○ Desviación inicial.	

Dependencias principales	**Funciones implementadas**	• optim2D10Ejec.m • obtenerMatrizOptimaCompCMAES2D.m • fitnessCMAESComp2D.m
	Otras funciones	• knnclassify.m • cmaes.m

Tabla 22: Función prin2DCMAES10ejec.m

Nombre	prinCMAESMultiobj.*m*	

Descripción	Realiza 10 validaciones cruzadas de 10 folds cada una con los datos del fichero cargado.
	Para cada fold se obtendrá una matriz de proyección (de dimensión variable) optimizada mediante CMA-ES que maximice los aciertos con k-NN (k=1). En la función de fitness se penalizará el número de columnas de la matriz de proyección evaluada, esta variable puede graduarse en el fichero fitnessCMAESCompMultiobj.m, se llama penalizador.
	Para cada conjunto de test, se proyectarán los datos con las matrices obtenidas para clasificarlos con k-NN (k=1), además también se calculan los resultados de k-NN (k=1) de los datos sin proyectar.
	Escribirá dentro de la carpeta "resultados" (dentro de Codigo_Entrega) en un fichero los resultados completos de test además de sacarlos por pantalla.
	
	Desde la función principal se inicializa los parámetros fundamentales de ejecución del algoritmo:
	• Nombre del fichero de datos.
	• Parámetros de CMA-ES:
	  ○ Número máximo de evaluaciones
	  ○ Desviación inicial.	

Dependencias principales	Funciones implementadas	• optimCMAESMultiobj.m • obtenerMatrizOptimaCompCMAESMultiobj.m • fitnessCMAESCompMultiobj.m
	Otras funciones	• knnclassify.m • cmaes.m

Tabla 23: Función prinCMAESMultiobj.m

<table>
<tr><td>Nombre</td><td colspan="2">prinGamultiobj.m.</td></tr>
<tr><td>Descripción</td><td colspan="2">Realiza 10 validaciones cruzadas de 10 folds cada una con los datos del fichero cargado.
Para cada fold se obtendrá una matriz de proyección (de dimensión variable) optimizada mediante el algoritmo genético multiobjetivo (gamultiobj.m) que maximice los aciertos con k-NN (k=1).
Para cada conjunto de test, se proyectarán los datos con las matrices obtenidas para clasificarlos con k-NN (k=1), además también se calculan los resultados de k-NN (k=1) de los datos sin proyectar.
Escribirá dentro de la carpeta “resultados” (dentro de Codigo_Entrega) en un fichero los resultados completos de test además de sacarlos por pantalla.

Desde la función principal se inicializa los parámetros fundamentales de ejecución del algoritmo:

• Nombre del fichero de datos.
• Parámetros de la función de fitness:
  ○ Constante k de penalización por número de columnas de la solución obtenida.
• Parámetros del algoritmo genético:
  ○ Tamaño de población.
  ○ Número de generaciones.</td></tr>
<tr><td rowspan="2">Dependencias principales</td><td>Funciones implementadas</td><td>• optimGamultiobj.m
• obtenerMatrizOptimaCompGAM
• fitnessCompGAM</td></tr>
<tr><td>Otras funciones</td><td>• gamultiobj.m
• knnclassify.m</td></tr>
</table>

Tabla 24: Función prinGamultiobj.m

APÉNDICE B: Planificación y Presupuesto

B.1 Planificación

Dentro del CD entregado del proyecto se encuentra el archivo de microsoft project con la planificación del proyecto.

El proyecto se dividió en tres fases principales: problema de estancamiento, proyección a dos dimensiones y problema multiobjetivo.

Cada una de las fases principales consta de varias subfases: estudio del arte sobre cada uno de los problemas, elaboración de la experimentación (incluye diseño e implementación de la solución en Matlab) y elaboración de un informe de resultados.

Por último, se elaboró la memoria final a partir de los distintos estudios del estado del arte y de los distintos informes de resultados obtenidos.

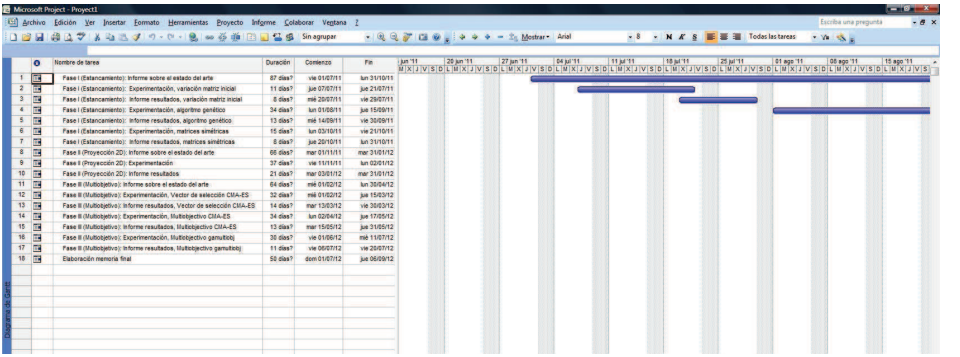

Ilustración 35: Planificación

El total de días trabajados es de 311 a media jornada equivalente a 5,18 meses/persona.

B.2 Presupuesto

Los distintos módulos que conforman el presupuesto del proyecto son los siguientes:

- Personal: Una única persona trabajando durante 15 meses, según el coste por hora de un Ingeniero Superior en la Universidad Carlos III de Madrid sería de 22.219,82€.
- Ordenador de sobremesa intel core i7, 6 GB de RAM: 1600€.
- Portátil Acer Aspire: 900€.
- Licencia de MATLAB 7.8: De tipo académico, 500€.
- Licencias para bibliotecas de MATLAB. Bioinformatics Toolbox y Statistics Toolbox, 400 x 2 = 800€.
- Microsoft Office 2007: Gratis, obtenido de la universidad Carlos III.
- Windows Vista 64 bits: Gratis, obtenido de la universidad Carlos III.
- Windows 7 Ultimate 32 bits: Gratis, obtenido de la universidad Carlos III.

El presupuesto total es de 28.974€ VEINTIOCHO MIL NOVECIENTOS SETENTA Y CUATRO 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 excel con los datos rellenos.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:
Alfredo Moreno Garrido
2.- Departamento:
EVANNAI
3.- Descripción del Proyecto:
- Titulo: TÉCNICAS EVOLUTIVAS MULTIOBJETIVO PARA LA EVOLUCIÓN DE MATRICES DE PROYECCIÓN EN PROBLEMAS DE CLASIFICACIÓN
- Duración (meses): **15**
Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):
Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Moreno Garrido, Alfredo		Ingeniero Senior	5,18	4.289,54	22.219,82	
		Ingeniero		2.694,39	0,00	
					0,00	
					0,00	
		Hombres mes	**5,18**	**Total**	22.219,82	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador de sobremesa intel core i7	1.600,00	100	15	60	400,00
Portátil Acer Aspire	900,00	100	15	60	225,00
		100		60	0,00
		100		60	0,00
		100		60	0,00
					0,00
				Total	625,00

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Licencias Matlab		1.300,00
	Total	1.300,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	22.220
Amortización	625
Subcontratación de tareas	0
Costes de funcionamiento	1.300
Costes Indirectos	4.829
Total	28.974