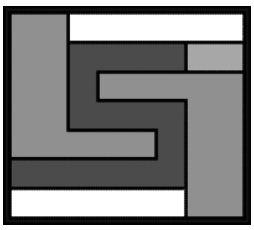
PONDERACIÓN LOCAL EVOLUTIVA DE LA REGLA kNN



DEPARTAMENTO DE LENGUAJES Y SISTEMAS INFORMÁTICOS

Memoria presentada para optar al grado de Doctor

Daniel Mateos García

Supervisada por el Prof. Dr. José Cristóbal Riquelme Santos

Universidad de Sevilla , 2013

Este trabajo es una contribución al proyecto «Análisis Inteligente de Información Medioambiental» (TIN2011-28956-C02-02) financiado por el Ministerio de Ciencia e Innovación.

A mi sobrina Irene,
porque no sabía que algo tan pequeño
pudiera llenar tantos corazones.

Índice general

Índice de figuras

Índice de cuadros

Agradecimientos

Hasta ahora había creído que las cosas cuanto antes se hicieran, mejor. Pero con el transcurrir de los años, uno se da cuenta de que lo importante de verdad es cumplir objetivos, ya sea antes o después.

En mi caso, esta disertación ha llegado en un buen momento. Tan bueno como en algún momento pasado o incluso futuro. Seguramente, si me hubiera doctorado hace algunos años atrás, podría haber escalado con mayor facilidad hacia otras categorías profesionales. Eso mismo piensan muchos de mis compañeros, castigados sin tregua por los efectos de una crisis injusta.

No dejo de pensar en cómo Cristina o Jose María, han tenido que emigrar fuera de nuestras fronteras para poder tener un futuro más claro. O cómo Carlos se levanta a veces pensando en cómo podrían haber sido las cosas si le hubieran declarado «apto» para el perfil de profesor colaborador hace algunos años. Todos ellos son un ejemplo de superación a seguir, porque lejos de tirar la toalla, siguen investigando y enriqueciendo el Conocimiento con sus aportaciones. Su aptitud la acreditan, entre otras cosas, las publicaciones de decenas de páginas en revistas de impacto, y no un párrafo hierático en Arial sobre la carilla de una carta en serie. Yo estoy seguro de que el tiempo les dará la razón.

Por ellos pienso que esta memoria llega en un buen momento. Porque ellos me han hecho ver las cosas de otra manera. Me han enseñado a apreciar lo que tengo y lo que soy, a luchar ante la adversidad. A caerme y levantarme.

Tampoco me puedo olvidar de mi mujer Lidia, y de mi hermano Alejandro, ahora con su pareja Sara y una preciosa niña en el mundo de nombre Irene, que aún sin hablar, puede aliviar el silencio de los que ya se fueron. Y también están mis padres Marisol y Gabriel, y mi segunda familia «política». En todos ellos encuentro la inspiración, el tesón y el optimismo en tiempos difíciles. Son un apoyo incondicional de valor incalculable que siempre están ahí cuando les necesito.

Y por supuesto, Jorge y Pepe, ambos compañeros y amigos, más de lo segundo que de lo primero, y de las personas más talentosas y brillantes que conozco. Ambos poseen la humildad

del aprendiz y la determinación del maestro, virtudes sine qua non para la investigación y la labor docente.
Gracias a ellos, estoy escribiendo estas líneas.

Resumen

En la literatura, existen numerosas técnicas para mejorar el rendimiento de la regla de los k vecinos más cercanos (en inglés, k Nearest Neighbours o kNN). Caben destacar aquellas que se centran en la selección del número de vecinos (parámetro k) y las que establecen unos pesos para el conjunto de características de los ejemplos que conforman los datos a clasificar. En este trabajo, exponemos una propuesta basada en algoritmos evolutivos llamada *k-Label Dependent Evolutionary Distance Weighting* (kLDEDW) que además de estimar un k óptimo, calcula un conjunto de pesos locales que dependen de cada clase. De esta manera, a partir de un conjunto de datos de entrenamiento, nuestra propuesta es capaz de calcular un valor óptimo de k que se complementa con una transformación local del espacio de características. Para validar nuestro trabajo de forma exhaustiva, se han utilizado tanto datos de benchmarking como reales. Los primeros nos han permitido efectuar rigurosos tests estadísticos contra otras propuestas de la literatura basadas en ponderación local. Con los segundos, hemos podido mostrar el comportamiento de kLDEDW frente a datos masivos y ruidosos.

1

Introducción

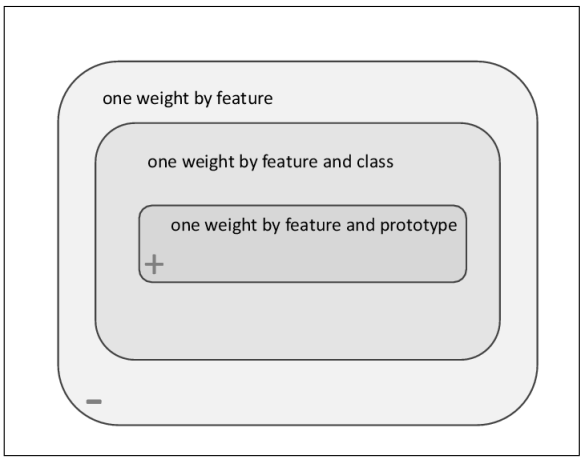
«Lanza primero tu corazón y tu caballo saltará el obstáculo. Muchos desfallecen ante el obstáculo. Son los que no han lanzado primero el corazón.»

– Noel Clarasó

El uso de modelos basados en pesos es muy común en aprendizaje automático, y más concretamente, en problemas de clasificación [83]. Un ajuste adecuado de pesos durante la fase de entrenamiento, mejora la tasa de predicción de un modelo dado. Las redes neuronales artificiales son quizás el ejemplo más claro [50, 63], pero también son utilizados en las máquinas de vectores de soporte [74] y ampliamente en el vecino más cercano [12] con el fin de conseguir un mejor ajuste de la capacidad de predicción. Todos estos métodos tienen en común la búsqueda de pesos óptimos en la etapa de entrenamiento evitando en la medida de lo posible el sobreajuste. En cuanto al método de optimización, éste puede ser de naturaleza analítica (p. ej. RELIEF [78]) o también heurística, como la computación evolutiva [3] o la búsqueda tabú [80].

Si revisamos las investigaciones en la literatura relativas a modelos de pesado y vecinos, podemos comprobar que la mayoría de ellos usan un único conjunto de pesos para todas las instancias de los datos en observación. Por ejemplo, en el vecino más cercano pesado, se obtiene un valor ω_i para cada característica f_i. Esta notación representa la importancia de f_i en el cálculo de la distancia entre dos instancias.

Existen además propuestas de pesado local, que usan pesos aplicados principalmente a prototipos (regiones) [25] o clases [3, 11]. Teniendo en cuenta el coste computacional del enfoque basado en prototipos y desde el punto de vista del paradigma evolutivo, hemos analizado la última opción. Esto es, la obtención de una matriz de pesos (un peso por característica y clase) (ver Figura 1.1).



El nivel más interno representa un pesado local más específico (+), mientras que los niveles más externos representan un sistema global de pesado, y por tanto menos especializado (-).

Figura 1.1: Niveles de pesado

Exploramos la hipótesis de que la influencia sobre una característica no debería ser la misma para cada etiqueta o clase. Por ejemplo, si quisiéramos clasificar pacientes de acuerdo a tres etiquetas que representasen variantes de una enfermedad, la caracterítica «edad» podría ser más importante para una variante que en otra, y por tanto, podría ser más apropiado tener diferentes pesos en función de cada variante. La dificultad de esta hipótesis es obvia: si queremos clasificar un nuevo ejemplo con etiqueta desconocida, y tenemos diferentes conjuntos de pesos para cada clase, ¿qué conjunto de pesos debería aplicarse?
Como resultado de la hipótesis analizada y su posible resolución, proponemos una heurística basada en algoritmos evolutivos que primeramente busca una matriz de pesos por clase en la fase de entrenamiento, y posteriormente (en la fase de prueba), selecciona uno de los vectores de pesos para clasificar una instancia de prueba dada, de acuerdo a un clasificador basado en los k vecinos más cercanos (kNN). Además de esto, la búsqueda del conjunto de pesos se complementa con el cálculo de un número de vecinos óptimo.

1.1. Objetivos

A continuación, se van a detallar los objetivos principales de este trabajo. Algunos de ellos ya han sido alcanzados parcialmente, mientras que otros, sirven como punto de partida para trabajos futuros.

- **Implementación del método propuesto**

Establecida la hipótesis de partida, la principal tarea consiste en su prueba o refutación. Para ello es necesario una revisión profunda del estado del arte para, de esta manera, establecer un marco de trabajo en el que se identifiquen aquellos aspectos que puedan ser mejorados, o incluso que supongan una nueva implementación. Además, se hace necesaria la elección de las herramientas que posibiliten el diseño de una o varias soluciones. En nuestro caso, la hipótesis de partida consiste en que *un conjunto de pesos locales sustentado en un k óptimo, debería mejorar el comportamiento de la regla kNN*. Necesitamos por tanto elegir un método de búsqueda para la comprobación de tal hipótesis. En nuestro caso, y dada nuestra experiencia, el método expuesto en esta memoria está basado en computación evolutiva. Además de lo anterior, es sabido que para problemas reales, la computación evolutiva tiene dificultades en su rendimiento debido a sus características. Para paliar tales deficiencias, hemos considerado una arquitectura de paralelismo apropiada en función de las necesidades y los recursos disponibles hasta el momento.

- **Elaboración de pruebas**

Una vez obtenida una solución estable, es necesario establecer una comparativa entre nuestro método, y los más afines encontrados en la literatura. Para ello ha sido imprescindible la implementación de algunos de ellos, lo cual ha incrementado notablemente el conocimiento del trabajo relacionado que gira en torno a la hipótesis, pero también, ha aumentado la dificultad de la fase experimental. En cualquier caso, este factor ha impulsado notablemente la calidad de nuestro trabajo. Para la fase experimental, se han elegido datos de carácter heterogéneo (múltiples clases y distintos tipos de características), los cuales, han tenido que ser buscados y formateados en algunos casos. También se han utilizado datos reales con el fin de facilitar la transferencia tecnológica de nuestro método. En cuanto a la validez de los resultados, éstos han sido dotados de objetividad mediante el uso de test estadísticos que ya han sido utilizados en la literatura [16, 30].

- **Publicación de resultados**

Los puntos anteriores culminan con la publicación de resultados. Actualmente, se han conseguido tres publicaciones en congresos nacionales e internacionales, y dos artículos en revistas indexadas de cierta relevancia (*Pattern Recognitions Letters* con 1.034 de factor de impacto en *Journal Citation Reports®*, y *Neurocomputing* con 1.580 [55],

[31]). Pero tal y como se observa en las conclusiones de la disertación, quedan abiertas líneas de investigación muy interesantes, cuyos resultados se sumarán previsiblemente, a los objetivos ya conseguidos en este punto.

- **Transferencia tecnológica**

  Otro objetivo de relevancia consiste en la aplicación de nuestro método en más casos reales y en la intensificación de las relaciones de colaboración con entidades públicas y privadas. Este hecho, tendrá como consecuencia una inmersión multidisciplinar que dará como resultado nuevas ideas de investigación, y nuevas publicaciones postdoctorales.

1.2. Periodo de investigación

Con respecto a los objetivos ya alcanzados en el periodo de investigación, podemos destacar los siguientes:

1.2.1. Principales aportaciones originales

1.2.1.1. Artículos publicados en revistas de impacto

- Mateos-García D, García-Gutiérrez J, Riquelme-Santos JC. On the evolutionary optimization of k-NN by label-dependent feature weighting. Pattern Recognition Letters, vol. 33, Issue 16, pp. 2232-2238, 2012.

- García-Gutiérrez J, Mateos-García D, Riquelme-Santos JC. EVOR-STACK: a label-dependent evolutive stacking on remote sensing data fusion. Neurocomputing, vol. 74, n 19, pp. 115-122, 2012.

1.2.1.2. Artículos publicados en congresos internacionales

- Mateos-García D, J García-Gutiérrez, JC Riquelme. Label Dependent Evolutionary Feature Weighting for Remote Sensing Data. 5th IC on Hybrid Artificial Intelligence Systems (HAIS '10). LNCS 6077, pp. 272-279, 2010.

- García-Gutiérrez J, D Mateos-García, JC Riquelme. A SVM and k-NN Restricted Stacking to Improve Land Use and Land Cover Classification. 5th IC on Hybrid Artificial Intelligence Systems (HAIS '10). LNCS 6077, pp. 493-500, 2010.

1.2.1.3. Artículos publicados en congresos nacionales

- García-Gutiérrez J, D Mateos-García, JC Riquelme. Stacking Restringido Sobre SVM y K-nn para Mejorar la Clasificación de Mapas Lulc. III Congreso Español de Informática (CEDI 2010). Garceta Grupo Editorial, pp. 243-250, 2010.

1.2.2. Participación en proyectos

- Análisis Inteligente de Información Medioambiental (TIN2011-28956-C02-02 - Investigador)
- Heurísticas escalables para la extracción de conocimiento en grandes volúmenes de información (TIN2007-68084-C02-02 - Investigador)
- Técnicas emergentes de minería de datos para la extracción de conocimiento de grandes volúmenes de información: aplicación a datos científicos e industriales (TIN2004-00159 - Investigador)

1.3. Organización

En los siguientes capítulos de esta memoria, se mostrará cómo la combinación de matrices de pesos y un número adecuado de vecinos proporcionan una mejora objetiva en el comportamiento de la regla kNN. Para la consecución de dicho fin, se ha organizado la memoria como sigue. En el capítulo 2, se describe el estado del arte de aquellos aspectos relacionados con el propósito final de esta disertación. Se abordarán por tanto las bases del algoritmo kNN y sus variantes, y se explicarán los paradigmas más representativos en el aprendizaje de modelos ponderados. Por otro lado, puesto que la optimización de k y de la matriz de pesos se llevan a cabo mediante un algoritmo evolutivo, se ha dedicado el capítulo 3 a la exposición de las bases de la computación evolutiva. El principal problema que presenta el paradigma evolutivo es la eficiencia. Es por eso que nuestro algoritmo ha sido implementado para ejecutarse en paralelo en una de las muchas vertientes que existen en la literatura. Algunos de estos enfoques son descritos en el capítulo 4, donde también se introducen los principales aspectos de la computación concurrente y distribuída. En el capítulo 5 se detallará en profundidad el propósito, funcionalidad y diseño de nuestra propuesta mientras que los resultados obtenidos, serán ampliamente analizados en el capítulo 6. Finalmente en el capítulo 7, se verán las conclusiones y el trabajo futuro.

2

Estado del arte

«El conocimiento se adquiere leyendo la letra pequeña de un contrato; la experiencia, no leyéndola.»

– Francis Bacon

En este capítulo, se repasarán los conceptos y técnicas de la literatura necesarios para una mejor comprensión del objetivo de esta memoria. Como se ha indicado en el capítulo introductorio, vamos a describir un clasificador basado en la regla kNN, que es mejorado mediante la búsqueda de una serie de pesos y un valor óptimo de k. Es por ello que en las sucesivas secciones, se va a desgranar cada elemento en el contexto establecido en el estado del arte, para al final justificar en cierta medida nuestra motivación y consecuente aportación. Así, en las secciones 2.1 y 2.2 se repasarán los conceptos básicos del vecino más cercano y sus variantes más importantes respectivamente. El concepto general de pesado en clasificadores se verá en la sección 2.3. Y por último en la sección 2.4, se aunarán los conceptos de ponderación y regla kNN.

2.1. Vecino más cercano

La regla del vecino más cercano (NN, Nearest Neighbour), es una técnica de aprendizaje basada en la hipótesis de que los miembros de un conjunto, suelen compartir características comunes o algunos valores cercanos. Por tanto, mediante la observación de los «vecinos» más próximos a un elemento del que se desconoce su pertenencia a alguna de las poblaciones conocidas, es posible deducir la información necesaria para categorizarlo de manera automática. Concretamente, la forma más básica de la regla NN considera sólamente el vecino más próximo.

Las bases de esta hipótesis fueron establecidas por Fix y Hodges [27, 26] a principio de los años 50, pero no fue hasta mediados de los 60 cuando Cover y Hart [14] enunciaron formalmente la regla del vecino más cercano para el reconocimiento de patrones.
Debido a la sencillez del concepto y a su eficacia, este método se ha convertido en una de las técnicas de clasificación más usadas hasta ahora, pudiéndose contar por cientos las variantes y mejoras que la comunidad científica ha ido desarrollando hasta el día de hoy.
La regla NN puede ser descrita de la siguiente manera:
Sea $\varepsilon = \{e_1, ..., e_m\}$ un conjunto de datos con m ejemplos o instancias etiquetadas, donde cada ejemplo e_j contiene n atributos $(x_{j1}, ..., x_{jn})$ pertenecientes al espacio métrico Ω, y es clasificado con una etiqueta $y_l \in Y = \{y_1, ..., y_k\}$. Sea además, $\Gamma : \Omega \to Y$ una aplicación tal que $\Gamma(e_j) = y_l$. La clasificación de un nuevo ejemplo e' cumple que

$$\Gamma(e') = \Gamma(e_i) \Leftrightarrow \forall j \neq i . d(e', e_i) < d(e', e_j) \tag{2.1}$$

donde d expresa una distancia (o función de similaridad) definida en el espacio n-dimensional Ω. De esta manera, un ejemplo de prueba es etiquetado según la clase de su vecino más cercano.
La elección de la función de similaridad es importante, ya que determina la semántica del concepto de similitud, y por ende de vencidad. Tanto es así, que la utilización de distintas funciones sobre un mismo conjunto etiquetado, puede producir resultados muy diferentes a la hora de clasificar un nuevo ejemplo.
Prever a priori la idoneidad de una función de similaridad para un conjunto dado, es un problema abierto. Es el experto el que, en base a su experiencia y a través de la observación, realiza tal elección. Pero también existen propuestas que permiten el aprendizaje de funciones de similaridad *ad hoc* (véase [66] por ejemplo).
Una generalización de la regla NN, consiste en extenderla a los k vecinos más cercanos, de tal manera que la asignación de la etiqueta se realiza en base a la clase mayoritaria. Esta generalización se denomina kNN.
Los principales problemas de este enfoque, son la elección de k y la resolución de empates cuando no existe una etiqueta mayoritaria.
Para ambas dificultades, existen propuestas en la literatura. Por ejemplo, Riquelme et al. [46] intentan predecir el comportamiento de k en el espacio de características, para de esa manera obtener un patrón que determine a priori cuál es el número idóneo de vecinos para clasificar un ejemplo dado.
Para prevenir empates, se suelen utilizar valores impares de k, y si aún así persistiese el

empate, se suele elegir o bien una de las etiquetas mayoritarias de manera aleatoria, o la del vecino más cercano.
El método kNN pertenece al conjunto de técnicas de aprendizaje denominadas perezosas (lazy learning). Esto es debido a que no existe un modelo que represente al conjunto de entrenamiento, ni existe una fase de aprendizaje. El principal inconveniente que presenta este enfoque, radica en la necesidad de tener que recorrer todos los ejemplos de entrenamiento para descubrir cuáles son los vecinos más cercanos a un punto a clasificar. Esto supone un gran coste computacional en tiempo, pero también en espacio (todo el conjunto de entrenamiento debe estar en memoria).
A pesar de lo anterior, y de la necesidad de parametrizar tanto la función de similaridad como el número de vecinos, y de resolver empates, kNN demuestra un buen comportamiento en muchos tipos de problemas. Así, podemos encontrar aplicaciones en el campo de la compresión de datos [33], bioinformática [60], multimedia [28], visión artificial [9] o recuperación de información [53].

2.2. Variantes del vecino más cercano

En esta sección, se verán diferentes mejoras para la regla NN[1]. En general, estas modificaciones pueden estar dirigidas a mejorar la tasa de clasificación mediante ponderación, o procesamiento de los datos de entrenamiento, pero también existen propuestas que se centran en la mejora de la complejidad computacional.
El sistema arquetípico de pesado en el paradigma NN, utiliza una regla de votación en la que los distintos miembros del conjunto de vecinos, son ponderados en función de su distancia a la otra instancia a clasificar. Dudani [17] fue el primero que propuso un esquema de este tipo con varias funciones de pesado. La base de este esquema es que un vecino de menor distancia debe tener un peso mayor que uno de mayor distancia. La primera de las funciones de ponderación, consistía en un escalado lineal en función de la distancia de cada vecino. Así, el más cercano tendría un valor de 1, el más lejano tendría un valor de 0, y el resto tendría un peso con un valor normalizado de su distancia.
Además de la anterior, también se hace mención a la inversa de la distancia, y a una función de similaridad cuyos pesos se basan en la posición de cada vecino con respecto al resto.
Otra función de ponderación que podemos encontrar en la literatura, se basa en la obra de Shepard [75], la cual aboga por una ley universal de percepción que establece que la relevan-

[1]En adelante, se hablará de regla NN y kNN indistintamente, y también se hablará tanto de «el vecino más cercano», como de «los vecinos más cercanos» cuando se aluda a la regla o paradigma en cuestión.

cia de un estímulo previo para la generalización de uno nuevo, es una función exponencial decreciente de la distancia.
En trabajos más recientes como el de Parvin et al. [64] encontramos una propuesta llamada Modified kNN, en la que el peso de un punto del conjunto de entrenamiento, se calcula en base a la proporción de vecinos de igual etiqueta sobre el total de la vencidad.
Zeng et al. [85] definen un nuevo concepto para clasificar un punto sin etiqueta. El método introduce la idea de pseudo vecino, que se calcula en base a la suma pesada de las distancias para una clase dada. Cada vecino está dotado de un orden establecido por su distancia, siendo su peso la inversa de tal posición. Esto hace que los más cercanos tengan un valor mayor que los más lejanos. Después, el pseudo vecino con menor suma es elegido como el más cercano.
Z. Yong [86] propone un método de clustering para calcular los vecinos más cercanos. El proceso incluye la eliminación de todos los ejemplos del conjunto de entrenamiento que lindan con las fronteras de decisión. A continuación, se calculan los centroides, y cada punto es agrupado según el valor de k. Por último, cada cluster tendrá un peso en función del número de ejemplos que contenga.
En cuanto al procesamiento de vecinos para el aumento de la eficiencia, el algoritmo Condensed Nearest Neighbour propuesto inicialmente por Hart (CNN [41]) elimina aquellos que se consideran duplicados. El criterio a seguir es suprimir los puntos que muestren similaridad con otros datos de entrenamiento.
Gates et al. proponen una mejora de este método [32]. Reduced Nearest Neigbour (RNN) incluye uno o más pasos consistentes en la eliminación de aquellos patrones que no afectan al resultado de clasificar el resto de los datos de entrenamiento.
Por otro lado, para corregir las ineficiencias de la búsqueda tradicional del vecino más cercano, existen en la literatura multitud de propuestas basadas en estructuras de datos para mejorar el almacenamiento y búsqueda de los vecinos. Las más conocidas, son aquellas que se sustentan en árboles. En general, el objetivo de estas propuestas, es el de reducir la carga del cálculo de la distancia mediante la incorporación de información adicional sobre las instancias del conjunto de entrenamiento. La idea básica es que si un punto A es muy distante a otro punto B, y a su vez B es muy cercano a un tercero C, entonces A y C serán también muy distantes entre sí.
Una primera aproximación al uso de árboles en kNN, la encontramos en el uso de KD Trees [5]. El algoritmo que conforma esta estructura, realiza particiones de los datos a lo largo de los ejes de forma recursiva, de forma que dichos datos quedan divididos y agrupados en regiones afines.

Para paliar las deficiencias de los KD Trees con alta dimensionalidad, Omohundro [61], introduce el concepto de Ball Tree. A diferencia de los KD Trees, los Ball Trees particionan los datos en una serie de hiperesferas anidadas. Aunque el proceso de construcción es más costoso que en el caso anterior, la estructura resultante permite una búsqueda de vecinos más eficiente incluso con alta dimensionalidad.
Durante el proceso de generación del árbol, el Ball Tree divide recursivamente los datos en nodos definidos por un centroide y un radio, de tal manera que cada punto del nodo está contenido en la hiperesfera que representa. De esta manera, la distancia entre un punto de prueba y un centroide dado, es suficiente para determinar los límites inferior y superior de todos los puntos contenidos en la hiperesfera.
Más recientemente, Beygelzimer propone una estructura en árbol llamada Cover Tree [6]. La hipótesis de partida de este tipo de estructuras se basa en que los datasets (o los espacios métricos en los que están inmersos), muestran cierto crecimiento restringido o limitado independientemente del número de dimensiones. Se ha observado que tal pauta de crecimiento, ocurre por ejemplo en redes peer-to-peer y en Internet.
El árbol resultante, puede ser visto como una jerarquía de niveles, en la que los niveles más bajos contienen cada punto del espacio métrico. Cada nivel tiene asociado un número entero que decrece en una unidad en cada descenso. Además, deben cumplirse las siguientes propiedades: cada nivel está contenido en el anterior, y cada nodo sólo tiene un progenitor; la distancia entre dos puntos de dos niveles contíguos i e $i-1$, es menor o igual a 2^i; por último, todos los puntos de un mismo nivel i, tienen una distancia superior a 2^i entre sí.

2.3. Aprendizaje basado en pesos

Dentro del aprendizaje automático, existen dos ramas que definen perfectamente el uso de pesos en la creación de modelos de predicción. Éstas son las redes neuronales artificiales y las máquinas de vectores de soporte. El objetivo de esta sección es mostrar cómo el concepto de ponderación depende del contexto. Así, en las redes neuronales artificiales los pesos se establecen en las interconexiones entre neuronas, mientras que en las máquinas de vectores de soporte, definen los hiperplanos de decisión. En ambos casos, el sistema de ponderación forma parte del proceso de aprendizaje en sí, mientras que en nuestro trabajo (y en la regla kNN en general) es una herramienta de mejora. Es por eso que hemos relegado a la sección 2.4 todos los detalles concernientes al uso de pesos en los vecinos más cercanos.
A continuación, describimos las características más importantes de dos de las vertientes más clásicas del aprendizaje ponderado.

2.3.1. Redes neuronales artificiales

Las redes de neuronas artificiales (RNA o en inglés ANN), son quizás el paradigma más representativo de aprendizaje basado en pesos. Sus fundamentos teóricos se basan en el funcionamiento del sistema nervioso de los animales, esto es, un sistema interconectado de neuronas que conforma una red colaborativa, y que es capaz de producir un estímulo de salida.

Es en el año 1943 cuando los neurólogos McCulloch y Pitts, describen los primeros modelos RNA [56]. Posteriormente, Hebb continuó ampliando los fundamentos del aprendizaje neuronal, y formalizó los resultados con la «regla de Hebb» [43]. En 1958, Rosenblatt desarrolló el perceptrón simple [70], y más tarde, Widrow y Hoff presentaron la primera aplicación real que denominaron ADALINE [84].

Pero no fue hasta la década de los 80, cuando se generalizó el uso y diseño de RNAs gracias en gran parte al trabajo de Hopfield (red de Hopfield)[45], y Rumelhart y McLelland (algoritmo de retropropagación) [71].

Las unidades básicas de una RNA se denominan neuronas. Cada neurona recibe un conjunto de entradas y devuelve una salida. A su vez, cada salida está condicionada a tres tipos de funciones. La función de propagación (o de excitación), es un sumatorio de cada entrada multiplicada por el peso de su interconexión. La función de activación es la encargada de modificar a la anterior, pero puede ser prescindible. Por último, la función de transferencia, se aplica al valor devuelto por la función de activación si existiese, o a la de propagación en caso contrario, y se utiliza para acotar la salida de la neurona con el fin de dotarla de un valor interpretable. Entre las funciones de transferencia más frecuentes, encontramos la sigmoidea (en el intervalo [0,1]) y la tangente hiperbólica (en el [-1,1]).

En cuanto a la tipología de RNAs, podemos clasificarlas en función de la topología de la red, del tipo de aprendizaje o del tipo de entrada (continua o discreta).

Si nos centramos en la estructura de la red, podemos definir tres tipos básicos de RNAs. Redes monocapa (p. ej. perceptrón [70], ADALINE [84]), multicapa (p. ej. perceptrón multicapa [71]) y redes recurrentes (p. ej. Elman [20], Hopfield [45], máquina de Boltzmann [44]). Las dos primeras (redes monocapa y multicapa) se caracterizan por propagarse hacia delante, en el sentido de que todas las señales avanzan desde la capa de entrada hacia la de salida, sin existir ciclos ni interconexiones entre neuronas de la misma capa. Las redes recurrentes por contra, presentan ciclos cerrados de activación neuronal.

En relación al modelo de aprendizaje, éste puede ser supervisado cuando la RNA necesita un conjunto de entrada cuya respuesta es conocida a priori (p. ej. perceptrón simple [70], ADALINE [84], perceptrón multicapa [71], red backpropagation [71]), o autoorganizado en

caso contrario (p. ej. memorias asociativas [42], red de Hopfield [45], máquina de Boltzmann [44], máquina de Cauchy [79], red de aprendizaje competitivo [47], red de Kohonen [48], red de resonancia adaptativa [35]).
Además de los dos casos anteriores, también existen propuestas híbridas en las que se utiliza una función para mejorar la convergencia (p. ej. red de base radial [58]).
Por último, podemos clasificar las RNAs en función del tipo de datos que son capaces de procesar. Cuando los valores son continuos y normalmente acotados, las redes se denominan analógicas (p. ej. Hopfield [45], Kohonen [48], aprendizaje competitivo [47]). Por contra, las redes discretas son aquellas que soportan datos de naturaleza categórica, normalmente de tipo booleano (p. ej. máquinas de Boltzmann y Cauchy [44, 79], red discreta de Hopfield [45]).

2.3.2. Máquinas de vectores de soporte

Otra categoría clásica en el conjunto de clasificadores ponderados, son las Máquinas de Vectores de Soporte (MVS o SVM en inglés). Las MVS fueron desarrolladas inicialmente por Vapnik y su equipo en los laboratorios AT&T [13], y pertenecen a la familia de los clasificadores lineales. También pueden ser vistos como un caso particular de la regularización de Tikhonov y están estrechamente relacionadas con las RNAs.
La idea que subyace en este conjunto de técnicas, radica en la separación máxima de las fronteras de decisión, de manera que sea más fácil la clasificación de una nueva instancia en función de la región del modelo a la que pertenezca. Para conseguirlo, se construye uno o más hiperplanos en un espacio de dimensionalidad muy alta, que podría incluso ser infinita. Si se consigue una buena separación entre clases, el modelo será apto para clasificar correctamente instancias de etiqueta desconocida.
Este concepto de «separación óptima» es la base conceptual de las MVS. Debido a que este tipo de técnicas busca el hiperplano que tenga la máxima distancia (o margen) con respecto a los puntos más cercanos, las MVS también son conocidas como clasificadores de margen máximo.
Otro aspecto a tener en cuenta, es la terminología utilizada. A menudo la literatura sobre aprendizaje automático, no hace distinción entre atributo y característica cuando se refiere a uno de los elementos del vector que representa una instancia de un conjunto de datos. En las MVS, un atributo es una variable predictora, mientras que una característica es un atributo transformado, que se usa para definir un hiperplano. Al vector formado por los puntos más cercanos al hiperplano, se le llama vector de soporte.
Uno de los puntos a favor de las MVS, es su simplicidad conceptual. Así, en función de

la dimensión del espacio en observación, la manera más sencilla de realizar una separación entre dos regiones adyacentes, es a través de una línea recta, un plano recto o un hiperplano N-dimensional. Pero en dominios reales, es común tratar con curvas no lineales de separación, categorías no disjuntas, y múltiples clases. Este es el principal problema de los clasificadores lineales.
Para salvar los inconvenientes inherentes al mundo real, se pensó en la utilización de funciones kernel. Una función kernel, es capaz de proyectar la información a un espacio de características de mayor dimensión, lo que facilita un mayor rendimiento de los sistemas de aprendizaje lineal.
Más formalmente, Una función núcleo o kernel es un producto interno en el espacio de características, que tiene su equivalente en el espacio de entrada [38]:

$$K(x, x') =< \Phi(x), \Phi(x') > \tag{2.2}$$

donde K, es una función simétrica positiva definida que cumple las condiciones de Mercer. Entre las funciones kernel más representativas se encuentran la polinomial homogénea, el perceptrón, la función de base radial Gaussiana y la sigmoide.
Aunque las MVS están pensadas inicialmente para datos con sólo dos clases, es posible aplicarlas a problemas con más categorías. Las dos líneas principales de resolución se basan en dividir los datos en dos grupos varias veces, y aplicar un clasificador por cada división. Posteriormente, para la elección de la etiqueta será necesario adoptar alguna estrategia de consenso (p. ej. voto mayoritario). La primera de las líneas, establece grupos que enfrenta a los datos de una sola clase con el resto. Habrá un par de subconjuntos por cada clase. La segunda línea, realiza cada división contemplando sólo dos de las clases cada vez. En ese caso, habrá tantas divisiones como combinaciones de las clases dos a dos.

2.4. Búsqueda de pesos para los vecinos más cercanos

Según se ha comentado en el capítulo de introducción, los métodos de pesado pueden clasificarse como globales o locales, atendiendo al ámbito de aplicación de los datos en consideración. El objetivo fundamental en ambos casos es el mismo: transformar el rango de la función de similaridad utilizada en el cálculo de los vecinos. Esta transformación se consigue principalmente, mediante la ponderación de las características de los datos en función de su importancia. Pero, qué entendemos por importancia. O mejor dicho, con respecto a qué elementos una característica puede ser considerada como más relevante que otra. A nivel

general, una característica puede ser más importante que otra para todas las categorías que existan en los datos bajo estudio. O también, una característica puede ser más importante que otra dependiendo de la clase, o incluso de la instancia o prototipo. Es lógico pensar, que a mayor especialización, mejor descripción, y dado que el problema fundamental radica en el tamaño del espacio de búsqueda y el aumento de complejidad en los datos, la solución elegida vendrá determinada principalmente por los recursos disponibles, y el alcance de la tecnología del momento.

Si nos centramos en métodos globales, Raymer et al. presentan una propuesta para seleccionar y eliminar características mediante un algoritmo genético basado en el vecino más cercano [69]. Este algoritmo optimiza un vector de pesos que es usado para escalar las características originales. Además de esto, también utilizan un vector de bits para seleccionar atributos simultáneamente. En un trabajo posterior, muestran un algoritmo evolutivo híbrido basado en la función discriminante de Bayes [68]. La finalidad de esta propuesta, es la de aislar características pertenecientes a grandes datasets de origen biomédico seleccionando y extrayendo atributos relevantes.

Además de los métodos basados en computación evolutiva, también existen otras heurísticas en la literatura. Así, Tahir et al. presentan una propuesta híbrida para simultáneamente seleccionar atributos y aplicarles pesos mediante búsqueda tabú, y de esa manera mejorar la tasa de clasificación de los k vecinos más cercanos [80].

Si nos centramos en el pesado de regiones de separación, Fernández et al. proponen un sistema de ponderación local en conjunción con un clasificador basado en prototipos [25]. Los pesos se calculan iterativamente después de normalizar los datos. Esta normalización se basa en la posición de las instancias con respecto al prototipo (o región) al que pertenecen.

Continuando con métodos locales, Alsukker et al. usan evolución diferencial como técnica de optimización para buscar pesos que afectan a diferentes elementos de los datos de entrenamiento [3]. En su trabajo se describen cuatro enfoques: pesado de atributos, pesado de vecinos, pesado de clases y pesado mixto (atributos y clases). Es en éste último donde se obtienen los resultados más satisfactorios. Así, el método de pesado mixto obtiene un vector de pesos mediante la concatenación de un vector que contiene un peso por cada atributo, y otro que contiene un peso por cada clase.

Mohemmed et al. presentan un clasificador bajo el paradigma del centroide más cercano [57]. La base de este método, es la búsqueda de instancias prototípicas que son calculadas a partir de las instancias de los datos de entrenamiento, y tomando como medida de concentración la media aritmética. Cuando una instancia de etiqueta desconocida debe ser clasificada, se calcula la distancia de ésta a cada prototipo. Para buscar aquellos centroides

que minimizan el error de clasificación, se utiliza un enjambre de partículas como técnica de optimización. Por otro lado, una de las funciones de similaridad utilizadas en este trabajo, consiste en una versión de la distancia euclídea que depende de la etiqueta de una de las instancias que participan en el cálculo de dicha distancia. Esta función, es introducida por Paredes et al. y también es utilizada en nuestro trabajo [62]. El objetivo de dicha función, es el de mejorar el comportamiento del vecino más cercano.
Una primera aproximación de la propuesta de Paredes et al., considera un peso por cada característica e instancia para los datos de entrenamiento, lo que hace poco viable el número de parámetros a tener en consideración en el proceso de aprendizaje. De esta manera, los autores presentan tres tipos de reducción: un peso por clase y característica (dependencia de clase), un peso por prototipo (dependencia de prototipo) y una combinación de los dos anteriores. La fase de optimización se lleva a cabo mediante gradiente descendiente.
Existen también referencias al procesamiento de datos desbalanceados [52]. Liu et al. definen una medida llamada Class Confidence Weight (CCW) que mide la probabilidad de que el valor de un atributo pertenezca a una clase. La estimación de CCW se realiza mediante modelos de mezcla para atributos numéricos, y con redes bayesianas para los categóricos.

Tal y como se ha descrito en los diversos métodos mencionados, podemos resumir que para mejorar la regla del vecino más cercano, se puede aplicar un conjunto de pesos, ya sea de manera global o de forma local, a una función de similaridad. En nuestro trabajo, hemos analizado la última opción. A grandes rasgos los objetivos son dos. El primero de ellos, consiste en alcanzar un equilibrio entre complejidad y mejor ajuste. Y en segunda instancia, mostrar objetivamente la efectividad de un método local de pesado basado en los k vecinos más cercanos. Para el primer objetivo, hemos elegido un diseño basado en computación evolutiva, con el fin de obtener una matriz de pesos (un peso por cada clase y característica), y el mejor valor del parámetro k. Para el segundo fin, hemos evaluado los resultados experimentales con rigurosos tests estadísticos, que ya han sido utilizados en la literatura [16, 30]. Todo estos aspectos, serán expuestos y detallados a lo largo de la memoria.

3

Algoritmos evolutivos

> «*Extinction is the rule. Survival is the exception.*»
>
> – Carl Sagan, *The Varieties of Scientific Experience.*

Existen multitud de variantes bajo el paradigma de la computación evolutiva (CE). Sin embargo, la idea que subyace en todas estas técnicas es la misma: dada una población de individuos soluciones a un problema, se establece una selección natural marcada por la presión del entorno (sobreviven los más aptos), y esto produce una mejora en la aptitud de la población con el paso de las generaciones.

Dada una función que debe ser maximizada, se deberán crear un conjunto de soluciones candidatas, esto es, elementos pertenecientes al dominio de dicha función (también llamados individuos), y posteriormente aplicar la función como medida abstracta de bondad o calidad (el valor más alto, será la mejor solución).

En base a esta medida de bondad, algunos de los mejores candidatos serán elegidos para poblar la siguiente generación a través de operaciones de recombinación y/o mutación sobre ellos. La recombinación es una operación aplicada a dos o más candidatos seleccionados (también llamados progenitores), que da como resultado uno o más candidatos denominados hijos. Por contra, el operador de mutación es aplicado a un solo candidato produciendo otro nuevo.

La aplicación de recombinaciones y mutaciones, da como resultado un conjunto de nuevos candidatos (la descendencia). Esta descendencia, competirá con el resto de individuos (en base a su aptitud) por un lugar en la siguiente generación. Este proceso, continuará de forma iterativa hasta que un candidato tenga una calidad suficiente (una solución), o hasta un límite establecido (p. ej., un número de generaciones).

Durante este proceso, existen dos fuerzas fundamentales que conforman la base de los siste-

mas evolutivos. Por un lado, los operadores de variación (recombinación y mutación), que hacen posible la diversidad necesaria en la generación de candidatos. Y por otro el operador de selección, que actúa como un filtro de calidad. La combinación de los operadores de variación y selección, ayudan por lo general a mejorar los valores de aptitud, y consecuentemente las poblaciones.

Intuitivamente podemos observar, cómo poco a poco, el proceso de evolución va alcanzando, o al menos se va aproximando, a los valores óptimos deseados.

Alternativamente, la evolución puede ser vista como un proceso de adaptación. Desde este punto de vista, la aptitud no es considerada como una función a ser optimizada, sino más bien como una expresión de unos requisitos que responden a un entorno determinado. A medida que estos requisitos se van alcanzando generación tras generación, la población se va adaptando cada vez más a dicho entorno.

Nótese que la mayoría de los componentes del proceso evolutivo son estocásticos. Durante la selección, los individuos más aptos tienen más probabilidad de ser elegidos que los de menor adaptación. No obstante, eso no implica que incluso individuos considerados débiles, no tengan la oportunidad de convertirse en progenitores o incluso de sobrevivir. Por otro lado, durante el proceso de recombinación, la elección de los fragmentos de cada candidato es aleatoria. Y también ocurre lo mismo en la operación de mutación. Las partes que son alteradas en un individuo, y los nuevos valores que se almacenarán en ellas, son elegidos aleatoriamente.

En la Figura 3.1, podemos ver el esquema general de un algoritmo evolutivo.

```
INICIO
  INICIALIZAR población con soluciones candidatas aleatorias;
  EVALUAR cada candidato;
  REPETIR HASTA (CONDICIÓN DE TERMINACIÓN sea satisfecha)
    1 SELECCIONAR progenitores;
    2 RECOMBINAR pares de progenitores;
    3 MUTAR descendencia resultante;
    4 EVALUAR nuevos candidatos;
    5 SELECCIONAR candidatos para próxima generación;
  FIN REPETIR
FIN
```

Figura 3.1: Esquema general de un algoritmo evolutivo.

Podemos observar que este esquema se engloba dentro de la categoría de algoritmos «generate-and-test». La función de evaluación representa una estimación heurística de la

calidad de una solución, y el proceso de búsqueda está conducido por los operadores de variación y selección. Los Algoritmos Evolutivos (AEs) poseen un número de características que lo ayudan a posicionarse dentro de la familia de métodos «generate-and-test»:

1. Los AEs están basados en una población, esto es, procesan una colección de soluciones candidatas,
2. Generalmente usan recombinación para sintetizar la información de varios candidatos en uno nuevo,
3. Son estocásticos

Todas las variantes de la computación evolutiva, siguen las mismas líneas generales explicadas hasta ahora, y difieren sólo en detalles de implementación. Por ejemplo, si nos centramos en la representación de los individuos (la manera de codificar una solución) podemos encontrarnos un conjunto de símbolos perteneciente a un alfabeto cuando hablamos de Algoritmos Genéticos (AGs). O vectores con codificación real en las Estrategias Evolutivas (EEs). Otro ejemplo, podría ser máquinas de estado finito si nos referimos a la Programación Evolutiva clásica (PE), o árboles para sistemas de Programación Genética (PG).
Todas estas diferencias tienen en común su origen histórico. Técnicamente, una representación será preferible a otra cuando se ajuste mejor a un problema dado, o lo que es lo mismo, cuando dicha representación facilite la codificación de un candidato haciéndolo más natural. Por ejemplo, para resolver un problema de satisfacibilidad, la elección más adecuada sería usar un algoritmo genético y una cadena de bits de longitud n, donde n sería el número de variables lógicas.
Es importante apuntar, que los operadores de recombinación y mutación dependen de la representación de los individuos. Así, para una instancia en un diseño basado en programación genética, la operación de recombinación debe estar preparada para trabajar con árboles, mientras que en los algoritmos genéticos, debe estarlo para operar con cadenas.
A diferencia de los operadores de variación, la selección sólo debe tener en cuenta la aptitud de cada candidato, y por tanto, no depende de su codificación. Si existen diferencias entre los mecanismos de selección de los diferentes enfoques del paradigma evolutivo, se debe más bien a una decisión tradicional de diseño, que a una necesidad técnica.

3.1. Componentes de un algoritmo evolutivo

En esta sección describiremos el funcionamiento de los Algoritmos Evolutivos en detalle. Los AEs tienen un número de componentes, procedimientos u operadores que deben

ser especificados para definirlos correctamente. Los más importantes son la representación (definición de individuos), la función de evaluación (o función de bondad), el mecanismo de selección y los operadores de variación (recombinación, mutación y reemplazo).
Cada uno de estos componentes definen un particular tipo de AE. Además de lo anterior, es necesario definir un procedimiento de inicialización y una condición de terminación.

3.1.1. Representación (Definición de los individuos)

El primer paso a tener en cuenta en un AE, es relacionar el «mundo real» con el diseño del AE, o lo que es lo mismo, establecer un puente entre el contexto del problema y su resolución. Los objetos que conforman las posibles soluciones del problema a considerar, se definen como fenotipos, y su codificación a través de los individuos es denominada genotipos.
La primera decisión de diseño es comúnmente llamada representación, y consiste en especificar cómo los fenotipos deben ser mapeados hacia un conjunto de genotipos que representan a tales fenotipos. Por ejemplo, si tenemos un problema de optimización sobre números enteros, el conjunto de enteros debería formar el conjunto de fenotipos. En este punto, se podría decidir que su representación viniera dada por su codificación binaria. Así, el 18 podría ser representado por el genotipo 10010.
Es importante resaltar que el espacio fenotípico podría ser muy diferente del espacio genotípico, y que la búsqueda del AE se centraría en los genotipos. Una solución (un buen fenotipo) se obtendría decodificando el mejor genotipo obtenido tras la terminación del proceso evolutivo.
En la terminología relativa a la CE, se suelen usar muchos sinónimos para referirse a los elementos de ambos espacios. En el lado del contexto del problema, se suele hablar indistintamente de solución candidata, fenotipo e individual para denotar puntos del espacio de soluciones. Este espacio es conocido normalmente como espacio fenotípico. En el lado del AE, es común utilizar los términos genotipo, cromosoma, y de nuevo individual para referirse a puntos del espacio de búsqueda. Este espacio es a menudo denominado espacio genotípico.
También existen muchos términos para referirse a los elementos que integran los individuos. Un segmento de inviduo es comúnmente llamado variable, locus (loci en plural), posición o gen. Un objeto que ocupa un segmento, puede ser llamado valor o alelo.
Es necesario apuntar, que la palabra «representación» se usa de dos formas ligeramente distintas. A veces, es referida al mapeo entre el espacio fenotípico y el genotípico. En este sentido, la semántica es sinónima de codificación. Por ejemplo, si hablamos de representa-

ción binaria, también podríamos referirnos a codificación binaria de soluciones candidatas. El mapeo inverso de genotipos a fenotipos, es generalmente denominada decodificación. Esto último evidencia la necesidad de que la representación sea invertible: para cada genotipo debe haber al menos un fenotipo que le corresponda.
La palabra «representación» también puede referirse a la estructura de los datos del espacio genotípico y no tanto al mapeo. Esta última interpretación, está presente en los operadores de mutación cuando hablamos por ejemplo de representación binaria o real.

3.1.2. Función de evaluación (Función de bondad)

El papel de la función de evaluación consiste en representar los requerimientos a los que hay que adapatarse. Al conformar las bases de la selección, facilita la mejora por evolución. Más concretamente, la función de bondad define el significado de mejora. Desde el punto de vista de la resolución de un problema, representa la tarea a resolver. Técnicamente, es una función o procedimiento que asigna una medida de calidad a los genotipos. Por ejemplo, si quisiéramos maximizar x^2 en el dominio entero, la bondad del genotipo 10010 podría definirse como el cuadrado de su correspondiente fenotipo: $18^2 = 324$.
A la función de evaluación, se le conoce comúnmente como función de bondad en CE. Esto es debido a que dicha función está frecuentemente asociada a problemas de maximización. Por tanto, el uso del término «bondad», puede causar cierta confusión cuando el problema a resolver requiere minimización. No obstante, pasar de una tarea de minimización a una de maximización y viceversa, es trivial desde un punto de vista matemático.
Muy a menudo, el problema a resolver mediante un AE es un problema de optimización. En este caso, es frecuente la utilización del término «función objetivo» y la función de evaluación (o de bondad) es normalmente idéntica, o es una transformación simple de la función objetivo.

3.1.3. Población

La población tiene como objetivo, almacenar un conjunto de posibles soluciones al problema. Una población es un multiconjunto de genotipos y forma una unidad de evolución. De hecho, es la población (y no los individuos de manera aislada) la que se adapta y evoluciona generación tras generación. Dada una representación, definir una población puede ser tan simple como especificar cuántos individuos habrá en ella (establecer su tamaño). En algunos AEs más sofisticados, la población tiene una estructura espacial adicional basada en una medida de distancia, o relación de vecindad. En estos casos, es necesario definir tal

estructura para especificar completamente la población.
En general, el tamaño de la población es constante y no cambia durante la búsqueda evolutiva. La diversidad de dicha población, es una medida del número de soluciones diferentes que existen en ella. No hay una una medida única para definir la diversidad, pero generalmente puede entenderse como el número de diferentes valores de bondad, el número de diferentes fenotipos, o el número de diferentes genotipos. También es posible utilizar otras medidas estadísticas como la entropía. Nótese que un único valor de bondad, no implica necesariamente un único fenotipo relacionado, y que un único fenotipo, no implica necesariamente un único genotipo. Sin embargo un genotipo sí que implica un único fenotipo y un único valor de bondad.
En cuanto a las operaciones sobre un AE, a diferencia de los operadores de variación, que actúan sobre uno o dos individuos progenitores, los operadores de selección (selección de padres y de supervivientes) trabajan a nivel de población. Por ejemplo, el mejor individuo de una población dada, puede ser elegido para sobrevivir a la siguiente generación, o el peor individuo puede ser elegido para ser reemplazado por uno nuevo.

3.1.4. Selección de progenitores

El papel de la selección, es el de distinguir entre individuos en base a su calidad, y de esta manera, conseguir que los mejores individuos se conviertan en padres de la siguiente generación. Los individuos padres, aportarán la información necesaria para la creación de descendencia. Junto al mecanismo de selección de supervivientes, la selección de progenitores será la responsable de la mejora en la calidad generacional. En CE, la selección de padres es típicamente probabilística. Así, los individuos de mayor calidad tienen más posibilidades de convertirse en progenitores que los de menos. Sin embargo, los individuos de menor calidad, también tienen una oportunidad (aunque menor) de convertirse en ascendiente. De no ser así, la búsqueda podría llegar a ser demasiada elitista y quedar atrapada en un óptimo local. Así ocurre por ejemplo en la selección de ruleta, en la que la probabilidad de que un individuo sea elegido es proporcional a su importancia, o también, en la selección por torneo, en la que la se eligen los progenitores a partir de un subconjunto de individuos elegidos al azar.

3.1.5. Operadores de variación

Los operadores de variación, permiten crear nuevos individuos a partir de otros existentes. En el correspondiente espacio fenotípico, esto permite la creación de nuevas soluciones

candidatas. Los operadores de variación se dividen en dos tipos basados en su aridad (número de objetos que el operador toma como entrada). Así, los operadores unarios son en general denominados mutación, mientras que los binarios reciben el nombre de recombinación o cruce.
Es importante tener en cuenta, que los operadores de variación son dependientes de la representación. Esto significa que para diferentes representaciones, deben definirse diferentes operadores. Por ejemplo, si los genotipos lo integran cadenas de bits, se puede utilizar como operación de mutación la inversión de 0 a 1 y viceversa. Si por el contrario, las soluciones candidatas se representan como estructuras arbóreas, entonces es necesario diseñar un operador de mutación más sofisticado.

3.1.5.1. Mutación

Un operador de variación unario, es comúnmente denominado mutación. Al aplicarse a un individuo, genera ligeras modificaciones en su genotipo. El operador de mutación es siempre estocástico: su salida depende de los resultados de una serie de elecciones al azar. Es necesario apuntar, que un operador unario arbitrario no es necesariamente un operador de mutación. En problemas específicos, podrían existir operadores heurísticos que actuasen sobre un solo individuo, y ser contemplados como mutación por el hecho de ser unarios. En general, la mutación tan sólo debe causar un cambio aleatorio e imparcial sobre el individuo.
El papel del operador de mutación es diferente para distintos escenarios de CE. Así, en PG no se usa en general; en AGs, se usa normalmente como un operador secundario que proporciona nuevas combinaciones de genes; mientras que en PE es el único operador de variación existente en el proceso de búsqueda.
Hay que recalcar que los operadores de variación, conforman la implementación de los pasos elementales en el espacio de búsqueda. La generación de un individuo, se puede ver como un avance hacia un nuevo punto dentro de dicho espacio. Desde esta perspectiva, la mutación tiene también un papel teórico: garantizar que el espacio esté conectado. Esto es importante en la medida de que, según algunos estudios[19], el descubrimiento de un óptimo global por parte de un AE, se basa en la propiedad de que cada genotipo que representa una posible solución, debería poder ser alcanzado por los operadores de variación. La manera más sencilla de satisfacer dicha propiedad, consiste en permitir que cualquier alelo de un individuo, pueda mutar en cualquier otro con una probabilidad distinta de cero. Sin embargo, también debe tenerse en cuenta que muchos investigadores creen que estas pruebas tienen una importancia práctica limitada, y muchas implementaciones de AEs, no poseen esta

propiedad.

3.1.5.2. Recombinación

Un operador de variación binario, es conocido como recombinación o cruce. Como su propio nombre indica, este operador combina información genotípica a partir de dos padres. Al igual que la mutación, la recombinación es un operador estocástico: la elección de qué partes de cada padre es combinada, y la manera en que se hace, depende de sorteos al azar. También, el papel de la recombinación es diferente en los distintos enfoques de la CE: en PG, es con frecuencia el único operador de variación; en AGs, se puede considerar el operador principal; y en PE, no se usa. Los operadores de recombinación de alta aridad (con más de dos padres) son matemáticamente factibles y fáciles de implementar, aunque no tienen equivalencia biológica. Es por ello que, en general, no se suelen utilizar, aunque existen estudios que revelan efectos positivos en el proceso evolutivo [18]. El principal objetivo de la recombinación es simple: el emparejamiento de dos individuos con diferentes pero deseables características, puede producir hijos que las combinan. Esta máxima, ha sido aplicada con éxito durante miles de años por criadores de plantas y animales, para producir especies que dan un mayor rendimiento o tienen otras características deseables. Los AEs, crean un número de hijos por recombinación aleatoria aceptando que algunos de ellos, tendrán combinaciones de rasgos no deseados, otros no serán ni mejor ni peor que sus progenitores, y algunos serán mejores.

3.1.6. Mecanismo de selección de supervivientes (reemplazo)

El objetivo de la selección de supervivientes, consiste en la distinción entre individuos en base a su calidad. Si bien comparte similaridades con la selección de progenitores, este operador es utilizado en una fase diferente del ciclo de vida del AE. El mecanismo de reemplazo, es usado después de la creación de la descendencia a partir de la población de una generación dada. Tal y como se ha mencionado en 3.1.3, el tamaño de la población suele ser constante, por lo que es necesario establecer qué individuos formarán parte de la próxima generación. Esta decisión está basada normalmente en los valores de bondad, favoreciéndose a aquellos individuos mejor adaptados, aunque también podemos encontrar diferenciaciones en base a conceptos como el de «edad».
A diferencia de la selección de progenitores, que suele tener un carácter estocástico, la selección de supervivientes es a menudo determinista. Una práctica común es la de unificar el conjunto de padres y el de hijos (o sólamente considerar el de hijos), y posteriormente

seleccionar los n mejores.
A menudo, la selección de supervivientes es conocida como estrategia de reemplazo. La elección de un término u otro viene dada por la proporción entre el número de descendientes y el tamaño de la población. Si el número de hijos es muy pequeño (por ejemplo 2 hijos en una población de 100 individuos), entonces es más eficiente reemplazar los individuos más antiguos y asumir que casi todo el mundo sobrevive. Si se genera mucha descendencia (por ejemplo 500 hijos), entonces es necesario establecer cuáles de ellos sobrevivirán a la siguiente generación.

3.1.7. Inicialización

La inicialización suele llevarse a cabo de una manera muy simple en la mayoría de los AEs: la primera población es generada aleatoriamente; aunque también es factible la utilización de heurísticas específicas al problema a optimizar con el fin de conseguir una población inicial de cierta calidad. Esto último requiere un esfuerzo computacional extra que merecerá la pena o no en función de la aplicación en cuestión.

3.1.8. Condición de terminación

Podemos distinguir dos casos a la hora de elegir una condición de terminación. Si el problema tiene una cota de bondad conocida, ésta podría ser considerada como condición de terminación. Sin embargo, dada la naturaleza estocástica de los AEs, a veces no hay garantías de poder alcanzarse un óptimo. Esta situación requiere una condición adicional de finalización, que asegure la convergencia del algoritmo. Algunas de las opciones más comunes para este propósito son las siguientes:

1. tiempo máximo transcurrido,
2. número total de generaciones,
3. durante un periodo de tiempo dado (es decir, para un número de generaciones), la mejora de la bondad permanece debajo de un valor umbral,
4. la diversidad de la población cae por debajo de un umbral dado

En los casos anteriores, el criterio de parada es una disyunción: se alcanza el valor óptimo o se satisface la condición x. Si el problema no tiene un óptimo conocido, no es necesaria tal disyunción.

4

Algoritmos evolutivos paralelos

«No hay un único mundo. Sino muchos mundos, y todos discurren en paralelo... Cada mundo es la creación de un individuo.»

– Paul Auster, *Un hombre en la oscuridad.*

A medida que avanza la tecnología y la disponibilidad de información, las exigencias en el procesamiento de datos también van en aumento. En general, el espacio de búsqueda en problemas de optimización, crece exponencialmente con el número de variables. Un ejemplo lo encontramos en la propuesta descrita a lo largo de esta memoria, donde se utiliza un algoritmo evolutivo para mejorar la regla de los vecinos más cercanos (ver capítulo 5 para más detalle). Uno de los principales problemas que plantea el uso de algoritmos evolutivos es el tiempo de procesamiento, sobre todo si tanto la función objetivo como el genotipo, deben soportar datos reales de alta dimensionalidad como es nuestro caso.
En vista de lo expuesto, los sistemas paralelos y distribuidos son sin duda la solución más plausible para satisfacer las exigencias de las aplicaciones de alto rendimiento.
Las siguientes secciones de este capítulo, introducen los conceptos de computación concurrente y distribuída, y su aplicación a los algoritmos evolutivos. Para tal fin, el apartado 4.1 comienza con la descripción de las arquitecturas típicas de multiprocesamiento, mientras que los detalles de la computación evolutiva paralela, serán descritos en la sección 4.2.

4.1. Computación paralela y distribuída

Podemos definir el procesamiento paralelo, como el procesamiento de información, en el que se acentúa la manipulación concurrente de elementos pertenecientes a uno o más procesos, que son capaces de resolver un solo problema [67]. Los computadores capaces de

realizar este tipo de procesamiento, se denominan computadores paralelos, y se caracterizan principalmente por tener un conjunto de procesadores.

Aunque existen varios esquemas que definen la computación paralela [29, 4, 81], ninguno se ha constituido en un estándar. Las diferencias entre estos esquemas, están dirigidas por la arquitectura de cada uno de ellos. Así podemos encontrar diferencias en la organización del espacio de direcciones, en la red de interconexión, o en la granularidad de los procesadores.

Los computadores paralelos, se pueden dividir en las categorías SIMD (Single Instruction, Multiple Data) y MIMD (Multiple Instruction, Multiple Data). Estas categorías, forman parte de la conocida taxonomía propuesta por Flynn [29].

Los computadores SIMD, se caracterizan por el procesamiento de una misma operación sobre diferentes datos, pero de manera simultánea (una instrucción es aplicada a un vector de datos). De esta manera, este tipo de arquitecturas explota el paralelismo a nivel de datos (Figura 4.1). Las CPUs actuales están diseñadas para incluir instrucciones SIMD, con el fin de mejorar el rendimiento de los procesos multimedia.

El principal problema de este paradigma, es que la mayoría de los compiladores no generan instrucciones SIMD. De hecho, la vectorización en procesadores de lenguajes, es actualmente una linea activa de investigación. Aún así, existen algunas propuestas basadas en computación evolutiva, que explotan las características SIMD de los procesadores gráficos (ver [49]).

Los computadores MIMD tienen varios procesadores que trabajan coordinados a través de

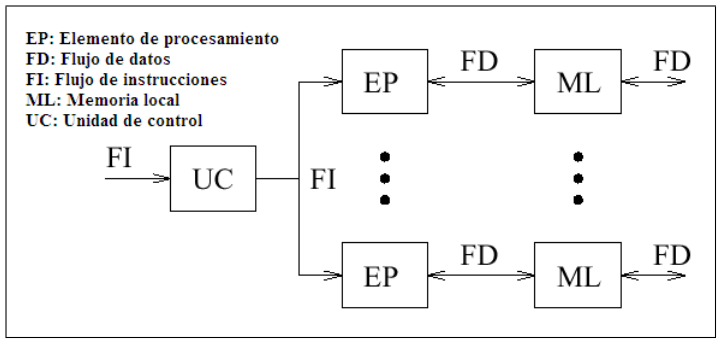


Figura 4.1: Arquitectura SIMD

una red de interconexión. Cada procesador es capaz de ejecutar un proceso distinto independientemente de los demás (Figura 4.2). Atendiendo a la organización del espacio de direcciones, los sistemas MIMD se subdividen a su vez en dos categorías: sistemas de memoria compartida, o multiprocesador; y sistemas de memoria distribuida, llamados comúnmente multicomputador.

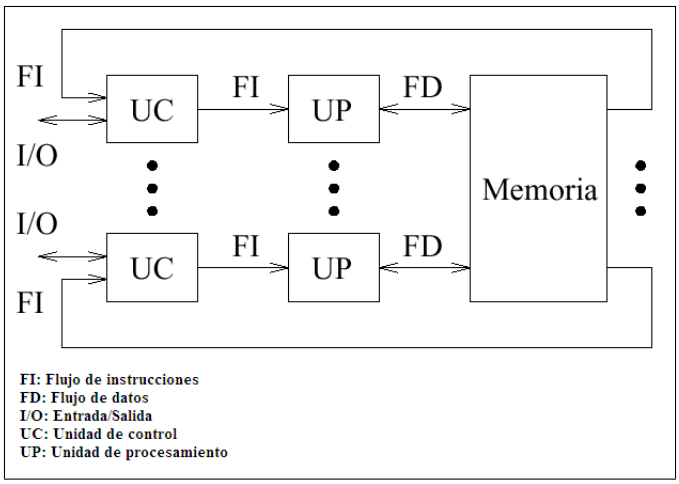


Figura 4.2: Arquitectura MIMD

Los sistemas multiprocesador se caracterizan por leer y escribir en un solo espacio de direcciones físico, a través de una red de interconexión que habitualmente tiene forma de bus (Figura 4.3). De esta manera, la memoria compartida permite a los procesadores escribir datos, para que después puedan ser leídos por el resto.

Los principales problemas de este esquema, se centran en la vulnerabilidad de la integridad de los datos, y en la degradación de la eficiencia. Para evitar conflictos de escritura el programador debe usar los mecanismos clásicos de sincronización (p. ej. semáforos). Por otro lado, la eficiencia puede ser mejorada mediante el uso de una memoria caché por procesador. De esta forma, se podría mejorar el tiempo de acceso a la información más frecuente. No obstante, la incorporación de estas memorias, requiere de protocolos que aseguren la coherencia entre la memoria principal, y la memoria caché de cada procesador.

En los sistemas multicomputador, cada procesador tiene su propia memoria, de tal forma

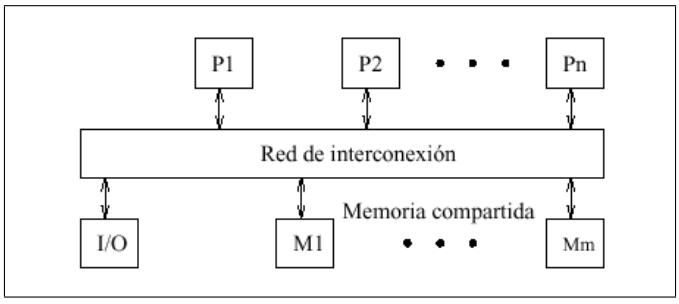


Figura 4.3: MIMD Multiprocesador

que ésta sólo puede ser accedida por el procesador correspondiente (Figura 4.4). En este

caso, la comunicación entre los procesadores de la red se produce mediante el envío de mensajes (message passing). A esta categoría pertenecen los computadores nCUBE 2, Paragon

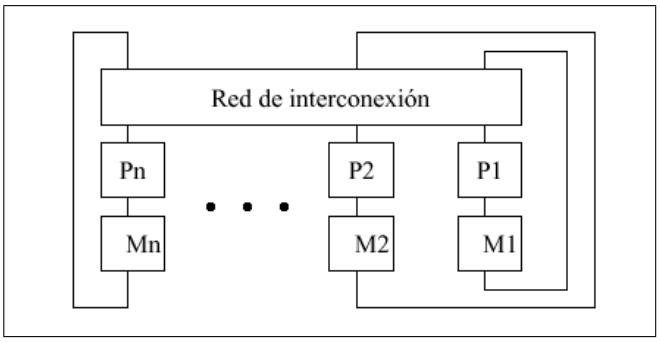


Figura 4.4: MIMD Multicomputador

XP/S, Cosmic Cube, y CM-5, que se caracterizan por ser sistemas fuertemente acoplados. También pertenecen a este grupo los clusters y las redes LAN, que por contra son denominados sistemas débilmente acoplados.
Un cluster, es un conjunto de computadores conectados a través de una red de comunicaciones. Los nodos de estos sistemas en red, pueden estar formados tanto por estaciones de trabajo, como por ordenadores personales, y pueden estar conectados mediante redes de propósito general como Ethernet, o con tecnologías especiales de alta velocidad como Fast Ethernet, Gigabit Ethernet, Myrinet, InfiniBand, o SCI.
Otros sistemas que pertenecen a esta categoría, son los sistemas de procesamiento paralelo masivo (MPP, Massively parallel processing). Estos supercomputadores, están constituídos por cientos o miles de procesadores. A diferencia de otros sistemas multicomputador, éstos utilizan redes de interconexión *ad hoc* de alto rendimiento, que generalmente están diseñadas por los mismos fabricantes.

4.2. Paralelismo en algoritmos evolutivos

Los Algoritmos Evolutivos Paralelos (AEPs), son una consecuencia a las necesidades de rendimiento exigidas por problemas de extrema complejidad, en los que el tiempo de ejecución utilizando AEs secuenciales es demasiado alto. Por este motivo, se estudiaron las vías posibles de paralelización de los componentes de un AE, dando lugar a tres grandes modelos [10]: algoritmos maestro-esclavo, algoritmos de grano fino y algoritmos de grano grueso. Es quizás en estos últimos, también llamados AEs distribuídos, donde se pueden encontrar un mayor número de estudios y conclusiones destacables.

Por ejemplo y como veremos más adelante, algunos trabajos demuestran que es posible enriquecer la diversidad de los individuos, mediante la creación de varias poblaciones en lugar de una sola. Así, una buena política de migración de los individuos de cada subpoblación, puede ayudar significativamente a que el algoritmo no converja hacia mínimos o máximos locales.

Antes de iniciar la paralelización de un algoritmo, es necesario establecer qué elementos del mismo son susceptibles de paralelizarse. En el caso de los AEs, es evidente que la función de bondad de cada individuo, es una tarea que realizada en paralelo no afecta a su comportamiento. Por contra, el operador de selección debe ser aplicado de forma global a toda una población, si se quiere mantener los resultados de la versión secuencial. Esta primera aproximación, recibe el nombre de algoritmos maestro-esclavo.

Otra vertiente de paralelización, consiste en la división de la población tradicional en subpoblaciones que están dotadas de cierta independencia, pero con algún mecanismo de comunicación entre ellas. Este sistema de paralelización, es quizás el de mayor acogida como alternativa al homólogo secuencial. Esto es debido a que, además de la mejora en el rendimiento computacional, las diferentes líneas evolutivas proporcionan una mejora objetiva en el proceso general de búsqueda. En la terminología de estos sistemas, se dice que estos algoritmos dan lugar a especies o nichos de individuos separados.

Esta segunda aproximación, se divide a su vez en algoritmos de grano grueso y algoritmos de grano fino. Las diferencias entre ambas propuestas radica en el tamaño de las poblaciones (mayores en las primeras), y en los mecanismos de interacción entre los individuos.

Por último, podemos encontrar propuestas híbridas que combinan varias de las lineas anteriores en dos niveles. A este tipo de diseño se le denomina algoritmo evolutivo jerárquico.

4.2.1. Algoritmos maestro-esclavo

Estos algoritmos, poseen una única población de individuos gestionada por el nodo maestro. Dicha población, ha de ser repartida entre los nodos esclavos, que se encargarán de la evaluación de los individuos, o incluso de las operaciones de cruce y mutación en algunos casos. Una vez que cada nodo esclavo termina su proceso, el resultado es devuelto al nodo maestro, que se encargará de la etapa de selección y posterior reparto.

Normalmente, la operación que se suele implementar en paralelo es la evaluación de los individuos, que además, suele ser la más compleja. Por otro lado, el hecho de ser una operación independiente de la población, la hace totalmente paralelizable.

El intercambio de información entre nodos es simple: el nodo maestro, envía el subconjunto de individuos que corresponde a cada esclavo, y éstos le devuelven los valores de bondad.

Por otro lado, esta comunicación puede ser síncrona o asíncrona. En el primer caso, el nodo maestro debe esperar la finalización del cálculo de bondad para todos los individuos, antes de empezar con la siguiente generación. En el segundo, el algoritmo no espera a los nodos más lentos, agilizándose de esta manera el proceso evolutivo. Es fácil deducir, que la comunicación síncrona tiene un comportamiento similar a la versión secuencial de un AE, no siendo exactamente así en la modalidad asíncrona.
En los algoritmos maestro-esclavo, no existe ninguna especificación concreta acerca de la arquitectura subyacente [10]. Esto hace, que puedan ser implementados tanto en sistemas de memoria compartida, como de memoria distribuída. En el primer caso, la población puede estar almacenada en la memoria común, de tal manera que cada esclavo, podría leer directamente de ella aquellos individuos que le fueran asignados. En el caso de memoria distribuída, sería el nodo maestro el encargado de asignar y distribuir los individuos correspondientes a cada esclavo.
En cuanto al comportamiento de este tipo de algoritmos, se han realizado diversos estudios con el fin de evaluar su rendimiento. Algunos resultados interesantes los podemos encontrar en el trabajo de Abramson, Mills y Perkins [2]. En él, se observó para los AEs maestro-esclavo, un incremento de las prestaciones razonablemente alto, en comparación con las versiones secuenciales, usándose para ello hasta 16 procesadores simultáneamente. Curiosamente, al aumentar más aún el número de procesadores, se pudo apreciar una degradación en el rendimiento. Este efecto fue justificado por los autores como una consecuencia debida al aumento del coste en las comunicaciones.

4.2.2. Algoritmos de grano fino

Este tipo de algoritmos, ha sido diseñado para ser implementados sobre arquitecturas masivamente paralelas. En esta ocasión, la población se encuentra distribuída entre los distintos procesadores, siendo la situación ideal, un individuo por procesador.
Las operaciones de cruce y selección, se harán entre individuos pertenecientes a un mismo vecindario de individuos adyacentes, y de acuerdo a la representación espacial presentada por la arquitectura antes citada. Para favorecer la interacción (aunque leve) entre todos los individuos de la población total, se permite el solapamiento de vecindarios.
Para determinar la influencia del tamaño de los vecindarios sobre la selección de individuos, se han realizado algunos estudios al respecto. Así, Sarma y DeJong [72] encontraron que la relación entre el radio de los vecindarios y el radio de la población completa, podía influir de forma importante en este aspecto. Además, cuantificaron el tiempo que tardaba una solución aceptable, en propagarse por toda la población, estableciendo para ello, distintos

tamaños para los vecindarios.
Otros estudios, se han centrado en la influencia de la topología espacial de la arquitectura. Schwehm [73] trabajó sobre la resolución del particionamiento de grafos, utilizando para ello representaciones en anillo, toroidal, cúbica $16 \times 8 \times 8$ e hipercúbica $4 \times 4 \times 4 \times 4 \times 4$. Aunque el autor establece la estructura toroidal como la más apta para una convergencia más rápida, no especifica detalles sobre la calidad de las soluciones encontradas.

4.2.3. Algoritmos de grano grueso

Entre las características más importantes de estos algoritmos, destacan el uso de múltiples subpoblaciones, y la migración de individuos entre ellas. Debido a esto, las subpoblaciones evolucionarán de manera independiente, por lo que el ratio de migración será un factor a tener en cuenta a la hora de obtener resultados satisfactorios.
En la literatura, el primer estudio de AEPs sobre múltiples poblaciones se le atribuye a Grosso en 1985 [36]. Paradójicamente, algunos de los resultados mostrados en este estudio, arrojan más sombras que luces sobre las ventajas del uso de este tipo de sistemas frente a los AEs tradicionales. Un ejemplo de esto, se observa en el hecho de que los individuos que evolucionan en poblaciones aisladas, convergen más rápidamente hacia valores óptimos, pero con unos resultados mejorables por los algoritmos tradicionales secuenciales. De hecho, una vez que se alcanza un determinado ratio de migración, que dependerá del problema y de la topología empleada, las soluciones encontradas por estos algoritmos son equiparables en calidad, a las encontradas por la versión secuencial.
Este estudio y otros posteriores, demuestran que el buen funcionamiento de estos algoritmos, depende en gran medida de varios factores, entre los que destacan, la frecuencia con la que se realicen las migraciones, y la topología de comunicación entre subpoblaciones.

4.2.3.1. Influencia de las migraciones

Es necesario tener en cuenta varios factores importantes a la hora de establecer la política de migraciones entre poblaciones. El primero de ellos, tiene que ver con la frecuencia con la que éstas se llevarán a cabo. La mayoría de las implementaciones existentes, programan las migraciones a intervalos fijos o, dicho de otra forma, de una forma síncrona. Otra tendencia, se basa en una política de migraciones asíncrona, de manera que éstas sólo se efectúan cuando se produce un evento. Un ejemplo, lo podemos ver en el estudio descrito en la tesis de Grosso [36]. En él, se propone la proximidad a la convergencia de la población, como referencia del mecanismo de migración. Braun [8] hizo algo parecido, aunque en este caso

las migraciones se realizaban una vez que la población había convergido completamente. De esta manera, se intentaba restaurar la diversidad perdida por el aislamiento de cada subpoblación.
La búsqueda del momento adecuado para la migración de los individuos, es un aspecto muy a tener en cuenta a la hora de diseñar un algoritmo de estas características. Así, una migración prematura, podría no aportar ninguna mejora al resto de poblaciones, debido sobre todo a la escasa calidad de los individuos enviados. Además, se estaría desperdiciando valiosos y costosos recursos de red. Sin embargo, una migración muy tardía, podría afectar también negativamente, y aumentar considerablemente el tiempo de convergencia al óptimo global, o incluso propiciar que ésta no se produjera.
Otro aspecto a tener muy en cuenta, es la elección de los individuos que se envían en las migraciones. Pettey, Leuze y Grefenstette [65] proponen enviar el mejor individuo de cada población. Los resultados experimentales que obtuvieron, indicaban que los resultados eran de una calidad similar a los obtenidos por AEs con una única población.
Por otro lado, Marin, Trelles-Salazar y Sandoval [54] propusieron que cada subpoblación, transcurridas las generaciones necesarias para realizar la migración, enviaran su mejor individuo a un nodo maestro. Este nodo, se encargaría de elegir a los mejores individuos entre todos los recibidos, y posteriormente, de reenviarlos de nuevo a todas las poblaciones. Con un número pequeño de nodos (alrededor de seis), obtuvieron resultados que propiciaban un *speedup*[1] casi lineal en los tiempos de ejecución.

4.2.3.2. Influencia de la topología de comunicación

Con frecuencia, éste es un factor que no es tomado muy en cuenta a la hora de desarrollar un AEP. A pesar de ello, la topología de comunicación, puede determinar la velocidad con la que las mejores soluciones, son propagadas hacia el resto de poblaciones. Tanto es así, que si el grado de conectividad es alto, y/o el diámetro de la red es pequeño, las soluciones consideradas como buenas, se expandirán rápidamente a través de todas las poblaciones, favoreciendose así la evolución hacia el óptimo. En caso contrario, las poblaciones estarán más aisladas unas de otras, por lo que su evolución será más lenta, y por tanto, habrá una demora en la incorporación de los mejores genes en las nuevas soluciones.
Por otro lado, es necesario tener en cuenta que la densidad de conexiones puede influir negativamente en el algoritmo, ya que supone una sobrecarga importante en el tráfico de

[1] el speedup es una medida de la mejora en el tiempo de ejecución de un algoritmo paralelo, en relación con la versión secuencial. Se dice que el speedup es lineal, cuando su valor coincide con el número de procesadores

la red. La elección de una buena topología, dará como resultado un mejor rendimiento del AEP.
Generalmente, es común el uso de topologías estáticas que no cambian en la medida que avanza el proceso de evolución. Las más utilizadas suelen ser las topologías de anillo y de hipercubo de cuatro dimensiones. Sin embargo, existen propuestas en las que las conexiones entre poblaciones no están definidas desde el principio, sino que se establecen sobre la marcha cuando se cumpla algún tipo de condición. Así, la diversidad de la población o la distancia entre genotipos son ejemplos de criterios que podemos encontrar en la literatura [59].

4.2.4. Algoritmos Híbridos

Este grupo de algoritmos, combinan dos de los modelos vistos hasta ahora en una estructura de dos niveles. En el nivel superior, suele haber siempre un algoritmo de grano grueso. En el nivel inferior se han probado los tres tipos de algoritmos vistos: maestro-esclavo, de grano fino y de grano grueso. Esta estructuración en dos niveles, es conocida como jerarquía, por lo que a estos algoritmos se les denomina comúnmente algoritmos evolutivos jerárquicos.
Atendiendo a las combinaciones posibles, podemos encontrar estructuras con un algoritmo multipoblación en el nivel superior y un algoritmo de grano fino en el nivel inferior. Existen varias implementaciones de este tipo de algoritmos. Gruau [37] usó un algoritmo multipoblación, cuyas poblaciones estaban conectadas en forma toroidal de dos dimensiones. Cada una de las poblaciones, fue organizada siguiendo una estructura de grano fino (una cuadrícula de dos dimensiones). Otros autores, como Lin, Goodman y Punch [51], conectaron las poblaciones del nivel superior usando una topología de anillo, y conservaron la distribución en cuadrícula de dos dimensiones para el nivel inferior. Compararon esta implementación con varios tipos de AEs, usando para ello un problema de planificación. Entre los algoritmos usados para la comparativa, se encuentran algunas implementaciones paralelas de las vistas anteriormente, y versiones secuenciales de un AE. Los resultados que obtuvieron con este algoritmo híbrido, fueron mejores que los obtenidos por el resto de propuestas.
Como segunda alternativa, podemos encontrar un algoritmo multipoblación en el nivel superior, y un algoritmo maestro-esclavo en cada una de las poblaciones. Bianchini y Brown [7] experimentaron con estos algoritmos, y consiguieron soluciones de la misma calidad que las obtenidas con algoritmos multipoblación y maestro-esclavo, pero con un menor coste temporal.
Como tercera opción de hibridación, se pueden considerar algoritmos multipoblación en los niveles superior e inferior. Se ha podido comprobar, que usando una topología altamente

conectada en el nivel inferior, y una frecuencia de migración elevada, se consigue una mejor distribución de las soluciones óptimas a través de las poblaciones. En el nivel superior, se suele optar por frecuencias de migraciones bajas, consiguiéndose así una complejidad similar a la de un algoritmo paralelo con multipoblaciones tradicional [34].

5

k-Label Dependent Evolutionary Distance Weighting

«La pregunta de si un computador puede pensar no es más interesante que la pregunta de si un submarino puede nadar.»

– Edsger Wybe Dijkstra

En este capítulo, describiremos nuestro método de pesado local sobre un k óptimo, al que hemos llamado *k-Label Dependent Evolutionary Distance Weighting* (kLDEDW). Al ser el objetivo final la obtención de un clasificador basado en la regla kNN, los pasos que comprenden nuestra propuesta comienzan con la búsqueda de una matriz de pesos y un k óptimo a partir de unos datos de entrenamiento. Una vez obtenido el modelo, el sistema estará preparado para la categorización de nuevas instancias pertenencientes al dominio considerado en la fase anterior.
Las siguientes secciones, explican todos estos aspectos con detalle. Así, la sección 5.1 presenta el propósito y motivación de esta propuesta, mientras que el uso de los parámetros obtenidos en el proceso de aprendizaje (pesos y número de vecinos) es expuesto en la sección 5.2. Finalmente, en la sección 5.3 se explicará el algoritmo de búsqueda en detalle.

5.1. Propósito

Tal y como se ha descrito previamente, el objetivo perseguido en nuestro estudio, es el de buscar un conjunto de pesos y un valor adecuado para el parámetro k, para así optimizar la regla de los vecinos más cercanos en procesos de clasificación. La base de nuestro método consiste en que el conjunto de pesos encontrado en el proceso de aprendizaje, no será el

mismo para todas las instancias a clasificar, sino que dependerá de la etiqueta o clase de la instancia vecina involucrada en el cálculo de la distancia. Así, los pesos a encontrar, conformarán una matriz con tantas filas como etiquetas, y tantas columnas como características. Si bien existen referencias al uso de métodos locales de pesado, podemos decir que éstas contrastan con el enfoque adoptado mayoritariamente en la literatura, en el que el foco de estudio se centra en un vector de atributos (uno por cada característica independientemente de la clase). La obtención de una matriz de pesos, permitirá transformar el espacio de características para una posterior clasificación. Además de esto, esta transformación estará conducida por un valor de k óptimo, que también es calculada de manera simultánea en el proceso de aprendizaje.

A pesar de que un conjunto de pesos por etiqueta proporciona más información que un simple vector, plantea el problema de seleccionar el conjunto de pesos (que se corresponde con una fila de la matriz) que debe ser aplicado en la fase de prueba. Esto es debido a que, por un lado, los ejemplos a clasificar no disponen de etiqueta, y por otro, existe la casuística de que las instancias participantes en el cálculo de la distancia, pudieran pertenecer a clases distintas.

Lo descrito anteriormente, plantea la necesidad de establecer una heurística que determine qué vector de pesos, debe ser utilizado en el proceso de clasificación de un ejemplo dado. En nuestro caso, la heurística considerada se basa en la regla de los vecinos más cercanos, pero con una función de similaridad modificada.

En la siguiente sección, se describirá el proceso de aprendizaje a partir de un conjunto de datos de entrenamiento, se formalizará la función de similaridad utilizada, y se detallará el proceso de clasificación de una instancia con etiqueta desconocida.

5.2. Proceso

Si bien es cierto que los clasificadores originales basados en la regla kNN, no necesitan fase de aprendizaje, en nuestro método se hace necesaria la búsqueda de un conjunto de pesos y de un número de vecinos óptimo. Este proceso tendrá lugar a partir de los datos de entrenamiento, y dará como resultado un modelo predictivo con capacidad de generalización. Además, a diferencia de otras técnicas, al contemplarse el número de vecinos y la influencia de las características para una clase dada, el modelo resultante posee una interesante capacidad de descripción del dominio estudiado.

Tal y como podemos observar en la Figura 5.1, el ciclo de vida del método estudiado comienza con la disponibilidad de un conjunto de datos de entrenamiento, de los que deseamos

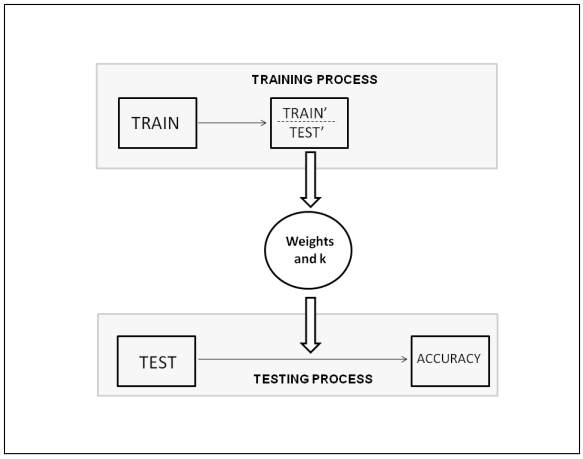


Figura 5.1: Procesos de entrenamiento y prueba

extraer el conocimiento necesario para mejorar la regla kNN, esto es, un conjunto óptimo de vecinos y una matriz de pesos que modificará una función de similaridad. Para ello, el propio conjunto de entrenamiento se particionará en sendos subconjuntos de entrenamiento y prueba, de tal manera que para cada instancia del subconjunto que hace de testing, se calcularán los vecinos más cercanos según los mejores pesos y mejor número de vecinos encontrados hasta ese momento en un proceso iterativo de optimización. Los ejemplos que conforman la vecindad, pertenecen al subconjunto de entrenamiento, y la tasa de acierto sobre el subconjunto de prueba, irá determinando la calidad del modelo. Una vez alcanzada la convergencia hacia un óptimo global, los pesos y el número de vecinos deducidos, son empleados con la totalidad del conjunto de datos para clasificar nuevos ejemplos de etiqueta desconocida. En términos de experimentación, y tal y como podemos observar en la Figura 5.1, nuestro método será evaluado y comparado con otros, utilizando para ello ejemplos de testing con etiqueta conocida, y midiendo la precisión obtenida en su clasificación. Para medir la precisión en la clasificación de un ejemplo de prueba, primero se oculta la etiqueta del mismo, y luego se compara con la etiqueta predicha por el modelo.

Para simplificar la notación, asumimos que el conjunto de clases o etiquetas puede ser representado por el conjunto de números enteros entre 1 y el número de etiquetas b. Así, sea $D = \{(e, l) \mid e \in \mathbb{R}^f \text{ y } l \in \{1, 2, ..., b\}\}$ el conjunto de datos objeto de estudio, siendo f el número de características, y b el número de etiquetas. Sea *label* una aplicación que asigna a cada elemento e, la clase a la que pertenece. Supongamos que D se divide en dos conjuntos TR y TS, siendo cada uno de ellos el conjunto de entrenamiento y el conjunto de prueba respectivamente, tal que $D = TR \cup TS$ y $TR \cap TS = \emptyset$. De esta manera, los ejemplos de TS (conjunto de prueba) serán usados para comprobar la bondad de kLDEDW, y por

tanto, no formarán parte del cálculo de los pesos y k (ver Figura 5.1 en la que los datos de prueba pertenecen a un conjunto diferente al de entrenamiento). Tal y como se detallará en la sección 5.3, la obtención de una matriz $W = (\omega_{ij})_{b \times f}$ se realizará sólo a partir de los ejemplos de TR. Esta matriz será usada posteriormente, para modificar el cálculo de la distancia entre una instancia $x \in TR$, y otra $y \in TS$ durante la fase de prueba. Para este propósito, se utiliza una distancia pesada que se define como sigue [62]:

$$d_w(x,y) = \sum_{j=1}^{f} \omega_{label(x)j}(x_j - y_j)^2 \quad \text{con } x \in TR, y \in TS$$
$$\omega_{label(x)j} \text{ es el elemento (peso) de la matriz de pesos correspondiente a la fila } label(x) \text{ y a la columna } j \tag{5.1}$$

La justificación del uso de esta distancia modificada, viene dada por la dependencia de los pesos con respecto a una clase, de tal manera que el cálculo de la distancia entre un ejemplo de prueba y, y un ejemplo de entrenamiento x, depende de la etiqueta de x. Tal y como podemos ver en la ecuación 5.1, la selección de la fila de la matriz está determinada por la etiqueta del ejemplo de entrenamiento. Así, una vez que está definida la distancia d_w, la clasificación de un ejemplo perteneciente a TS es llevada a cabo mediante la regla de los vecinos más cercanos, pero utilizando como medida de similaridad d_w.
Veamos un ejemplo. Supongamos que tenemos seis ejemplos de entrenamiento pertenecientes a tres clases diferentes y con dos características. Sean $x_1 = (1.2, 2.3, 1)$, $x_2 = (3.2, 1.3, 1)$, $x_3 = (4.3, 3.3, 2)$, $x_4 = (3.2, 4.7, 2)$, $x_5 = (3.2, 4.3, 3)$, $x_6 = (5.2, 1.3, 3)$ los ejemplos a considerar. Los primeros dos valores representan las características, y el tercer valor la clase (1, 2 o 3).
Si quisiéramos predecir la etiqueta de un punto $y = (3.2, 4.6)$ aplicando la regla de los k vecinos más cercanos, el procedimiento sería como sigue:

$$d(x_1, y) = (1.2 - 3.2)^2 + (2.3 - 4.6)^2 = 9.29$$
$$d(x_2, y) = (3.2 - 3.2)^2 + (1.3 - 4.6)^2 = 10.89$$
$$d(x_3, y) = (4.3 - 3.2)^2 + (3.3 - 4.6)^2 = 2.9$$
$$d(x_4, y) = (3.2 - 3.2)^2 + (4.7 - 4.6)^2 = 0.01$$
$$d(x_5, y) = (3.2 - 3.2)^2 + (4.3 - 4.6)^2 = 0.09$$
$$d(x_6, y) = (5.2 - 3.2)^2 + (1.3 - 4.6)^2 = 14.89$$

Siendo 0.01 la distancia menor, para $k = 1$ la etiqueta se corresponde con la del punto x_4 (clase 2). Si consideramos $k = 3$, los puntos más cercanos son x_3, x_4 y x_5, y la clase mayoritaria vuelve a ser 2.

Por otro lado, para aplicar nuestro método, se debería calcular la distancia a y de los seis ejemplos de entrenamiento pero considerando un vector de pesos para cada clase y la ecuación 5.1. Si consideramos la matriz W $((1.4, 2.1), (3.1, 5.2), (0.7, 0.4))$, en la que $(1.4, 2.1)$ sería el conjunto de pesos para la clase 1, $(3.1, 5.2)$ el de la clase 2, y el vector restante el de la clase 3, el cálculo de la etiqueta de y vendría dado por:

$$d_w(x_1, y) = 1.4(1.2 - 3.2)^2 + 2.1(2.3 - 4.6)^2 = 16.709$$
$$d_w(x_2, y) = 1.4(3.2 - 3.2)^2 + 2.1(1.3 - 4.6)^2 = 22.869$$
$$d_w(x_3, y) = 3.1(4.3 - 3.2)^2 + 5.2(3.3 - 4.6)^2 = 12.539$$
$$d_w(x_4, y) = 3.1(3.2 - 3.2)^2 + 5.2(4.7 - 4.6)^2 = 0.052$$
$$d_w(x_5, y) = 0.7(3.2 - 3.2)^2 + 0.4(4.3 - 4.6)^2 = 0.036$$
$$d_w(x_6, y) = 0.7(5.2 - 3.2)^2 + 0.4(1.3 - 4.6)^2 = 7.156$$

Como el valor mínimo es 0.036, para $k = 1$ podemos determinar que la clase de y debe ser la misma que la de x_5 (clase 3). Si consideramos tres vecinos ($k = 3$), los puntos más cercanos son x_4, x_5 y x_6, y por tanto, la clase mayoritaria es también 3. Nótese que para $k = 5$, las clases 2 y 3 se contabilizarían dos veces, por lo que habría que establecer una política de resolución de empates. En nuestro caso optaríamos por la clase del vecino más cercano, que como vimos anteriormente es 3. Esta regla de clasificación está formalizada en la función $NearestN$ que será detallada en la sección 5.3.3.

Del ejemplo anterior se puede deducir que la elección de los pesos y el número de vecinos, es determinante para la tarea de predicción. Es por eso que en la fase de entrenamiento, se requiere de un proceso de búsqueda sobre los datos objeto de estudio. En la siguiente sección, se describirán los detalles del algoritmo evolutivo elegido para tal fin.

5.3. Cálculo de pesos y k

En esta sección, detallaremos el algoritmo evolutivo de búsqueda que permite obtener una matriz de pesos, y un valor óptimo de k. En la descripción de un algoritmo evolutivo, es necesario definir una codificación para los individuos, un conjunto de operadores genéticos, una función de bondad y una regla de reemplazo generacional. Todos estos aspectos se

describirán en las siguientes subsecciones.
En cuanto a la paralelización del algoritmo, y según lo expuesto en el capítulo 4, hemos seguido un diseño maestro-esclavo basado en la distribución del cálculo de la bondad de cada individuo. Aunque es un dato relevante a tener en cuenta, y que justifica la inclusión del capítulo sobre paralelismo, es necesario apuntar que este detalle sólo afecta a nivel de implementación y que por tanto, pasa totalmente desapercibido en las subsecciones de carácter formal que aparecen a continuación.

5.3.1. Codificación de individuos

Un individuo, es un conjunto de pesos para cada etiqueta, y un valor para el parámetro k. Los pesos se representan a través de una matriz que tiene una fila por cada clase, y una columna por cada característica. Cada elemento de la matriz, es un valor real con límites inferior y superior parametrizables. En nuestro caso, el rango inicial elegido es $[0, 1]$, por lo que un valor de 0 indicaría que la característica correspondiente es irrelevante. Sin embargo, es posible alcanzar valores por encima de uno a través de los operadores de cruze y mutación, de tal manera que a mayor magnitud, mayor importancia para la característica en cuestión. Para el número de vecinos, hemos elegido valores impares comprendidos en el rango $[1, 5]$. Por tanto, la población inicial estará compuesta por N matrices de dimensión $b \times f$, con valores reales aleatorios comprendidos entre 0 y 1, y N valores enteros aleatorios e impares comprendidos entre 1 y 5.

5.3.2. Operadores genéticos

Debido a que nuestro algoritmo maneja matrices en vez de estructuras lineales, hemos adaptado los operadores de cruce y mutación usados en los individuos. Así, el operador de cruce entre dos matrices, es llevado a cabo fila a fila, de tal manera que la fila i-ésima de uno de los progenitores, es cruzada con la fila i-ésima del otro. Para el cruce de dos vectores de números reales, hemos usado el cruce $BLX - \alpha$, un operador ampliamente utilizado en la literatura [23]. Para el parámetro k, la operación consiste en la elección aleatoria de uno de los dos valores correspondientes de los ancestros cruzados. El cruce $BLX - \alpha$ es descrito como sigue. Si ω_{ij} y ω'_{ij} son los elementos j-ésimos de la fila i-ésima de cada progenitor, el nuevo gen es un valor real elegido aleatoriamente en el rango $[\omega_{min} - I\alpha, \omega_{max} + I\alpha]$ (Figura 5.2).

Con respecto al operador de mutación y su efecto en las matrices de pesos, cada fila tiene una probabilidad p de que uno de sus pesos w, sea incrementado o decrementado en una

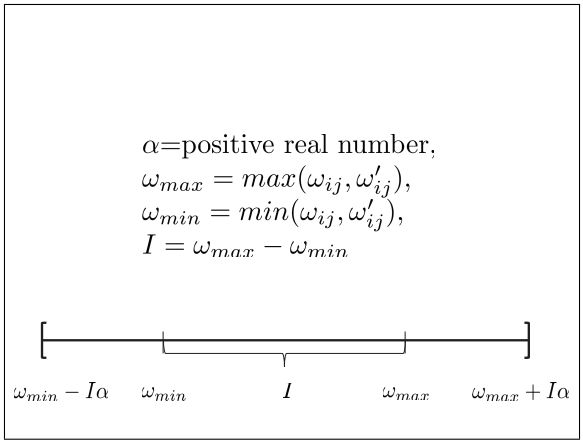


Figura 5.2: Cruce BLX-α

cantidad $\delta * w$. Inicialmente $\delta \in [0, 1]$, pero para un mejor ajuste, la cota superior es reducida en g/G cada g generaciones, donde G es el número total de generaciones. Por ejemplo, si $G = 100$ y $g = 10$, durante las primeras diez generaciones, la cota superior para δ será 1; en las siguientes diez, ésta será 0.9; en las siguientes diez, será 0.8; y así sucesivamente. Para el parámetro k, su valor es incrementado o decrementado en dos unidades dentro del rango $[1, 5]$ y con una probabilidad de p.

5.3.3. Función de bondad

Hay que apuntar, que durante la fase de entrenamiento, sólo se usa el subconjunto $TR \subset D$. Además, la función de bondad debe premiar a aquellos individuos que obtienen una mejor tasa de clasificación, pero intentando prevenir el sobreajuste sobre los datos de entrenamiento. Por este motivo, tanto la fase de entrenamiento como la función de bondad, están basados en validación cruzada y precisión de clasificación respectivamente. Debido a que las etiquetas del conjunto de entrenamiento son conocidas, es posible contabilizar la tasa de acierto en las predicciones, y tomarla como una medida de calidad para un clasificador.

En la Figura 5.3 se muestra el cálculo de la bondad usando $m \times s$ validaciones cruzadas, donde m es el número de veces que se repite el proceso de validación (línea 3) y s, es el número de particiones del conjunto de entrenamiento TR (línea 4). Así, para cada validación, el conjunto TR es dividido en las s bolsas B_1, B_2... B_s. Después, para cada bolsa B_j, se evalúa el rendimiento de una clasificación, donde B_j actúa como conjunto de prueba, y el resto de TR como conjunto de entrenamiento. Esta evaluación es conducida por la función *Evaluate*, que será descrita posteriormente. El error de clasificación sobre cada B_j es acumulado en

```
1: Fitness(W, k, TR) : error
2: error = 0
3: for i = 1 to m do
4:    Divide TR into s bags: B_1...B_s
5:    partialError = 0
6:    for j = 1 to s do
7:       partialError = partialError + Evaluate(W, k, TR − B_j, B_j)
8:    end for
9:    partialError = partialError/s
10:   error = error + partialError
11: end for
12: error = error/m
13: return error

14: {nótese que en la función Evaluate, Train y Test son subdivisiones de los datos de entrenamiento
    (el conjunto real de prueba es desconocido en este punto, y por tanto no se usa en el cálculo)}
    Evaluate(W, k, Train, Test) : error
15: error = 0
16: for each instTest in Test do
17:    lab = NearestN(W, k, Train, instTest)
18:    if lab ≠ label(insTest) then
19:       error = error + 1
20:    end if
21: end for
22: error = error/size(Test)
23: return error

24: NearestN(W, k, Train, y) : labY
25: sortedInst and kneighbours are empty Sorted Sets
26: for each x in Train do
27:    insert x into sortedInst ordered by d_w(x, y)
28: end for
29: kneighbours = sortedInst.get(k)
30: {en el caso de que existan empates, el resultado será: labY = label(sortedInst.getFirst())}
    labY = majorityLabel(kneighbours)
31: return labY
```

Figura 5.3: Función de bondad

promedio por la variable *partialError* (líneas 7 y 9), y por *error* en cada validación (línea 10). Finalmente, el valor de bondad es el resultado de calcular el promedio de todas las validaciones (línea 12).

La función *Evaluate* toma como entrada la matriz W, el valor de k, el subconjunto que actúa como de entrenamiento $TR - B_j$ y el subconjunto que actúa como conjunto de prueba B_j (línea 7). Por tanto, el resultado de esta función es la precisión de un clasificador basado en la matriz W sobre B_j, y tomando como referencia $TR - B_j$ para el cálculo de los vecinos. El clasificador usado para este propósito, es una versión de los k vecinos más cercanos que toma como función de similaridad, la distancia euclídea modificada descrita en Eq. 5.1. Para cada instancia del conjunto que actúa como de prueba (línea 16), la etiqueta devuelta por la función $NearestN$ es la mayoritaria de acuerdo a los k ejemplos más cercanos pertenecientes al conjunto que actúa como de entrenamiento (línea 17). Si la etiqueta devuelta no se corresponde con la etiqueta real de la instancia de prueba, el error se incrementa en 1 (línea 19). Seguidamente, el error resultante es normalizado con el tamaño del conjunto que actúa como de prueba (línea 22). Por tanto, el valor devuelto por *Evaluate* es un número real comprendido entre 0 (todos los ejemplos clasificados) y 1 (ningún ejemplo clasificado).

La función $NearestN$ es la encargada de calcular las instancias más cercanas pertenecientes al conjunto de referencia, con respecto a cada instancia y del conjunto sometido a evaluación (línea 24 y siguientes). Cada ejemplo del conjunto de referencia, es insertado en un conjunto ordenado de acuerdo a su distancia a y, de tal manera que el primer ejemplo del conjunto ordenado resultante, se correspondería con el vecino más cercano a y, y el último al más lejano (línea 27). Una vez seleccionados los k vecinos más cercanos del conjunto ordenado (línea 29), se devuelve la etiqueta mayoritaria (lineas 30 y 31).

5.3.4. Reemplazo generacional

En cuanto al reemplazo generacional de los individuos de una generación a los de la siguiente, hemos elegido un diseño elitista, donde el mejor de los individuos pasa de una generación a otra sin ser afectado por el operador de mutación. El resto de la nueva población está formado de la siguiente manera: Si el número de individuos es N, $C - 1$ son generados por clonación del mejor de los individuos de la generación anterior, y los siguientes $N - C$ individuos son creados mediante el operador de cruce. Todos los individuos excepto el primero, están sujetos al operador de mutación con una probabilidad p. Para seleccionar los individuos que participarán en la operación de cruce, se ha probado tanto el método de la ruleta como el de torneo sin que se hayan apreciado diferencias significativas.

6

Resultados

«Duda siempre de ti mismo, hasta que los datos no dejen lugar a dudas.»

– Louis Pasteur

Este capítulo presenta la configuración del algoritmo, y los resultados obtenidos cuando lo comparamos con otras propuestas basadas en los vecinos más cercanos. Para la primera consideración, y tal y como se muestra en la sección 6.1, se ha realizado un análisis de sensibilidad con el fin de observar el comportamiento de nuestra propuesta ante diferentes situaciones, modificando para ello el valor de los parámetros más importantes.
Con respecto al diseño experimental, éste se ha dividido en dos fases. En la primera, la cual es detallada en la sección 6.2, se muestran los resultados obtenidos con varios datasets del repositorio UCI[1] y su validación estadística cuando kLDEDW es comparado con otras propuestas. El principal objetivo de este paso, es el de demostrar con el preciso rigor experimental, si nuestro método es mejor que otros frente a una variedad lo suficientemente significativa de situaciones diferentes, esto es, frente a una colección lo bastante amplia y heterogénea de datasets.
En una siguiente fase, se pretende demostrar la funcionalidad del algoritmo frente a casos reales de alta dimensionalidad. Para ello, en la sección 6.3, se han utilizado datos de teledetección formados por la adquisición de mediciones LIDAR junto con ortofotografías, mientras que en la sección 6.4, mostramos los resultados obtenidos con la información extraída de un sistema CRM perteneciente a la compañía de telecomunicaciones Orange, y que formó parte de la competición KDDCUP del año 2009.
Todos estos conceptos son detallados a continuación.

6.1. Parámetros

La configuración estándar de un algoritmo evolutivo es ampliamente discutida en la literatura. Pero para aquellos parámetros que son exclusivos, es necesario realizar un análisis de su comportamiento.

Para observar la sensibilidad de los parámetros de nuestro trabajo, hemos realizado 8 experimentos para 4 datasets diferentes. 4 de los experimentos, fueron llevados a cabo variando el parámetro α del operador de cruce, y los 4 restantes, variando el parámetro g del operador de mutación. Como se ha podido comprobar en la subsección 5.3.2, el parámetro α hace referencia al cruce $BLX-\alpha$ para individuos con codificación real, y g, es un parámetro *ad hoc*. Además, para cada experimento, se ha usado 2×2 validaciones cruzadas un total de 3 veces.

Podemos observar que aunque la influencia de los parámetros no es significativa, se aprecia un mejor comportamiento del algoritmo para un valor de $g = 20$ (ver Cuadro 6.1). Por

Cuadro 6.1: Análisis de sensibilidad para α y g

	balance s.	*haberman*	*liver d.*	*tae*
$\alpha = 0.3$	86.704	73.148	60.203	55.665
$\alpha = 0.4$	86.704	**73.910**	58.901	56.651
$\alpha = 0.5$	86.491	73.529	**60.829**	**56.765**
$\alpha = 0.6$	**86.944**	72.386	60.059	56.217
$g = 5$	86.678	72.277	60.926	55.995
$g = 10$	86.997	72.985	61.409	56.659
$g = 15$	87.317	72.494	**61.552**	54.124
$g = 20$	**87.343**	**73.802**	61.312	**58.855**
media	**86.897**	**73.066**	**60.649**	**56.366**
desv. típ.	**±0.310**	**±0.643**	**±0.890**	**±1.32**

otro lado, aunque el parámetro α parece ser algo menos influyente, podemos apreciar en la literatura que éste suele tener un valor estándar de 0.5.

Teniendo en cuenta todo lo anterior, podemos tomar como valores por defecto para los parámetros estudiados $\alpha = 0.5$ y $g = 20$. El resto de parámetros de nuestro algoritmo evolutivo, son aquellos que aparecen ampliamente en la literatura. Esto es, un tamaño de

población formado por 100 individuos, un elitismo del 10 % (valor de C, ver subsección 5.3.4), y una probabilidad de mutación del 10 %. Respecto al número de generaciones, las pruebas muestran una estabilidad en la función objetivo cuando se alcanzan entre 90 y 100 iteraciones, por lo que se ha tomado 100 como número de generaciones.
Por último, y con el fin de realizar una comparación objetiva con otros algoritmos en la fase de experimentación, se ha considerado la misma configuración de nuestro algoritmo en el resto de secciones de este capítulo.

6.2. Datos UCI

Cuadro 6.2: Precisión de los métodos relacionados

	kLDEDW	**Paredes [62]**	**Mohemmed [57]**	**Fernández [25]**	**AlSukker [3]**
Validación	**5x10CV**	**100x5CV**	**10x3CV**	**100x10CV**	**10x(66 %-33 %)**
australian	85.943	82.63	85	–	–
balance s.	88.186	82.02	90.7	–	–
breast w.	96.529	96.31	96.2	–	99.93
diabetes	74.533	72.01	75.8	74.118	81.04
glass	73.593	71.48	–	63.714	–
heart s.	79.47	77.66	81.2	–	–
ionosphere	92.207	–	–	–	98.85
liver d.	61.004	59.78	–	65.294	–
sonar	86.466	–	–	–	90.19
vehicle	72.43	70.62	57.7	–	–
vote	95.651	93.39	–	–	–
vowel	98.815	98.64	–	–	–
wine	97.913	98.56	93.8	–	–

Podemos observar en el Cuadro 6.2, los resultados obtenidos por los diferentes trabajos referenciados en la sección 2.4 con datasets del repositorio UCI [1]. Como se ha visto previamente, los trabajos presentados tienen en común el uso de pesos locales. Debido a la disparidad en los métodos de validación utilizados en cada uno de ellos, y también a los datos usados, esta tabla es solo informativa, y no debe utilizarse como estudio comparativo en ningún caso. Dicho lo anterior, podemos observar que la experimentación más completa es llevada a cabo por Paredes et al. [62].
Para llevar a cabo los experimentos, y un estudio comparativo adecuado, hemos implementado DE4 [3], y compilado el código fuente de CW [62] que está disponible en:
`http://users.dsic.upv.es/~rparedes/research/CPW/index.html`.

También, hemos comparado kLDEDW con IBk (implementación de los k vecinos más cercanos en el framework WEKA [40]) utilizando tres valores típicos de k: 1, 3 y 5. Por otro lado, hemos utilizado en las pruebas treinta datasets del repositorio UCI de muy diversa naturaleza, con diferente número de etiquetas y tipos de atributos. Como se verá después, este número de pruebas permitirá llevar a cabo un análisis estadístico fiable.
Con respecto al método de prueba, hemos elegido 10 validaciones cruzadas y para reducir la influencia de la naturaleza aleatoria de los algoritmos evolutivos, hemos realizado 5 experimentos para cada dataset, cada uno de ellos con una ordenación diferente de los datos. Por tanto, los experimentos implementados por dataset son 50 en total (10 validaciones cruzadas realizadas 5 veces).
Los datos utilizados durante la experimentación, fueron preprocesados con el framework WEKA. Así, los atributos nominales fueron convertidos a numéricos, todos los valores fueron normalizados, y finalmente, los valores desconocidos fueron sustituídos por la media de los valores conocidos del correspondiente atributo. Además, los algoritmos comparados en los experimentos, fueron probados con los mismos datasets preprocesados (ver Cuadro 6.3).

Cuadro 6.3: Precisión de cada algoritmo estudiado

	kLDEDW	IB1	IB3	IB5	CW	DE4
australian	**85.94 ± 2.04**	80.20 ± 2.25	83.55 ± 1.92	84.37 ± 1.24	80.92 ± 4.27	81.88 ± 2.20
balance s.	88.18 ± 1.00	86.80 ± 0.97	86.90 ± 1.10	**88.28 ± 0.95**	85.63 ± 3.22	58.40 ± 6.59
breast t.	**71.01 ± 5.50**	68.29 ± 5.01	63.90 ± 5.98	65.39 ± 7.65	70.74 ± 10.07	67.54 ± 3.71
breast w.	96.52 ± 1.16	95.65 ± 1.05	96.63 ± 0.98	**97.18 ± 1.08**	96.75 ± 2.15	96.34 ± 1.31
car	**96.38 ± 0.59**	93.12 ± 0.59	93.12 ± 0.59	93.12 ± 0.59	73.18 ± 29.87	96.32 ± 0.67
cmc	46.89 ± 2.32	44.35 ± 1.37	**47.05 ± 1.68**	45.93 ± 1.58	44.01 ± 4.04	44.85 ± 1.65
diabetes	74.53 ± 2.04	70.93 ± 2.10	74.38 ± 2.21	**74.72 ± 1.54**	68.74 ± 3.99	67.07 ± 2.51
e. coli	86.13 ± 2.11	80.23 ± 2.84	84.83 ± 2.19	**86.48 ± 1.96**	80.53 ± 5.93	79.91 ± 2.89
glass	73.59 ± 3.41	70.01 ± 3.33	68.62 ± 3.41	66.12 ± 4.64	**75.23 ± 7.4**	64.94 ± 6.39
haberman	**74.40 ± 2.60**	67.03 ± 2.59	71.58 ± 2.71	71.07 ± 1.77	71.51 ± 4.55	56.01 ± 3.96
heart s.	**79.47 ± 2.78**	75.20 ± 3.11	78.52 ± 3.39	78.36 ± 3.13	76.22 ± 6.94	74.37 ± 3.26
hill v.	**53.32 ± 2.26**	50.31 ± 1.52	51.16 ± 2.40	51.39 ± 2.58	52.86 ± 5.34	52.27 ± 2.65
ionosphere	**92.20 ± 2.18**	86.89 ± 2.46	86.13 ± 1.78	85.61 ± 1.52	91.68 ± 4.63	91.97 ± 2.11
liver d.	61.00 ± 3.78	59.31 ± 3.96	61.80 ± 3.85	58.35 ± 3.70	**63.33 ± 7.96**	62.54 ± 3.43
lymphoma	**85.12 ± 5.69**	81.73 ± 3.03	78.70 ± 4.14	78.53 ± 4.30	84.85 ± 8.93	79.86 ± 4.11
mammographic	81.04 ± 1.69	76.83 ± 1.89	80.97 ± 1.84	**82.20 ± 1.52**	75.96 ± 4.07	76.07 ± 1.66
mfeat m.	**72.04 ± 0.91**	65.83 ± 1.49	69.65 ± 1.30	71.08 ± 1.15	65.99 ± 2.57	66.63 ± 1.02
ozone	94.00 ± 0.26	92.27 ± 0.95	93.96 ± 0.30	**94.04 ± 0.31**	93.80 ± 0.61	78.06 ± 1.24
pendigits	99.49 ± 0.04	99.36 ± 0.12	99.37 ± 0.08	99.27 ± 0.08	**99.57 ± 0.17**	99.43 ± 0.04
postoperative p.	69.47 ± 4.28	62.89 ± 3.08	69.24 ± 4.31	**72.40 ± 3.87**	63.77 ± 13.61	43.80 ± 9.79
sonar	86.46 ± 3.78	84.85 ± 3.44	83.00 ± 4.93	82.56 ± 3.68	**88.97 ± 6.60**	85.02 ± 2.69
sponge	90.86 ± 3.88	92.33 ± 3.08	88.78 ± 1.74	88.78 ± 1.74	**95.00 ± 6.18**	87.14 ± 7.11
tae	61.33 ± 6.63	60.96 ± 6.14	50.35 ± 7.71	53.65 ± 5.86	**66.88 ± 12.17**	63.68 ± 7.56
transfusion	**76.34 ± 1.25**	69.46 ± 2.08	73.81 ± 1.50	75.99 ± 1.65	68.05 ± 15.39	64.24 ± 2.26
vehicle	72.43 ± 1.72	70.00 ± 1.44	70.59 ± 1.58	70.91 ± 1.53	**72.47 ± 3.92**	71.02 ± 1.73
vote	**95.65 ± 1.67**	93.00 ± 1.56	93.95 ± 1.84	94.01 ± 2.31	95.31 ± 3.07	94.39 ± 1.32
vowel	98.81 ± 0.62	99.07 ± 0.34	96.41 ± 1.46	92.75 ± 1.35	99.23 ± 0.87	**99.30 ± 0.30**
wine	97.91 ± 1.92	94.51 ± 1.84	95.64 ± 2.29	95.48 ± 2.35	**97.99 ± 3.51**	97.48 ± 2.08
yeast	57.24 ± 1.15	52.91 ± 1.43	55.21 ± 1.13	**57.56 ± 1.38**	53.47 ± 2.83	50.98 ± 1.55
zoo	91.08 ± 3.41	95.37 ± 2.82	92.73 ± 2.79	94.66 ± 2.17	**96.07 ± 5.93**	94.24 ± 4.08
	80.29 ± 2.42	**77.32 ± 2.26**	**78.02 ± 2.44**	**78.34 ± 2.31**	**78.29 ± 6.36**	**74.86 ± 3.06**

El Cuadro 6.3 muestra los resultados en porcentaje de los 30 datasets usados en la experimentación. Cada fila contiene la precisión media de 50 ejecuciones para cada clasificador (10 validaciones cruzadas realizadas 5 veces). Cabe destacar, que para la familia IBk un valor alto del parámetro k no implica necesariamente una mayor precisión. Por tanto, la elección de un valor adecuado de k, es importante en el comportamiento de los vecinos más cercanos.
A pie del Cuadro 6.3, se muestra la media de todas las pruebas. Si comparamos nuestro trabajo con el resto, se puede observar una mejora en los resultados obtenidos. Por otro lado, podemos ver que los promedios obtenidos por nuestro algoritmo son en general, mejores que los del resto, y que la desviación típica es próxima a la obtenida por la familia IBk y menor que la de los dos trabajos basados en pesado local. Así, nuestro método es más preciso en 11 de los 30 datasets observados, y es segundo en 13 de los 19 restantes. En la siguiente subsección, mostraremos que esta mejora es significativa desde un punto de vista estadístico.

6.2.1. Análisis estadístico

Con el fin de establecer una evaluación adecuada de los resultados, hemos llevado a cabo una serie de tests, que estadísticamente validan la comparativa realizada previamente entre todos los algoritmos que forman parte de la experimentación. En nuestro caso, hemos utilizado el test no paramétrico de Friedman (equivalente al test ANOVA con medidas repetidas) y el procedimiento post-hoc de Holm. La razón de utilizar tests no paramétricos, radica en la flexibilidad de su uso si los comparamos con los homólogos paramétricos, que con frecuencia incumplen algunas de las condiciones indispensables para su utilización, esto es, independencia, normalidad y homocedasticidad[16, 76, 30]. Concretamente, Demšar [16] sugiere que la condición de esfericidad[1], que es una propiedad similar a la homocedasticidad pero más apropiado para ANOVA con medidas repetidas, no puede ser garantizada debido a la naturaleza de los algoritmos de aprendizaje y de los datasets.
Para mostrar tal afirmación, podemos ver en el Cuadro 6.4 la varianza de las diferencias para cada par de algoritmos estudiados. Dichos valores están ordenados de menor a mayor. Si nos fijamos en kLDEDW e IB1 por ejemplo, podemos ver en el Cuadro anterior de los resultados (6.3), que para el dataset «australian» la diferencia en el rendimiento de ambos clasificadores es de 5.74 (85.94 menos 80.2). Asimismo, si nos fijamos en «balance s.» podemos apreciar una diferencia de 1.38 en el rendimiento, y para el resto de datasets

[1]la condición de esfericidad ocurre cuando la varianza de las diferencias de rendimiento entre dos clasificadores, es similar para todos los pares posibles de clasificadores

Cuadro 6.4: Varianza de diferencias (supuesto de esfericidad)

Algoritmos	Varianza
IB3-IB5	2.65
kLDEDW-IB1	6.77
kLDEDW-IB3	7.06
kLDEDW-IB5	8.53
IB1-IB3	12.14
IB1-IB5	15.14
IB1-CW	19.89
kLDEDW-CW	27.84
IB3-CW	38.46
IB5-CW	42.75
IB1-DE4	51.42
kLDEDW-DE4	61.58
CW-DE4	71.59
IB3-DE4	74.94
IB5-DE4	83.67

podríamos proceder de la misma manera. El valor 6.77 en el Cuadro 6.4 se corresponde con la varianza de todas esas diferencias entre kLDEDW e IB1. Como se puede apreciar, valores tan distintos de varianza evidencian una más que probable violación de la condición de esfericidad.

Por otro lado, el uso de un método post-hoc viene determinado por el hecho de que el test de Friedman, no es capaz por sí solo de identificar las diferencias entre el mejor algoritmo encontrado y sus competidores, sino que sólo detecta si existe o no disparidad en el conjunto de resultados. El método de Holm sí es capaz de detectar diferencias entre un algoritmo y otros restantes.

En cuanto al funcionamiento del test no paramétrico de Friedman, éste se compone de los siguientes pasos. Primero, para un dataset dado, se calcula el ránking de los resultados obtenidos por cada algoritmo, donde el mejor es dotado con un valor de 1, y al peor se le asigna k, siendo k el número de algoritmos comparados. Esta operación, es llevada a cabo

para los N datasets considerados, por lo que el ránking final, estará formado por la media obtenida en cada algoritmo. La hipótesis nula consiste en la asunción de que los resultados obtenidos por cada algoritmo son equivalentes (tienen la misma distribución de probabilidad o son distribuciones con la misma mediana), y por tanto, no son estadísticamente diferentes en cuanto a rendimiento. La Ecuación 6.1 representa el estadístico usado en el test de Friedman, donde $R_j = \frac{1}{N}\sum_i r_i^j$ y r_i^j es el ránking para el algoritmo i-ésimo y el j-ésimo dataset. Además de esto, χ_F^2 sigue una distribución χ^2 con $k-1$ grados de libertad. El Cuadro 6.5 muestra los ránkings obtenidos para cada algoritmo.

$$\chi_F^2 = \frac{12N}{k(k+1)}(\sum_j {R_j}^2 - \frac{k(k+1)^2}{4}) \tag{6.1}$$

Cuadro 6.5: Ránking promedio de los algoritmos estudiados (Friedman)

Algoritmo	Ránking
kLDEDW	2.033
CW	2.999
IB5	3.45
IB3	3.783
DE4	4.233
IB1	4.500

Siendo el valor p obtenido de $1.918E-6$ y $\alpha = 0.05$, podemos rechazar la hipótesis nula. El método post hoc de Holm, es un procedimiento a posteriori que permite comparar un algoritmo de control (en este caso kLDEDW) con el resto. Si tenemos k algoritmos, la hipótesis nula consiste en la presunción de que algunos de los restantes $k-1$ algoritmos, tiene la misma calidad que el de control. El procedimiento comienza comprobando las hipótesis secuencialmente, de acuerdo a la magnitud de p y en orden creciente.

$$z = (R_i - R_j)/\sqrt{\frac{k(k+1)}{6N}} \tag{6.2}$$

Sean $p_1, p_2, ..., p_{k-1}$ los valores p ordenados, tal que $p_1 \leq p_2 \leq ... \leq p_{k-1}$. El método de Holm compara cada p_i con $\alpha/(k-i)$, comenzando por el menor valor de p. Si $p_1 < \alpha/(k-1)$, entonces la hipótesis correspondiente puede ser rechazada, permitiendo comparar

p_2 con $\alpha/(k-2)$. Si la segunda hipótesis también es rechazada, el proceso continúa. Si hubiera alguna hipótesis que no pudiera ser rechazada, el resto de hipótesis (todavía sin evaluar) se considerarían aceptadas. En la Ecuación 6.2 podemos ver el estadístico utilizado para comparar el algoritmo i-ésimo con el j-ésimo. En nuestro caso, todas las hipótesis son rechazadas para $\alpha = 0.05$ (ver Cuadro 6.6).

Cuadro 6.6: Método de Holm

i	Algoritmo	p	$\alpha/(k-i)$
1	IB1	$3.282E-7$	0.01
2	DE4	$5.253E-6$	0.0125
3	IB3	$2.914E-4$	0.0166
4	IB5	0.003	0.025
5	CW	0.0454	0.05

Siendo las diferencias entre los seis algoritmos evaluados estadísticamente significativas, podemos concluir que kLDEDW obtiene los resultados más satisfactorios.

6.3. Datos LIDAR

La teledetección, es una disciplina empleada en tareas tales como gestión de recursos naturales, monitorización del entorno, respuesta a los desastres naturales etc.
Desde hace algún tiempo, el aprendizaje automático se está utilizando en teledetección con el fin de mejorar el rendimiento y su aplicabilidad. Por otro lado, para mejorar las técnicas tradicionales de teledetección basadas normalmente en imágenes, se está generalizando el uso de sensores activos como LIDAR (Light Detection and Ranging en inglés)[21]. Este hecho implica un aumento en la complejidad de los datos, lo que justifica aún más la utilización de técnicas de minería de datos y aprendizaje automático.
LIDAR es una tecnología de sensores activos capaces de medir propiedades de la luz (normalmente láser), con el fin de registrar información sobre objetivos distantes. En tareas de teledetección, estos sensores van acoplados a un avión para registrar la información de retorno proporcionada por distintos pulsos lanzados desde el propio sistema. Después de cada vuelo LIDAR, se obtiene una nube de puntos que es almacenada en una base de datos. Cada punto contiene diversa información, como por ejemplo su posición espacial (coordenadas z,

x e y), intensidad de retorno, número de retornos en una secuencia (si un pulso produce varios impactos en el suelo) etc.

Uno de los productos más importantes que se derivan de las técnicas mencionadas anteriormente, son los mapas LULC (Land Use/Land Cover), los cuales almacenan la información necesaria, para categorizar diferentes tipos de terreno en función de sus características morfológicas o funcionales. Esta información, puede ser muy útil para la gestión de entornos naturales (por ejemplo, para mapear y estimar el riesgo de incendio).

Nuestro caso de estudio, consiste en un mapa LULC de la zona geográfica del Alto Tajo (ver Figura 6.1). Los datos a clasificar, están formados por la fusión de variables derivadas del sistema LIDAR, junto con la información de una fotografía multiespectral (ver Cuadro 6.7) y suman un total de 2010 instancias etiquetadas con distintos tipos de suelo.

En cuanto a los terrenos, éstos se pueden clasificar como «carretera», «solar», «arbustal», «alameda», «robledal», «pradera», «pinar» y «encinar joven».

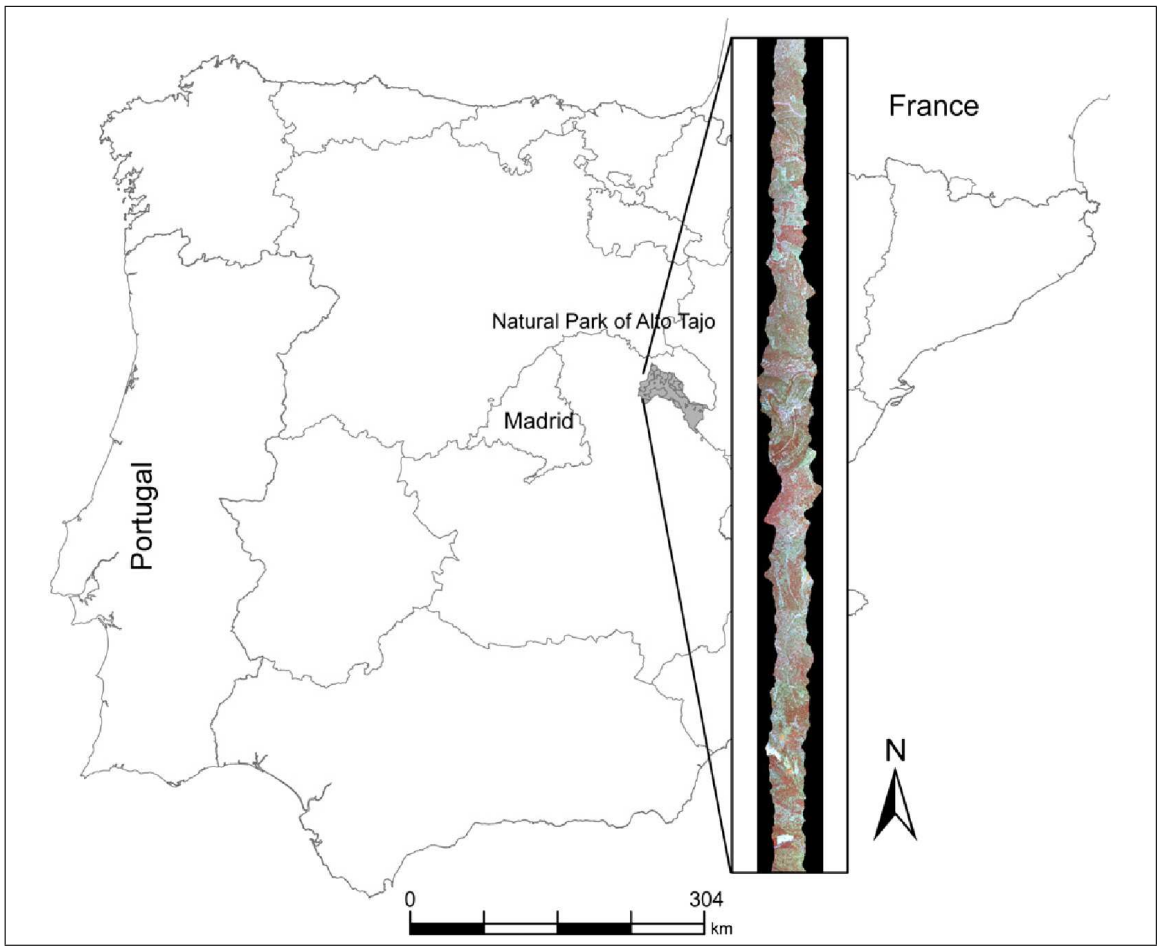


Figura 6.1: Zona de estudio

Cuadro 6.7: Atributos y bandas extraídos de los datos LIDAR y de la imagen multiespectral para Alto Tajo

Variable	Descripción	Variable	Descripción
B1	420-450 nm	B2	450-520 nm
B3	520-600 nm	B4	600-620 nm
B5	630-690 nm	B6	690-750 nm
B7	750-900 nm	B8	910-1050 nm
B9	1550-1750 nm	B10	2080-2350 nm
NDVI	$\frac{(B7-B5)}{(B7+B5)}$	SAVI	$\frac{(B7-B5)(1.5)}{(B7+B5+1.5)}$
NDII1	$\frac{(B7-B9)}{(B7+B9)}$	NDII2	$\frac{(B7-B10)}{(B7+B10)}$
HMAX	Height maximum	HMEAN	Height mean
HMED	Height median	HKURT	Height kurtosis
HRANGE	Height range	HSKEW	Height skewness
HSTD	Height standard deviation	HCV	Height coefficient of variation

En esta ocasión, hemos probado nuestra propuesta ejecutándola 50 veces, esto es, 5×10 validaciones cruzadas, y comparándola con las implementaciones en WEKA de kNN (IBk) para los valores de k 1, 3 y 5. Como se puede apreciar en el Cuadro 6.8, la regla kNN es mejorada significativamente cuando se combinan adecuadamente un valor óptimo de k y una matriz de pesos por etiqueta. De hecho, con respecto al mejor resultado final (IB5 con una precisión de 83.568 %) logramos una mejoría del 2.5 %.

Por otro lado, podemos observar en la última columna los resultados de IB5, cuando éste es combinado con el selector de atributos CFS [39] y utilizando como algoritmo de búsqueda una estrategia Best-First [15]. La razón de este experimento es que la selección de características puede ser vista como un caso extremo de pesado, en el que sólo existen dos valores de ponderación: 1 y 0. En cada validación, los atributos seleccionados a partir del conjunto de entrenamiento, son aplicados tanto al conjunto de entrenamiento como al de prueba antes de procederse con la tarea de clasificación. Por tanto, y al igual que sucede con nuestra propuesta, los atributos seleccionados (los pesos y k en nuestro caso) pueden ser distintos en cada validación. Aunque el preprocesamiento proporciona cierta mejoría en los resultados de IB5, éstos siguen siendo menor en general cuando los comparamos con kLDEDW, donde se alcanza una mejoría del 1.6 % con respecto al anterior.

Cuadro 6.8: Precisión para los datos de Alto Tajo

#CV	kLDEDW	IB1	IB3	IB5	IB5 + CFS
1	84.577	82.587	81.592	81.095	84.577
2	88.557	82.587	83.582	83.333	85.572
3	85.904	82.587	82.421	82.255	83.416
4	86.07	82.587	82.09	82.587	84.204
5	85.871	81.758	82.189	82.587	84.279
6	86.235	81.716	82.836	83.25	84.826
7	85.856	82.189	82.161	82.658	84.506
8	85.883	82.836	82.463	82.587	84.08
9	86.014	82.658	82.753	82.753	84.08
10	86.02	82.587	82.786	82.886	83.93
11	83.582	82.366	80.597	82.587	84.577
12	85.323	82.239	79.602	81.592	84.08
13	86.235	83.582	82.919	84.245	85.406
14	87.065	80.597	83.831	85.448	85.697
15	87.065	81.758	83.98	85.97	85.97
16	86.733	82.711	83.25	85.075	85.406
17	86.638	83.085	82.871	84.151	85.004
18	86.754	82.67	83.396	84.577	85.261
19	85.959	82.374	82.863	83.858	84.743
20	85.92	82.836	82.736	83.881	85.075
21	88.06	82.034	86.567	85.075	83.085
22	87.313	82.189	83.831	83.582	83.831
23	86.899	85.075	82.919	84.08	84.577
24	86.07	83.333	82.96	84.577	85.199
25	86.965	82.421	83.582	85.373	85.473
26	86.318	82.711	82.753	84.494	84.826
27	85.999	83.284	82.161	83.724	84.364
28	85.883	82.504	82.338	83.769	84.08
29	86.014	82.161	82.366	83.858	84.356
30	86.169	81.903	82.786	84.03	84.726
31	85.075	81.537	82.587	84.08	85.075
32	87.811	81.891	83.333	83.582	85.075
33	86.733	83.085	82.421	82.587	84.245
34	86.194	83.582	82.836	83.085	84.328
35	85.771	82.919	82.388	83.184	83.781
36	85.655	82.587	82.007	82.836	83.997
37	85.643	82.289	82.303	82.8	84.506
38	85.634	82.09	82.587	82.898	84.391
39	85.572	81.947	82.587	82.919	84.19
40	85.821	82.152	82.736	83.383	84.428
41	84.577	81.592	81.592	82.09	81.592
42	86.318	81.94	83.831	84.328	82.587
43	85.738	80.1	83.748	84.411	84.08
44	85.199	80.597	82.214	82.96	83.333
45	84.776	80.763	82.289	83.085	83.184
46	85.489	80.97	82.753	83.914	84.245
47	85.999	80.1	83.085	84.08	84.861
48	85.883	80.846	83.147	84.142	84.639
49	85.627	81.237	82.974	83.914	84.301
50	85.92	81.592	83.234	84.179	84.378
	86.068 ± 0.864	**82.195 ± 0.925**	**82.757 ± 0.952**	**83.568 ± 0.992**	**84.408 ± 0.796**

6.4. Datos masivos (KDDCUP 2009)

Los sistemas CRM (Customer Relationship Management en inglés) son un elemento clave para las estrategias de márketing actuales. En el año 2009, la conocida competición KDDCUP, ofreció la oportunidad de acceder a una gran base de datos perteneciente a la compañía francesa de telecomunicaciones Orange (`http://http://www.kddcup-orange.com/`). Los objetivos del reto eran tres: predecir la propensión de los clientes a cambiar de proveedor (churn); a adquirir nuevos productos o servicios (appetency); y a contratar mejoras o complementos, para de esa manera, hacer la venta más rentable (up-selling). Las puntuaciones (scores) son la forma más práctica de generar conocimiento sobre los clientes en un sistema CRM. Una puntuación (la salida de un modelo) es el resultado de evaluar un conjunto de instancias perteneciente a una variable objetivo a explicar (es decir, en este caso, *churn*, *appetency* o *up-selling*). Los sistemas que producen puntuaciones, permiten proyectar en una población determinada información cuantificable. Las puntuaciones se calculan utilizando las variables de entrada que describen casos, y son después utilizadas por el sistema de información, para personalizar la relación con el cliente. Los laboratorios Orange, han desarrollado una plataforma capaz de construir modelos de predicción cuya principal característica es su escalabilidad para manejar cientos de miles de casos, y miles de variables. La detección rápida y robusta de las variables que más contribuyen a los resultados de predicción, son un factor clave en la implementación de estrategias de márketing.
Para cada uno de los objetivos, se ha elaborado un dataset de clase binaria con 50000 instancias y 230 atributos, de los cuales 190 son numéricos y 40 categóricos.
Puesto que en este estudio se está utilizando el paradigma del vecino más cercano y dado el carácter masivo de los datos a analizar, se han utilizado 10 validaciones cruzadas como método de experimentación. Si bien pudiera no ser suficiente para establecer una comparativa con el rigor necesario, es aceptable para observar el comportamiento de nuestra propuesta cuando la aplicamos a datos de estas características.
Además de lo anterior, cada conjunto de entrenamiento es sometido previamente a un muestreo con el algoritmo descrito por Vitter en 1985 [82]. Tras este paso, el conjunto de entrenamiento queda reducido a 5000 ejemplos de los 45000 iniciales.
Por otro lado, los datos bajo estudio sufren un gran desbalance con respecto a la clase, por lo que también se ha optado por realizar un muestreo con reemplazo de los mismos, para de esta manera, generar una distribución uniforme. Tras este paso, el conjunto de entrenamiento se queda finalmente con 2500 ejemplos distribuídos uniformemente entre las dos etiquetas de la clase. Tenemos por tanto en cada validación cruzada, 2500 ejemplos de

entrenamiento y 5000 ejemplos de prueba. En los Cuadros 6.9, 6.10 y 6.11 podemos ver

Cuadro 6.9: Precisión para el análisis de *appetency*

#CV	kLDEDW	IB1	IB3	IB5	IB1+CFS
1	97.82	91.56	80.66	71.9	92.52
2	98.2	90.12	78.08	70.3	91.3
3	97.82	90.72	78.72	69.68	92.58
4	97.98	90.64	78.78	69.52	91.48
5	98	90.1	79.12	70.98	91.26
6	98	90.68	79.76	70.78	92.18
7	97.78	90.94	80.8	72.64	91.98
8	97.48	90.82	80	71.38	93.24
9	98.1	92.58	83.1	74.84	92.3
10	98	92.12	82.22	73.28	93.56
	97.918	**91.028**	**80.124**	**71.53**	**92.24**

los resultados de 10 validaciones cruzadas cuando comparamos nuestra propuesta con IBk para 1, 3 y 5 vecinos, y con las mismas condiciones de preprocesado descritas anteriormente. Además, el rival con mejor rendimiento (en los tres casos IB1), es sometido a una reducción de características mediante CFS y Best-First, tal y como se hizo en el apartado 6.3.
En los tres casos (*appetency*, *churn* y *up-selling*) podemos ver que la elección de un adecuado número de vecinos, en consonancia con una matriz de pesos correctamente graduada por evolución, consigue unos resultados muy superiores a los conseguidos por la familia kNN, aún aplicando ponderación binaria (selección de atributos) a la variante de mejor comportamiento. Cabe destacar que incluso en el estudio del objetivo *churn*, el proceso de selección de atributos no hace sino empobrecer los resultados obtenidos.

Cuadro 6.10: Precisión para el análisis de *churn*

#CV	kLDEDW	IB1	IB3	IB5	IB1+CFS
1	92.22	74.14	55.54	50	26.5
2	91.04	72.66	55.58	50.92	25.88
3	91.56	71.72	55.12	51.4	25.96
4	92.14	70.5	52.98	49.28	26.32
5	92.1	69.9	51.5	47.74	60.28
6	91.4	70.22	52.1	49.36	22.68
7	92.12	70.94	52.14	47.84	27.74
8	91.44	72.7	54.92	50.72	60.18
9	91.7	71.26	55.26	49.98	61.36
10	91.58	71.52	52.92	47.46	66.6
	91.73	**71.556**	**53.806**	**49.47**	**40.35**

Cuadro 6.11: Precisión para el análisis de *up-selling*

#CV	kLDEDW	IB1	IB3	IB5	IB1+CFS
1	92.44	74.22	57.72	53.46	85.08
2	92.46	72.08	55.26	51.86	73.58
3	92.36	71.88	54.34	50.68	86.92
4	92.08	71.64	53.5	50.94	84
5	91.8	72.24	55.92	51.34	79.62
6	91.96	73.02	53.74	51.04	80.64
7	91.46	72.78	54.2	51	82.54
8	92	72.96	54.8	52	76.8
9	91.64	72.72	54.94	51.82	74.26
10	91.84	73.24	55.22	50.58	82.24
	92.004	**72.678**	**54.964**	**51.472**	**80.568**

7

Conclusiones

«Lo consiguieron porque no sabían que era imposible.»

– Jean Cocteau

En esta memoria, hemos explorado dos mejoras para la regla de los vecinos más cercanos. La primera de ellas consiste en la selección de un adecuado valor de k. La segunda, que se complementa con la primera, es un sistema de pesado de atributos.

Las principales características de nuestra propuesta son dos. Primero, los pesos no son iguales para todos los datos en observación sino que dependen de la etiqueta, y por tanto, el objetivo a optimizar es una matriz en vez del clásico vector que hallamos en la literatura. La segunda caraterística es que el número de vecinos es deducido en la fase de entrenamiento junto con la matriz de ponderación. La combinación de ambas características deriva en una mejora objetiva de la regla kNN.

La principal dificultad de este enfoque se manifiesta a la hora de decidir cuál o cuáles son los vectores de pesos a utilizar en el cálculo de los vecinos. El uso de una distancia pesada basada en las etiquetas del conjunto de entrenamiento, resuelve este problema.

En cuanto a la optimización del modelo, se ha elegido un algoritmo evolutivo con codificación real cuyos resultados han sido testeados con datasets del repositorio UCI. El número de conjuntos de datos utilizado y el uso de validación cruzada en la experimentación, ha permitido un riguroso análisis estadísticos de los resultados. Este análisis confirma que nuestra propuesta mejora los resultados de cinco clasificadores, incluyendo dos trabajos de pesado local.

En cuanto a la aplicabilidad del método descrito, se han realizado pruebas con datos reales en un problema de clasificación de suelos, y con datos masivos provenientes de la competición KDDCUP 2009. Los resultados obtenidos muestran una mejor capacidad de predicción,

cuando se compara con la regla kNN.

7.1. Trabajo futuro

Las líneas futuras de actuación, se pueden dividir en varios apartados. Por un lado, se pretende seguir aplicando nuestra propuesta en problemas reales de diversa naturaleza, con especial interés en problemas de teledetección y de origen biológico. Además, debido al diseño elegido, es posible la experimentación con otros tipos de algoritmos de optimización tales como CHC [22], evolución diferencial [77] o sistemas inmunes artificiales [24] por citar algunos.

Por último, cabe resaltar la posibilidad de utilizar nuestro método como algoritmo de *Selección de Características*. Como idea inicial, los pesos para cada etiqueta, permitirían a priori medir la importancia de cada atributo con respecto a dicha etiqueta. Así, para n atributos, el atributo con mayor peso, tendría un valor de 1 en el ránking, y el de menor peso tendría un valor de n. Para cada atributo, su importancia «global» podría ser vista como la suma de todos los ránkings (el de cada etiqueta), de tal manera que el atributo de menor suma podría considerarse el más importante, y el de mayor suma el de menos relevancia. En el apéndice A se puede ver una descripción del algoritmo propuesto, y su aplicación a los datos de teledetección mostrados en la sección 6.3.

Apéndice A

Ránking de atributos a partir de una matriz de pesos

```
1: {W es una matriz de pesos con n filas y m columnas. R es un vector de m elementos}
   Ranking(W) : R
2: R_1 = 0, R_2 = 0, ..., R_m = 0
3: for i = 1 to n do
4:    for j = 1 to m do
5:       R_j = R_j + Pos(W_ij, W_i) {Pos(W_ij, W_i) devuelve la posición según su peso del
   elemento W_ij en la fila (vector) W_i}
6:    end for
7: end for
8: return R

9: Pos(x, V) : i
10: ReverseSort(V) {ordenación del vector de mayor a menor peso}
11: for i = 1 to m do
12:    if V_i == x then
13:       break
14:    end if
15: end for
16: return i
```

Figura A.1: Algoritmo de ránking

Si bien es cierto que durante el desarrollo de esta memoria, se han tratado de resolver unas hipótesis de partida, no lo es menos el hecho de que hayan ido aparecido otras nuevas. Un ejemplo de esto lo encontramos en la posibilidad de utilizar nuestra propuesta como una alternativa factible a las técnicas de selección de características. La hipótesis de partida,

plantea la posibilidad de que la matriz de pesos también pudiera reflejar la importancia de cada atributo con respecto al dataset.

Una primera aproximación, consistiría en considerar la posición de cada atributo en función de su peso. Así, para una fila dada con m atributos, el atributo con mayor ponderación, tendría un valor de 1 en el ránking de pesos de la etiqueta correspondiente, y el de menor ponderación tendría un valor de m. Para cada atributo, su importancia «global» podría ser vista como la suma de todos los ránkings (el de cada etiqueta), de tal manera que el atributo de menor suma podría considerarse el de más importancia, y el de mayor suma el de menos.

En la Figura A.1 se muestra un algoritmo que describe el proceso anterior. Podemos ver en la línea 1, que el algoritmo toma como entrada una matriz de pesos y devuelve un vector de posiciones acumuladas (ránking). Al estar asignándose 1 al mejor atributo, y m al peor (siendo m el número de atributos) cuanto menor sea el valor en el ránking, más importante será el atributo. Para cada fila i, se calcula la posición de cada atributo j según su importancia (peso) dentro de la fila, y luego se acumula en suma (líneas 3,4 y 5). Para calcular la posición en el ránking de un valor x en un vector V (línea 9), primero se ordena el vector de mayor a menor (línea 10), y luego se busca la posición que ocupa x en dicho vector (líneas 11 y sucesivas). Cuanto mayor sea el valor x, menor será su posición en el vector.

En el Cuadro A.1 podemos observar la ejecución del algoritmo IB3 utilizando 10×10 CV sobre los datos de Alto Tajo utilizados en la sección 6.3. En cada columna, aparecen los resultados del clasificador combinado con nuestra propuesta, y con otros 2 conocidos algoritmos de selección. Cada fila, representa la precisión media del clasificador para distintos números de atributos. Así, la primera fila representa la precisión de clasificación cuando se elige el mejor atributo considerado por cada algoritmo; la segunda, representa la misma tarea pero con los dos mejores atributos; en la tercera, se consideran los tres mejores; y así sucesivamente. A diferencia de los algoritmos basados en ránking, el CFS sólo tiene en cuenta los atributos que considera mejores (los que no tienen correlación con ningún otro atributo). Es por eso que sólo considera 10 atributos en este caso.

A pesar de que el algoritmo descrito es sólo un prototipo, y de que se hace necesario hacer más pruebas, se puede apreciar en los resultados que la hipótesis planteada podría ser factible.

Lo primero que podemos ver, es que con todos los atributos la precisión media es de 82.816 %. Además, con 5 atributos, nuestra propuesta parece obtener una precisión notoriamente superior a la del resto (80.916 % frente a 70.177 % y 67.892 %). Y con 6, además se supera los resultados obtenidos con el conjunto completo de atributos (82.932 % frente a 82.816 %).

Cuadro A.1: Selección de características para datos de Alto Tajo

#atributos	Relief	kLDEDW	CFS
1	42.446	42.807	42.807
2	49.666	66.094	61.022
3	59.926	78.615	63.485
4	66.555	80.399	62.641
5	**67.892**	**80.916**	**70.177**
6	**69.524**	**82.932**	**73.631**
7	69.284	84.861	81.944
8	68.663	84.675	82.305
9	72.696	84.651	85.182
10	74.104	84.892	85.633
11	74.461	84.727	–
12	74.008	84.241	–
13	74.049	85.116	–
14	82.282	84.639	–
15	84.339	84.704	–
16	84.435	83.911	–
17	84.074	83.663	–
18	83.929	83.824	–
19	83.904	83.471	–
20	83.385	83.225	–
21	82.816	**82.816**	–

Aún con todo esto, y tal y como se ha mencionado más arriba, será necesario establecer un riguroso plan de experimentación para poder obtener resultados concluyentes, y realizar sucesivas mejoras si así fuera necesario.

Bibliografía

[1] D.J. Newman A. Asuncion. UCI machine learning repository, 2007.

[2] D.A. Abramson, G. Mills, and S. Perkins. *Parallelisation of a Genetic Algorithm for the Computation of Efficient Train Schedules*. Research report. School of Computing and Information Technology, Faculty of Science and Technology, Griffith University, 1993.

[3] A. AlSukker, R. Khushaba, and A. Al-Ani. Optimizing the k-nn metric weights using differential evolution. In *Multimedia Computing and Information Technology (MCIT), 2010 International Conference on*, pages 89 –92, 2010.

[4] Gordon Bell. Ultracomputers: a teraflop before its time. *Commun. ACM*, 35(8):26–47, August 1992.

[5] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[6] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *ICML'06: Proceedings of the 23rd international conference on Machine learning*, pages 97–104, New York, NY, USA, 2006. ACM Press.

[7] Ricardo Bianchini and Christopher M. Brown. Parallel genetic algorithms on distributed-memory architectures. Technical report, Rochester, NY, USA, 1993.

[8] Heinrich Braun. On solving travelling salesman problems by genetic algorithms. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 129–133, London, UK, UK, 1991. Springer-Verlag.

[9] Yizheng Cai, Nando de Freitas, and James Little. Robust Visual Tracking for Multiple Targets. pages 107–118. 2006.

[10] E. Cantu-Paz. A survey of parallel genetic algorithms. In *Calculateurs Paralleles, Reseaux et Systems Repartis*, volume 10, pages 141–171, 1998.

[11] Guo G. Wang K. Chen, L. Class-dependent projection based method for text categorization. *Pattern Recognition Letters*, 32(10):1493–1501, 2011.

[12] Liu H. Chai J. Bao Z. Chen, B. Large margin feature weighting method via linear programming. *IEEE Transactions on Knowledge and Data Engineering*, 21(10):1475–1488, 2009.

[13] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[14] T. Cover and P. Hart. Nearest neighbor pattern classification. *Information Theory, IEEE Transactions on*, 13(1):21–27, January 1967.

[15] Rina Dechter and Judea Pearl. Generalized best-first search strategies and the optimality of a*. *JOURNAL OF THE ACM*, 32(3):505–536, 1985.

[16] Janez Demšar. Statistical comparisons of classifiers over multiple data sets. *J. Mach. Learn. Res.*, 7:1–30, 2006.

[17] S. Dudani. The distance-weighted k-nearest-neighbour rule. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-6(4):325–327, 1975.

[18] A. E. Eiben, T. Bäck, Informatik Centrum Dortmund, Niels Bohrweg, and Niels Bohrweg. An empirical investigation of multi-parent recombination operators in evolution strategies, 1997.

[19] A. E. Eiben, B. Jansen, Zbigniew Michalewicz, and Ben Paechter. Solving csps using self-adaptive constraint weights: how to prevent eas from cheating. In *GECCO'00*, pages 128–134, 2000.

[20] J. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, June 1990.

[21] Todd L. Erdody and L. Monika Moskal. Fusion of LiDAR and imagery for estimating forest canopy fuels. *Remote Sensing of Environment*, 114(4):725–737, April 2010.

[22] Eshelman. The CHC Adaptive Search Algorithm : How to Have Safe Search When Engaging in Nontraditional Genetic Recombination. *Foundations of Genetic Algorithms*, pages 265–283, 1991.

[23] L. J. Eshelman and J. D. Schaffer. Real-coded genetic algorithms and interval-schemata. In Darrell L. Whitley, editor, *Foundation of Genetic Algorithms 2*, pages 187–202, San Mateo, CA, 1993. Morgan Kaufmann.

[24] J D Farmer, N H Packard, and A S Perelson. The immune system, adaptation, and machine learning. *Phys. D*, 2(1-3):187–204, October 1986.

[25] F. Fernandez and P. Isasi. Local feature weighting in nearest prototype classification. *Neural Networks, IEEE Transactions on*, 19(1):40 –53, 2008.

[26] E. Fix and J.L. Hodges. *Discriminatory Analysis: Nonparametric Discrimination, Small Sample Performance.* Report. Air University, USAF School of Aviation Medecine, 1952.

[27] Evelyn Fix and Jr. Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties. Technical Report Project 21-49-004, Report Number 4, USAF School of Aviation Medicine, Randolf Field, Texas, 1951.

[28] Myron Flickner, Harpreet Sawhney, Wayne Niblack, Jonathan Ashley, Qian Huang, Byron Dom, Monika Gorkani, Jim Hafner, Denis Lee, Dragutin Petkovic, David Steele, and Peter Yanker. Query by image and video content: The qbic system. *Computer*, 28:23–32, 1995.

[29] Michael J. Flynn. Some computer organizations and their effectiveness. *IEEE Trans. Comput.*, 21(9):948–960, September 1972.

[30] Salvador García and Francisco Herrera. An Extension on "Statistical Comparisons of Classifiers over Multiple Data Sets"for all Pairwise Comparisons. *Journal of Machine Learning Research*, 9:2677–2694, 2008.

[31] Jorge García-Gutiérrez, Daniel Mateos-García, and José C. Riquelme-Santos. Evorstack: A label-dependent evolutive stacking on remote sensing data fusion. *Neurocomput.*, 75(1):115–122, January 2012.

[32] Geoffrey W. Gates. The Reduced Nearest Neighbor Rule. *IEEE Transactions on Information Theory*, 18(3):431–433, 1972.

[33] A. Gersho and R.M. Gray. *Vector Quantization and Signal Compression*. Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1992.

[34] D.E. Goldberg. Personal communication. Technical report, 1996.

[35] Stephen Grossberg. Competitive learning: From interactive activation to adaptive resonance. *Cognitive Science*, 11(1):23 – 63, 1987.

[36] P.B. Grosso. *Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multilocus Model*. University of Michigan, 1985.

[37] Frédéric Gruau, L'universite Claude Bernard lyon I, Of A Diplome De Doctorat, M. Jacques Demongeot, Examinators M. Michel Cosnard, M. Jacques Mazoyer, M. Pierre Peretto, and M. Darell Whitley. Neural network synthesis using cellular encoding and the genetic algorithm., 1994.

[38] Steve R. Gunn. Support vector machines for classification and regression, 1998.

[39] M. A. Hall. *Correlation-based Feature Subset Selection for Machine Learning*. PhD thesis, University of Waikato, Hamilton, New Zealand, 1998.

[40] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[41] P. E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:515–516, 1968.

[42] M.H. Hassoun. *Associative neural memories: theory and implementation.* Oxford University Press, 1993.

[43] D.O. Hebb. *The Organization of Behavior: A Neuropsychological Theory.* Taylor & Francis (reprinted edition), 2002.

[44] Geoffrey E. Hinton and Terrence J. Sejnowski. Analyzing cooperative computation. In *Proc. 5th annual cong. of the Cognitive Science Society*, Rochester, NY, May 1983. Cognitive Science Society.

[45] John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, April 1982.

[46] Riquelme J., Ferrer F., and Aguilar-Ruiz J. Búsqueda de un patrón para el valor de k en k-nn. In *Proceedings of IX Conferencia de la Asociación Española para la Inteligencia Artificial*, 2001.

[47] Kevin Knight. Connectionist ideas and algorithms. *Commun. ACM*, 33(11):58–74, November 1990.

[48] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982. 10.1007/BF00337288.

[49] William B. Langdon and Wolfgang Banzhaf. A SIMD interpreter for genetic programming on GPU graphics cards. In *Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971 of *Lecture Notes in Computer Science*, pages 73–85, Naples, 26-28 March 2008. Springer.

[50] Gondra I. Liu L. Li, C.H. An efficient parallel neural network-based multi-instance learning algorithm. *Journal of Supercomputing*, pages 1–17, 2012.

[51] S. C. Lin, E. D. Goodman, and William F. Punch. Investigating Parallel Genetic Algorithms on Job Shop Scheduling Problems. In *EP '97: Proceedings of the 6th International Conference on Evolutionary Programming VI*, pages 383–393, London, UK, 1997. Springer-Verlag.

[52] Wei Liu and Sanjay Chawla. Class confidence weighted knn algorithms for imbalanced data sets. In *Proceedings of the 15th Pacific-Asia conference on Advances in knowledge discovery and data mining - Volume Part II*, PAKDD'11, pages 345–356, Berlin, Heidelberg, 2011. Springer-Verlag.

[53] Dario Lucarella. A document retrieval system based on nearest neighbour searching. *J. Inf. Sci.*, 14(1):25–33, January 1988.

[54] F.J. Marin, O. Trelles Salazar, and Francisco Sandoval Hernández. *Genetic Algorithms on LAN-message passing architectures using PVM : Application to the Routing Problem*, pages pp. 534–543. 1994. en Parallel problem solving from nature- PPSN III (Y. Davidor, H.-P. Schwefel, R. Manner, Eds.), Springer, LNCS 866.

[55] Daniel Mateos-García, Jorge García-Gutiérrez, and José C. Riquelme-Santos. On the evolutionary optimization of k-nn by label-dependent feature weighting. *Pattern Recognition Letters*, 33(16):2232 – 2238, 2012.

[56] Warren McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5:115–133, 1943. 10.1007/BF02478259.

[57] Ammar W. Mohemmed and Mengjie Zhang. Evaluation of particle swarm optimization based centroid classifier with different distance metrics. In *IEEE Congress on Evolutionary Computation'08*, pages 2929–2932, 2008.

[58] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Comput.*, 1(2):281–294, 1989.

[59] Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato. An efficient migration scheme for subpopulation-based asynchronously parallel genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 649–, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[60] Satoshi Niijima and Satoru Kuhara. Effective nearest neighbor methods for multiclass cancer classification using microarray data. 2008.

[61] Stephen M. Omohundro. Five balltree construction algorithms. Technical report, 1989.

[62] R. Paredes and E. Vidal. Learning weighted metrics to minimize nearest-neighbor classification error. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(7):1100–1110, 2006.

[63] D.-C. Park. Centroid neural network with weighted features. *Journal of Circuits, Systems and Computers*, 18(8):1353–1367, 2009.

[64] Hamid Parvin, Hosein Alizadeh, and Behrouz Minaei-bidgoli. Mknn: Modified k-nearest neighbor. In *Proceedings of the World Congress on Engineering and Computer Science*, 2008.

[65] Chrisila C. Pettey, Michael R. Leuze, and John J. Grefenstette. A parallel genetic algorithm. In *ICGA*, pages 155–161, 1987.

[66] Ali Mustafa Qamar, Eric Gaussier, Jean-Pierre Chevallet, and Joo Hwee Lim. Similarity learning for nearest neighbor classification. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining*, ICDM '08, pages 983–988, Washington, DC, USA, 2008. IEEE Computer Society.

[67] Michael J. Quinn. *Parallel Computing: Theory and Practice*. McGraw–Hill, Inc., New York, NY, 1994.

[68] M.L. Raymer, T.E. Doom, L.A. Kuhn, and W.F. Punch. Knowledge discovery in medical and biological datasets using a hybrid bayes classifier/evolutionary algorithm. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 33(5):802–813, 2003.

[69] M.L. Raymer, W.F. Punch, E.D. Goodman, L.A. Kuhn, and A.K. Jain. Dimensionality reduction using genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 4(2):164–171, July 2000.

[70] F. Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

[71] D.E. Rumelhart, J.L. McClelland, and P.D.P.R. Group. *Parallel Distributed Processing - Vol. 1: Foundations*. Computational Models of Cognition and Perception. MIT Press, 1987.

[72] Jayshree Sarma and Kenneth De Jong. An analysis of the effects of neighborhood size and shape on local selection algorithms. pages 236–244. Springer-Verlag, 1996.

[73] M. Schwehm. Implementation of genetic algorithms on various interconnection networks. In *In Proceedings of Parallel Computing and Transputers Application Conference*, pages 195–203. IOS Press (Amsterdam), 1992.

[74] Wang-X. Yu D. Shen, C. Feature weighting of support vector machines based on derivative saliency analysis and its application to financial data mining. *International Journal of Advancements in Computing Technology*, 4(1):199–206, 2012.

[75] Roger N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–1323, 1987.

[76] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 4 edition, 2007.

[77] Rainer Storn and Kenneth Price. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, December 1997.

[78] Y. Sun. Iterative relief for feature weighting: Algorithms, theories, and applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1035–1051, 2007.

[79] H.H. Szu and R.A. Messner. Adaptive invariant novelty filters. 74:518–519, 1986.

[80] Muhammad Atif Tahir, Ahmed Bouridane, and Fatih Kurugollu. Simultaneous feature selection and feature weighting using hybrid tabu search/k-nearest neighbor classifier. *Pattern Recognition Letters*, 28(4):438–446, 2007.

[81] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2006.

[82] Jeffrey S. Vitter. Random sampling with a reservoir. *ACM Trans. Math. Softw.*, 11(1):37–57, March 1985.

[83] Aha-D.W. Mohri T. Wettschereck, D. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms. *Artificial Intelligence Review*, 11(1-5):273–314, 1997.

[84] B. Widrow and M.E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, volume 4, pages 96–104. IRE, New York, 1960.

[85] Yong Zeng, Yupu Yang, and Liang Zhao. Pseudo nearest neighbor rule for pattern classification. *Expert Syst. Appl.*, 36(2):3587–3595, March 2009.

[86] Yong Zhou, Youwen Li, and Shixiong Xia. An improved knn text classification algorithm based on clustering. *JCP*, pages 230–237, 2009.