# Clasificadores eficaces basados en algoritmos rápidos de búsqueda del vecino más cercano

Departamento de Lenguajes y Sistemas Informáticos

Universidad de Alicante

Francisco Moreno Seco

Dirigida por: Luisa Micó Andrés
Jose Oncina Carratalá

24 de febrero de 2004

# Índice general

# Índice de cuadros

# Índice de figuras

# Prólogo

El paradigma clásico del Reconocimiento de Patrones (o de Formas) es el de un sistema que obtiene a través de unos sensores información acerca de un objeto y lo clasifica en una clase conocida. Existen muchas formas de representar la información que se extrae de un objeto, y una de las más comunes es la de representarlo mediante un vector de medidas (o *características*). El Reconocimiento Estadístico de Formas trata de construir clasificadores basándose en la variabilidad estadística del vector de características, bien estimando los parámetros de la distribución estadística de dicho vector (aproximación paramétrica), o bien estimando directamente la probabilidad a posteriori de la clase (aproximación no paramétrica). Uno de los sistemas de clasificación no paramétricos más simples es el clasificador basado en la regla del vecino más cercano, que consiste en clasificar el objeto desconocido (*muestra*) en la clase de su vecino más cercano según una medida de disimilitud o *distancia*.

La regla del vecino más cercano (abreviada frecuentemente como NN en inglés) es una regla muy sencilla de implementar, con unas propiedades estadísticas bien establecidas, por lo que resulta muy atractiva en muchas tareas reales de clasificación. Además, dicha regla se puede extender utilizando para la clasificación los $k$ vecinos más cercanos, lo cual suele mejorar las tasas de acierto en la clasificación. Por otro lado, el problema de la búsqueda del vecino más cercano es común a otras disciplinas además del reconocimiento de formas, como pueden ser las búsquedas en grandes bases de datos, la "minería" de datos (*data mining* en inglés), la geometría computacional, etc. Dado que la búsqueda exhaustiva, uno por uno, del vecino más cercano es en muchos casos muy costosa, en estos campos se ha desarrollado un gran interés por encontrar soluciones eficientes a este problema, lo cual ha producido una gran cantidad de algoritmos rápidos para encontrar el vecino más cercano. Prácticamente todos estos algoritmos se pueden extender fácilmente para encontrar los $k$ vecinos más cercanos, con un coste computacional promedio adicional que depende del valor de $k$, según se ha comprobado experimentalmente.

Muchos de los algoritmos rápidos para encontrar el vecino más cercano se basan en seleccionar en cada paso un candidato a vecino más cercano y calcular su distancia a la muestra, para a continuación eliminar o podar parte del conjunto de entrenamiento. Al final del proceso, el vecino más cercano será el candidato más cercano a la muestra.

El primer objetivo de esta tesis consiste en el estudio de las posibilidades de un algoritmo rápido de búsqueda del vecino más cercano, el LAESA, para obtener una disminución en la tasa de error sin coste adicional, utilizando para ello $k$ vecinos (aunque no sean los $k$ más cercanos); una vez abordado este objetivo, las ideas aplicadas a este algoritmo se han extendido a otros algoritmos con resultados similares e incluso mejores, y se han generalizado en una nueva regla de clasificación: la regla de los $k$ vecinos seleccionados más cercanos. La idea básica de esta regla es utilizar para la clasificación los $k$ candidatos más cercanos de entre los seleccionados por el algoritmo de búsqueda. Una buena parte del trabajo de esta tesis se ha dedicado a explorar las posibilidades y el comportamiento de esta regla, que produce tasas de acierto en la clasificación similares a las de la regla de los $k$ vecinos más cercanos pero con el coste computacional de la regla del vecino más cercano.

## Estructura de la tesis

El primer capítulo es una pequeña introducción al reconocimiento de formas y una “hoja de ruta” para situar al lector en los temas tratados en esta tesis, relacionados con la regla del vecino más cercano. A continuación, en el capítulo 2 se estudia con detalle algunos de los algoritmos más conocidos para la búsqueda eficiente del vecino más cercano, sobre los que se aplicará en un capítulo posterior la nueva regla de clasificación. Este capítulo describe con detalle todos los algoritmos y puede servir de ayuda para comprender o implementar alguno de ellos.

La primera de las aportaciones de esta tesis se presenta en el capítulo 3, en el que se estudian distintas variantes del algoritmo LAESA para mejorar la clasificación utilizando $k$ vecinos (no necesariamente los $k$ más cercanos). La última de estas variantes supuso la aparición de la regla de los $k$ vecinos seleccionados más cercanos, que es la principal aportación de este trabajo y se estudia con detalle en el capítulo 4. La tercera y última aportación de la tesis (capítulo 5) es la extensión de algunas técnicas conocidas para la búsqueda aproximada en espacios de vectores a algunos algoritmos válidos para espacios métricos en general, no restringidos a espacios de vectores. Finalmente, en el capítulo 6 se presenta un pequeño resumen de las aportaciones realizadas y se proponen líneas de trabajo futuras.

## Agradecimientos

Durante los años que he dedicado a trabajar en esta tesis han sido muchas las personas que me han animado a continuar y terminar este trabajo, y a todas ellas debo un sincero agradecimiento. Aún a riesgo de dejar de mencionar a algunas de ellas (que espero me perdonen), debo agradecer en particular su ayuda a varias personas: en primer lugar, mi más sincero y profundo agradecimiento a Luisa y Jose, sin cuyas buenas ideas y consejos no habría podido ni siquiera empezar esta tesis. Muchas de las ideas importantes de esta tesis son suyas y me las han cedido generosamente, y además han sabido dirigirme siempre de la forma más correcta, adaptándose a mi forma de trabajar, lo cual no siempre ha sido tarea fácil.

Debo agradecer también a mis compañeros del departamento de Lenguajes y Sistemas Informáticos su ayuda y ánimo en todo momento; en particular, a los equipos directivos que me han concecido sucesivas reducciones docentes y me han facilitado la tarea de realizar la tesis; a los técnicos Miguel Angel Varó y Miguel Angel Baeza que siempre me han ayudado con la mejor de las maneras; a mis compañeros de la secretaría, José Ramón Lillo y Vicente Ferri, que me han ayudado con muchos papeles y viajes, también siempre con la mejor voluntad. De entre los muchos compañeros que alguna vez me han echado una mano debo agradecer especialmente a Mikel L. Forcada los años que trabajamos juntos y lo mucho que aprendí de él, especialmente en lo referente a la escritura de artículos científicos; a Jorge Calera debo agradecer especialmente sus valiosas explicaciones de conceptos estadísticos; a José Manuel Iñesta su constante ayuda y confianza que me han permitido trabajar, de forma más o menos estrecha, con David Rizo, Pedro Ponce de León y Antonio Pertusa, a quienes también agradezco su colaboración; a Eva Gómez, que me facilitó muchísimo el trabajo con los caracteres manuscritos. Además de los mencionados anteriormente, he compartido almuerzos, comidas, cafés, reuniones y, en general, buenos ratos con otros compañeros como Pedro, Juan Antonio, Juanra, Pepe y Emilio, algo que les agradezco sinceramente.

Finalmente, debo agradecer a mi familia y amigos sus innumerables muestras de apoyo en todo momento, y en particular a mis padres por sus esfuerzos por proporcionarme una buena educación y unos buenos valores, entre otras muchas cosas. Por último quiero expresar mi infinita gratitud a mi compañera Alicia, por lo mucho que hemos compartido juntos.

*Francisco Moreno Seco*

# Parte I

# Introducción

# Capítulo 1

# Reconocimiento de formas y clasificación

Esta tesis se enmarca dentro del campo del Reconocimiento de Formas (o Reconocimiento de Patrones, *Pattern Recognition* en inglés), que es una materia que trata de imitar en la medida de lo posible la facultad de muchos seres vivos de reconocer y clasificar objetos del mundo real. Para realizar esta tarea, los seres vivos reciben una serie de señales provenientes del objeto (principalmente a través de los sentidos de la vista y del oído) y, en función de su conocimiento previo, lo clasifican en alguna de las categorías conocidas.

Las técnicas del Reconocimiento de Formas se pueden aplicar a una gran variedad de campos como la visión artificial, la robótica, la biomedicina, etc. Dentro de esta disciplina se distinguen dos ramas: el *Reconocimiento Estadístico (o Geométrico) de Formas*, en el que se asume que los objetos se representan mediante una serie de atributos que siguen alguna distribución estadística, y el *Reconocimiento Sintáctico (o Estructural)*, en el que los objetos tienen una estructura compleja modelizable mediante algún lenguaje o gramática. En concreto, el trabajo de esta tesis está relacionado con la regla de clasificación del vecino más cercano, que está enmarcada dentro del Reconocimiento Estadístico de Formas,[1] por lo que no trataremos en esta tesis la rama Sintáctica del Reconocimiento de Formas.

## 1.1. Diseño de un sistema de Reconocimiento de Formas

Un sistema típico de reconocimiento estadístico de formas consta de varias partes (véase la figura 1.1):

[1] Recientemente se ha publicado un artículo (Duin et al., 2002) sobre esta materia que propone además líneas futuras de trabajo.

1. **Adquisición de datos:** lo primero que debe realizar el sistema es obtener información del objeto desconocido a partir de los sensores de que disponga, construyendo un patrón o representación (que a partir de ahora llamaremos *muestra*); aunque es posible representar patrones mediante cadenas de caracteres, árboles y otras estructuras, lo más frecuente es que el patrón esté formado por un vector de números (o de valores que pueden ser codificados como números, como por ejemplo el color), en general un vector $\mathbf{x}$ en $\mathbb{R}^n$; el espacio de valores posibles $E$ para un patrón se conoce como *espacio de representación*. Los datos obtenidos a partir de los sensores deben filtrarse y normalizarse antes de ser procesados por el sistema. Cada componente del vector $\mathbf{x}$ recibe el nombre de *característica*; puesto que los posibles valores de cada característica tienen carácter aleatorio, se considera que $\mathbf{x}$ es un vector de variables aleatorias (*random vector*, Fukunaga (1990)).

2. **Extracción de características:** en muchos casos, los datos obtenidos directamente por los sensores pueden ser modificados o utilizados para obtener otras características (*extraer características*) que permiten que el sistema clasifique mejor. A partir de un vector $\mathbf{x}$ de dimensión $n$, se obtiene mediante transformaciones matemáticas otro vector $\mathbf{y}$ de dimensión $m$. También es frecuente *seleccionar* aquellas características que mejor discriminen las muestras, algo que se tiene que hacer al diseñar el clasificador. En general, cada característica de $\mathbf{y}$ se obtiene a partir de una o más características de $\mathbf{x}$.

3. **Clasificación:** suponiendo que los patrones pertenecen a una clase de un conjunto $\Omega$ de $c$ clases, $\omega_1, \omega_2, \ldots, \omega_c$, el sistema debe decidir a qué clase pertenece la muestra. Es posible también que el clasificador no pueda decidir a qué clase pertenece la muestra, en cuyo caso clasifica la muestra en una *clase de rechazo*. De manera más formal, un clasificador es una función $d : E \to \Omega$ tal que para todo patrón $x \in E$, $d(x) \in \Omega$. Se supone que los patrones de una misma clase están agrupados entre sí, y por tanto están cerca de otros patrones de su clase y lejos de los de otras clases.

El sistema de reconocimiento de formas se construye a partir de un conjunto de patrones $P$ denominado *conjunto de entrenamiento*. Si en dicho conjunto los patrones tienen asociada una clase (entonces se les suele llamar *prototipos*), se dice que el sistema utiliza *aprendizaje (o entrenamiento) supervisado*; si, por el contrario, los patrones no están etiquetados y lo que pretendemos es descubrir agrupaciones (clases) en el conjunto de entrenamiento (siendo el número de clases conocido o no), se habla de *aprendizaje no supervisado* o *clustering*, cuyo objetivo es etiquetar un conjunto de datos. En el resto de esta tesis supondremos que el conjunto de entrenamiento está etiquetado, y por tanto no hablaremos de aprendizaje no supervisado.

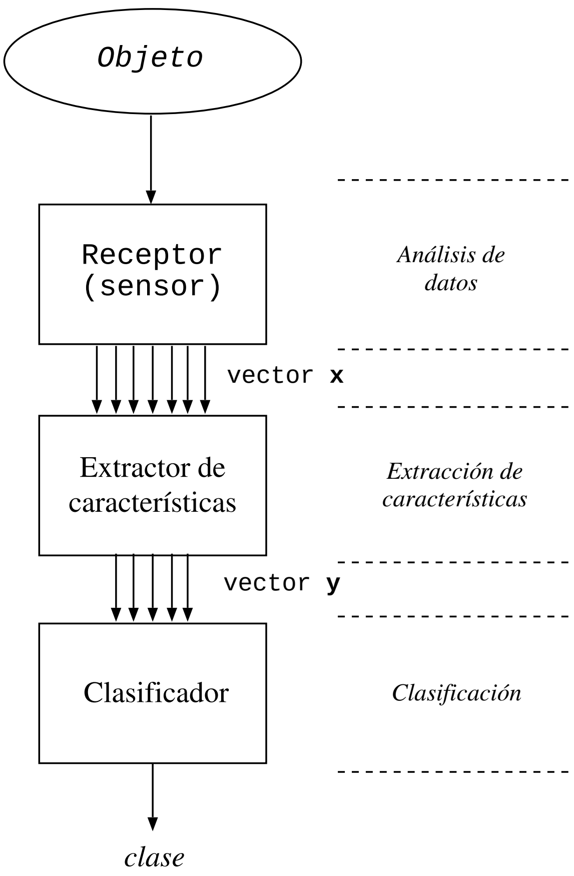


Figura 1.1: Estructura de un sistema de reconocimiento de formas (Cortijo, 2001).

El diseño del sistema suele comenzar con una fase de análisis de datos, seguida de la fase de diseño del clasificador propiamente dicho y de una última fase de evaluación de la eficacia del sistema, la cual puede producir cambios en una o en las dos fases previas.

### 1.1.1. Análisis de datos

La eficacia de un sistema de reconocimiento de formas depende en gran medida de la calidad de los datos utilizados para su construcción. En algunos casos, la dimensión de los patrones originales es tan elevada que la fase de clasificación es computacionalmente muy costosa; en otros casos, las tasas de acierto en la clasificación no son muy altas. Para intentar solventar estos problemas se puede realizar una *selección de características*, de forma que de todas las características presentes se escojan aquellas que mejor ayuden a la clasificación, y se desprecien aquellas cuya contribución no sea significativa. Esta fase puede suponer un aumento en la tasa de error, por lo que a veces es necesario llegar a un compromiso entre la reducción del tiempo de clasificación (reduciendo la dimensionalidad) y el aumento en la tasa de error.

Por otro lado, es posible construir funciones a partir de las características presentes en el patrón original $\mathbf{x}$, que permitan diferenciar mejor los patrones de distintas clases entre sí, lo cual produce un nuevo vector de características $\mathbf{y}$, en general no presentes en el patrón original (se suele hablar de un cambio en el espacio de representación). La elección de las funciones y las nuevas características se conoce como *extracción de características*.[2] Normalmente, una buena técnica de extracción de características produce una lista ordenada de nuevas características según su poder discriminante para la clasificación, de forma que en función de la dimensionalidad deseada del espacio de patrones se seleccionan más o menos.

Una vez seleccionadas aquellas características (originales o extraídas) que se desean para el sistema, se construye el módulo del sistema encargado de, a partir de los datos originales, construir el nuevo vector de características (el *extractor de características* de la figura 1.1).

### 1.1.2. Diseño del clasificador

El diseño de un clasificador se divide en varias etapas, cada una de las cuales influye en una cierta medida en el resultado final, por lo que suele ser frecuente rediseñar una o más etapas para mejorar dicho resultado.

[2] También se le suele dar el nombre de *extracción de características* a la fase del clasificador que obtiene en el momento de la clasificación los valores de las características nuevas a partir de las originales, obtenidas de los datos de los sensores; esta puede ser la causa de que algunos autores denominen a esta fase *generación de características* (como Theodoridis y Koutroumbas (1999)) o *elección de características* (Duda et al., 2001), para evitar confusiones.

De esta manera, el diseño de un clasificador se puede representar como un ciclo en el que en cada paso se toma una decisión y, al final, cuando se evalúa el clasificador, se revisan dichas decisiones y se vuelve a evaluar el clasificador (véase la figura 1.2). En la fase de diseño del clasificador una de las decisiones que debe tomarse es la de qué modelo de clasificador utilizar: por un lado, se puede optar por un *clasificador paramétrico*, que asume que la distribución estadística que siguen los datos es conocida, y trata de estimar los parámetros de dicha distribución. Por otro lado, se puede elegir un *clasificador no paramétrico*, en el que no se asume ninguna distribución estadística en particular (porque es posible que no se sepa que distribución siguen los datos), y se construye el clasificador utilizando únicamente la información de los datos del conjunto de entrenamiento. Esta última es la aproximación en la que se enmarca el trabajo de esta tesis, y es en la que nos centraremos a partir de ahora.

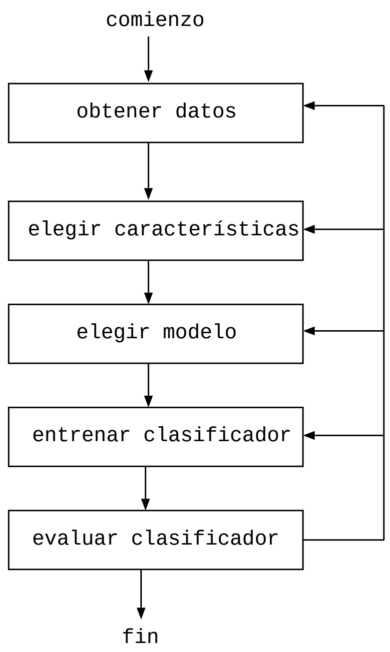


Figura 1.2: Ciclo de diseño de un sistema de reconocimiento de formas (Duda et al., 2001).

### Clasificador de Bayes

El clasificador de Bayes es, en circunstancias ideales, el mejor clasificador que se puede obtener y constituye una referencia a considerar cuando se diseña un clasificador. Dado un conjunto $\Omega$ de $c$ clases, $\omega_1, \omega_2, \ldots, \omega_c$, la *probabilidad a priori* de una clase $\omega_i$ se denota como $P(\omega_i)$; por otro lado, la *función de densidad de probabilidad* de una muestra $\mathbf{x}$ suponiendo que su clase es $\omega_i$ se denota como $p(\mathbf{x}|\omega_i)$. La regla de Bayes permite calcular la *probabilidad a posteriori* $P(\omega_i|\mathbf{x})$ (la probabilidad de que la clase sea $\omega_i$ cuando se observa $\mathbf{x}$) de la siguiente manera:

$$P(\omega_i|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{p(\mathbf{x})} \quad \text{donde} \quad p(\mathbf{x}) = \sum_{i=1}^{c} p(\mathbf{x}|\omega_i)P(\omega_i) \tag{1.1}$$

Suponiendo conocidas las probabilidades a priori y la función de densidad de probabilidad, el clasificador de Bayes asigna la muestra a la clase con mayor probabilidad a posteriori (Duda y Hart, 1973). El error obtenido por un clasificador de Bayes (*error de Bayes*) es el mínimo error posible y es el objetivo a alcanzar por cualquier clasificador.

### Clasificadores no paramétricos

Entre los distintos tipos de clasificadores no paramétricos se pueden destacar los que se basan en estimar las funciones de densidad de probabilidad, y los que estiman directamente la probabilidad a posteriori de la clase. Dentro de estos últimos, el más conocido es el clasificador basado en los $k$ vecinos más cercanos: si $k_i$ es el número de prototipos de la clase $i$ entre los $k$ más cercanos, la probabilidad a posteriori $P(\omega_i|\mathbf{x})$ se puede estimar como $\frac{k_i}{k}$ (Duda y Hart, 1973). De esta manera, el clasificador asigna la muestra $\mathbf{x}$ a la clase más frecuente de entre sus $k$ vecinos más cercanos, según una cierta medida de similitud o *distancia*. Si se denota como $\varepsilon^*$ el error de Bayes y como $\varepsilon$ el error de un clasificador basado en los $k$ vecinos más cercanos, se puede demostrar (Duda y Hart, 1973; Fukunaga, 1990) que, cuando el número de muestras tiende a infinito, este último error está acotado por el primero de la siguiente forma:

$$\varepsilon^* \leq \varepsilon \leq \varepsilon^*\Big(2 - \frac{c}{c-1}\varepsilon^*\Big) \leq 2\varepsilon^*$$

### Técnicas de clasificación no paramétricas basadas en distancias

Existen distintas técnicas o *reglas de clasificación* basadas en distancias. La más sencilla es la *regla de clasificación por distancia mínima*, en la que se parte del supuesto de que se ha elegido un representante $p_i$ por cada clase $i$, y clasifica la muestra $\mathbf{x}$ en aquella clase cuyo representante esté más cerca de ella, según la distancia definida en el espacio de representación.

La *regla de clasificación del vecino más cercano* es una extensión de la regla de clasificación por distancia mínima y parte de la suposición de que los prototipos de una misma clase están cerca unos de otros en el espacio de representación. Esta regla consiste en asignar a la muestra la clase de su vecino más cercano en el conjunto de entrenamiento.

La *regla de clasificación de los k vecinos más cercanos* es una extensión de la regla del vecino más cercano en la que se clasifica la muestra $\mathbf{x}$ en la clase más frecuente de entre los $k$ vecinos más cercanos. Estas dos reglas son muy sencillas de implementar y tienen unas propiedades estadísticas muy interesantes, como se ha comentado anteriormente.

Una regla de reciente aparición es la *regla de clasificación de los k vecinos de centroide más cercano* (Sánchez, 1998; Sánchez et al., 2000), que utiliza para la clasificación aquellos $k$ vecinos cuyo centroide se encuentre más cerca de la muestra. Es una regla especialmente adecuada cuando el conjunto entrenamiento es reducido y existe un cierto solapamiento entre las clases, pero necesita que los prototipos se representen como puntos en $\mathbb{R}^n$.

### 1.1.3. Evaluación del clasificador

El clasificador necesita un conjunto de datos, denominado *conjunto de entrenamiento* o *conjunto de aprendizaje*, a partir del cual clasifica muestras desconocidas. Para evaluar el error en la clasificación se utiliza un conjunto, llamado *conjunto de test*, de muestras etiquetadas que se presentan al clasificador para que éste las clasifique; si la clase asignada por el clasificador no coincide con la clase original de la muestra, se dice que el clasificador ha cometido un error. La tasa de error del clasificador se calcula como:

$$\frac{N_{err}}{N_{test}}$$

donde $N_{err}$ es el número total de errores cometidos por el clasificador, y $N_{test}$ es el tamaño del conjunto de test.

En la práctica, el conjunto de datos del que se dispone tiene un tamaño limitado $N$, por lo que para evaluar la eficiencia del clasificador es necesario dividir el conjunto en uno o más pares de conjuntos de entrenamiento y de test, y estimar el error de clasificación. Existen varias formas de dividir el conjunto de datos, y algunas de las más conocidas (véase el libro de Breiman et al. (1984) para más detalles) son:

**Partición (*holdout*):** consiste en dividir el conjunto de datos en dos conjuntos, uno para entrenamiento y otro para test (el error se estima únicamente a partir de un experimento).

**Validación cruzada (*cross-validation*):** consiste en realizar $m$ particiones del conjunto de datos en dos conjuntos (entrenamiento/test), de forma que todos los prototipos son utilizados para entrenamiento y para test (en particiones distintas). De esta forma, el error se estima como el promedio del error de los $m$ experimentos, lo cual resulta más adecuado ya que no depende tanto como en el método de partición de la elección concreta de los conjuntos de entrenamiento y de test. Cuando el conjunto de datos tiene un tamaño considerable ésta suele ser la elección más aconsejable.

**Dejando $k$ fuera (*leaving-$k$-out*):** en esta técnica, que es a la vez una forma de validación cruzada y una generalización de la conocida técnica *leaving-one-out*, se realizan $N/k$ particiones utilizando cada vez $N-k$ prototipos para entrenamiento y $k$ prototipos para test. Esta técnica es similar a la de validación cruzada, con $m = N/k$. Cuando $k = 1$, se está aprovechando para la estimación del error del clasificador prácticamente todo el conjunto de datos disponible, lo cual da una idea bastante precisa del comportamiento del clasificador en la práctica, con datos desconocidos. Esta alternativa es siempre recomendable, al menos cuando el conjunto de datos tiene un tamaño computacionalmente tratable.

**Nota bibliográfica:** Las referencias a los distintos métodos para dividir el conjunto de datos para después estimar el error de clasificación son muy antiguas: una de los primeros estudios sobre el método de partición (aparece citado en el libro de Duda y Hart (1973)) es un artículo de Highleyman (1962); un artículo proponiendo la técnica *leaving-one-out* como alternativa al método de partición es el de Lachenbruch y Mickey (1968) (se cita en los libros de Duda y Hart (1973) y Fukunaga (1990)).

## 1.2. Mejoras en la regla del vecino más cercano

Desde hace muchos años se han estudiado diferentes maneras de mejorar el comportamiento de la regla del vecino más cercano, incidiendo en dos aspectos:

1. **Tasa de acierto**: una forma de mejorar las tasas de acierto de un clasificador basado en la regla del vecino más cercano es eliminar aquellos prototipos que producen más errores de clasificación, que suelen ser aquellos situados relativamente lejos de los demás prototipos de su clase y cerca de los de otras clases. Estos prototipos se pueden eliminar aplicando un proceso conocido como *edición* (Devijver y Kittler, 1982), lo cual permite que las fronteras de decisión entre clases estén más "limpias" y el clasificador obtenga mejores tasas de acierto en la clasificación.

2. **Tiempo de clasificación**: existen diversas formas de rebajar el tiempo de clasificación, entre las que destacamos las siguientes:

   - Reducir el conjunto de entrenamiento al mínimo que permita una clasificación al menos tan correcta como la del conjunto inicial. Esta técnica se conoce como *condensado* (Devijver y Kittler, 1982), y tiene varios inconvenientes: necesita un tiempo de preproceso elevado, y cuando la dimensión o el número de clases son elevados apenas reduce el conjunto de entrenamiento.

- Reducir el tiempo de búsqueda del vecino más cercano, sin modificar el conjunto de entrenamiento. En el capítulo 2 se describen algunos de los algoritmos más conocidos para encontrar de forma rápida el vecino más cercano; en general, el objetivo es reducir el número de distancias que se calculan entre la muestra y los prototipos del conjunto de entrenamiento, sin incrementar demasiado el tiempo de procesamiento adicional. Muchos de estos algoritmos construyen en la etapa de preproceso un árbol que se recorre realizando algún tipo de poda durante la clasificación de la muestra desconocida.
- Reducir drásticamente el tiempo de búsqueda del vecino más cercano encontrando el vecino *aproximadamente* más cercano. Existen muchos trabajos en este sentido, pero el más citado es el de Arya et al. (1998), que se basa en encontrar un vecino cuya distancia a la muestra no sea mayor que $(1+\epsilon)d_{nn}$, siendo $d_{nn}$ la distancia al vecino más cercano. En el capítulo 5 se estudia con más detalle esta cuestión y se proponen modificaciones a varios algoritmos para realizar búsquedas aproximadas.

Estas técnicas no son incompatibles entre sí, por lo que es posible primero aplicar técnicas de edición y condensado al conjunto de entrenamiento para luego utilizar un algoritmo rápido para encontrar el vecino más cercano, aunque si el conjunto condensado es pequeño los algoritmos rápidos no mejoran el tiempo de clasificación de la búsqueda exhaustiva del vecino más cercano.

## 1.3. Esquema de situación del trabajo de la tesis

El trabajo de esta tesis está ampliamente relacionado con las reglas del vecino más cercano y de los $k$ vecinos más cercanos. La figura 1.3 muestra un esquema de la situación de la materia de esta tesis en el contexto del reconocimiento de formas, en el que aparece en dos apartados la regla de los $k$ vecinos más cercanos: por un lado, se puede utilizar, como las ventanas de Parzen, para la estimación de la función de densidad de probabilidad (Fukunaga, 1990); por otro lado, se puede utilizar para estimar directamente la probabilidad a posteriori. Las aportaciones de esta tesis están relacionadas con este segundo aspecto de la regla de los $k$ vecinos más cercanos.

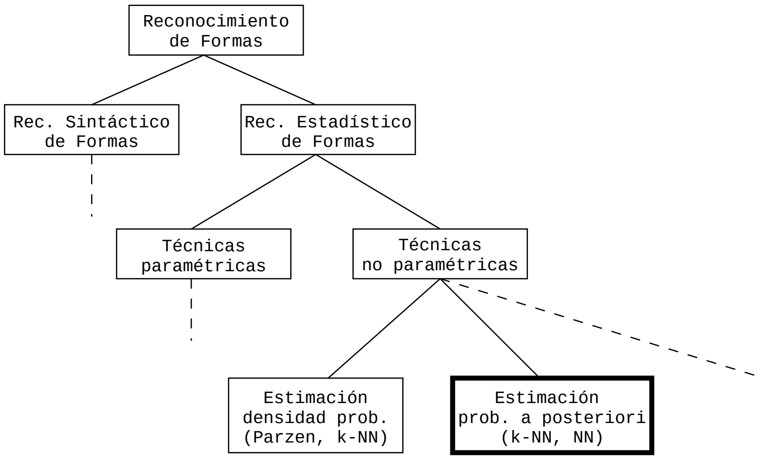


Figura 1.3: La regla NN en el contexto del reconocimiento de formas.

# Capítulo 2

# Algoritmos para la búsqueda del vecino más cercano

La regla de clasificación del vecino más cercano se utiliza en muchas tareas por su simplicidad y eficacia. El algoritmo más sencillo para implementar dicha regla es el conocido como *fuerza bruta* o *exhaustivo*, que calcula todas las distancias de la muestra desconocida a los prototipos del conjunto de entrenamiento y asigna a la muestra la clase de aquel prototipo cuya distancia sea mínima. Sin embargo, hay una serie de situaciones que hacen poco aconsejable la utilización de este algoritmo en la práctica: cuando el conjunto de entrenamiento es grande, o cuando el cálculo de la distancia es muy costoso en tiempo; además, en algunos casos la alta dimensionalidad de los datos hace que incluso la distancia euclídea resulte ser una distancia costosa y por tanto la búsqueda exhaustiva tampoco sea una opción válida.

Durante muchos años se han ido desarrollando algoritmos eficientes para encontrar el vecino más cercano que evitaran recorrer exhaustivamente todo el conjunto de entrenamiento calculando todas las distancias. Aunque en Reconocimiento de Formas se utiliza la búsqueda del vecino más cercano principalmente para la clasificación (también se utiliza en el agrupamiento o *clustering*), en otras comunidades científicas esta técnica también es relevante, como por ejemplo en las dedicadas a las bibliotecas digitales, grandes bases de datos y buscadores en Internet, que necesitan localizar el vecino más cercano en grandes conjuntos de datos como respuesta a preguntas del usuario.[1] En este contexto, a partir de la pregunta del usuario se construye un prototipo y se presentan como solución a su pregunta sus vecinos más cercanos en la base de datos.

[1]Precisamente uno de los fundadores del buscador `Google` es Sergey Brin, autor también de un algoritmo rápido para encontrar el vecino más cercano, el GNAT (Geometric Near-neighbor Access Tree, Brin (1995)). Ricardo Baeza-Yates ha dirigido el desarrollo un buscador, `buscopio` (con menor éxito) y también ha trabajado en algoritmos rápidos, y recientemente ha publicado un estudio muy interesante con otros autores (Chavez et al., 2001).

En la biblografía existen multitud de artículos (provenientes de distintos campos) proponiendo algoritmos eficientes para la búsqueda del vecino más cercano. Dasarathy (1991) realizó la primera recopilación de algoritmos sobre el problema de la búsqueda del vecino más cercano. Posteriormente se hizo una revisión que contiene una completa clasificación y un excelente estudio de los algoritmos más importantes para la búsqueda del vecino más cercano en espacios métricos generales (Chavez et al., 2001) (aquellos en los que no se supone que los prototipos se representan como vectores), y también se mencionan (aunque no se estudian con detalle) los algoritmos más conocidos para espacios de vectores.

Muchos de estos algoritmos encajan dentro de un esquema de búsqueda conocido como esquema de aproximación y eliminación (Ramasubramanian y Paliwal, 2000), y entre éstos algunos de los más conocidos son el k-d tree (Bentley, 1975; Friedman et al., 1977), el de Fukunaga y Narendra (Fukunaga y Narendra, 1975), el vp-tree (Yianilos, 1993) y el GNAT (Brin, 1995); otros algoritmos que encajan en este esquema pero son menos conocidos son los de la familia AESA (Vidal, 1986, 1994; Micó et al., 1994, 1996), que resultan muy eficientes cuando la distancia es costosa. Los algoritmos mencionados anteriormente (y otros muchos) se pueden extender fácilmente para encontrar los $k$ vecinos más cercanos.

Existen otros dos tipos de búsqueda que pueden hacerse con algunos de estos algoritmos: por un lado, la búsqueda del vecino aproximadamente más cercano de la muestra (Arya et al., 1998), que es un vecino que se encuentra como mucho a una distancia $(1+\epsilon)d_{nn}$ de la muestra, donde $d_{nn}$ es la distancia al vecino más cercano. Por otro lado, se pueden utilizar estos algoritmos para encontrar todos los prototipos del conjunto de entrenamiento que se encuentren como mucho a una distancia $r$ de la muestra; este tipo de búsqueda se conoce como *búsqueda por rango* y es la que se suele utilizar en búsquedas en grandes bases de datos.

En este capítulo se estudian algunos conceptos sobre espacios de representación y se presenta el esquema de aproximación y eliminación; a continuación, se describe con detalle cada uno de los algoritmos mencionados en el párrafo anterior, puesto que las aportaciones de esta tesis (capítulos 3, 4 y 5) tratan de mejoras en la clasificación utilizando estos algoritmos.

## 2.1. Espacios de representación

En muchos casos, los algoritmos propuestos para encontrar el vecino más cercano se basan en que los prototipos y la muestra desconocida son vectores en un espacio $n$-dimensional, $\mathbb{R}^n$. Sin embargo, la técnica del vecino más cercano se ha ido aplicando a otros problemas en los que no es posible o razonable encontrar una representación como vectores de los objetos a clasificar, y en los que existe alguna forma de medir la *distancia* entre dos objetos. Cuando dicha distancia cumple las propiedades de una *métrica*, el espacio de representación de los objetos se dice que es un espacio métrico.

**Definición:** Un espacio $E$ se dice que es un *espacio métrico* si existe una medida de similitud o *distancia* $d(\cdot,\cdot)$ entre dos elementos de dicho espacio que cumple las condiciones de una métrica:

1. Reflexividad: $\forall a, b \in E, d(a,b) = 0 \iff a = b$
2. Simetría: $\forall a, b \in E, d(a,b) = d(b,a)$
3. Positividad estricta: $\forall a, b \in E, a \neq b, d(a,b) > 0$
4. Desigualdad triangular: $\forall a, b, c \in E, d(a,b) + d(b,c) \geq d(a,c)$

Si la propiedad de la positividad se relaja a $d(a,b) \geq 0$, se dice que el espacio es *pseudo-métrico*. En general, es posible tratar todos aquellos elementos de un espacio pseudo-métrico para los que $d(a,b) = 0$ como un único elemento y de esta forma se obtendría un espacio métrico.

**Definición:** Un espacio métrico $E$ en el que los elementos se representan como puntos en $\mathbb{R}^n$ o vectores de dimensión $n$, se dice que es un *espacio de vectores*.[2]

Cuando un algoritmo hace uso de las coordenadas de cada prototipo se dice que trabaja en un espacio de vectores, mientras que si no utiliza las coordenadas y solamente se basa en la noción de distancia entre prototipos se dice que trabaja en un espacio métrico general o simplemente en un espacio métrico. Las distancias que se emplean normalmente en espacios de vectores son las llamadas métricas de Minkowski (aunque hay muchas más); dado un par de puntos $x, y \in \mathbb{R}^n$, la distancia se define como:

$$L_p = \Big( \sum_{i=1}^{n} |x_i - y_i|^p \Big)^{\frac{1}{p}}$$

Las distancias más utilizadas son la $L_1$, la $L_2$ (la distancia euclídea) y la $L_\infty$, que se define como:

$$L_\infty = \mathbf{max}\, |x_i - y_i|$$

La distancia euclídea es probablemente la que más frecuentemente se utiliza en espacios de vectores, y cumple las propiedades de una métrica.

Todos aquellos algoritmos que pueden funcionar en un espacio métrico pueden funcionar en un espacio de vectores usando una distancia que cumpla

[2] A veces se utiliza el nombre *espacio vectorial* para este tipo de espacios de representación, pero esta denominación no es correcta, un espacio vectorial requiere más propiedades. En inglés se suele utilizar *vector space*.

las propiedades de una métrica, como por ejemplo la distancia euclídea, pero lo contrario no es cierto: los algoritmos que utilizan coordenadas no se pueden usar en espacios que no sean de vectores. En muchas tareas reales de clasificación no es posible encontrar una representación adecuada utilizando vectores, y se utilizan representaciones basadas en cadenas, grafos, árboles, etc.; en estos casos es necesario utilizar algoritmos que se basen únicamente en la distancia y no utilicen las coordenadas.

En los algoritmos rápidos de búsqueda del vecino más cercano se suele utilizar una estructura de datos auxiliar para almacenar el conjunto de entrenamiento, normalmente un árbol. Cuando la distancia empleada es la distancia euclídea, el coste de calcular una distancia entre dos prototipos es muy pequeño comparado con el coste de recorrer dicha estructura de datos. Sin embargo, en espacios métricos en los que los prototipos no sean vectores, las distancias empleadas pueden resultar muy costosas en tiempo (como por ejemplo la distancia de edición entre cadenas, cuyo coste es proporcional al producto de las longitudes de las dos cadenas), y por tanto el número de distancias calculadas por el algoritmo de búsqueda es un factor muy importante para la eficiencia del mismo. En el trabajo de Chavez et al. (2001) sobre algoritmos de búsqueda en espacios métricos se sigue esta aproximación y se considera el número de distancias calculadas como el coste principal del algoritmo.

## 2.2. Búsqueda por aproximación y eliminación

Ramasubramanian y Paliwal (2000) hacen un estudio exhaustivo del esquema de búsqueda del vecino más cercano por aproximación y eliminación, que es la base de los algoritmos de la familia AESA y de otros muchos; de hecho, estos autores diseñaron el esquema a partir de los algoritmos de la familia AESA, y a continuación demostraron que otros muchos algoritmos encajan en él. Por ejemplo, en ese artículo se reformula un algoritmo clásico (Friedman et al., 1975) para adaptarlo al esquema de aproximación y eliminación. Todos los algoritmos utilizados en esta tesis se pueden considerar como ejemplos de la aplicación de este esquema.

El esquema general de búsqueda por aproximación y eliminación puede verse en la figura 2.1, y puede resumirse como:

1. De entre los prototipos del conjunto de entrenamiento se selecciona un candidato a vecino más cercano (aproximación).

2. A continuación, se calcula su distancia $d$ a la muestra.

3. Si dicha distancia resulta ser menor que la del vecino más cercano hasta el momento, $d_{nn}$, se actualiza el vecino más cercano y se eliminan del conjunto de entrenamiento aquellos prototipos que no puedan estar dentro de una hiperesfera de radio $d$ y con centro en la muestra

$d_{nn} := \infty$
**hacer** hasta que el conjunto de entrenamiento $P$ esté vacío
  $p_i := argmin_{p \in P}\text{Aprox}(x,p)$ *// aproximación*
  $d := d(x,p)$ *// distancia a la muestra*
  **si** $d < d_{nn}$ **entonces**
    $nn := p$ ; $d_{nn} := d$ *// actualización del vecino más cercano*
    $P := P - \{\ q : q \notin E(x, d_{nn})\ \}$ *// eliminación*
  **fsi**
**fhacer**

Figura 2.1: Esquema de búsqueda por aproximación y eliminación.

(eliminación), es decir, se eliminan aquellos prototipos que no puedan estar más cerca de la muestra que el vecino más cercano actual.[3]

4. Se repiten los pasos anteriores hasta que no queden prototipos por seleccionar en el conjunto de entrenamiento, bien porque hayan sido previamente seleccionados o bien porque hayan sido eliminados.

El objetivo de un algoritmo basado en este esquema es que el ahorro en distancias calculadas (por la eliminación de prototipos) compense el coste computacional de los procesos de aproximación y eliminación. Además, se puede comprobar que cuanto mejor sea el criterio de aproximación, más rápidamente se encontrará el vecino más cercano y más prototipos se podrá eliminar en pocos pasos. Los algoritmos de la familia AESA son un ejemplo de la aplicación de este esquema al proceso de búsqueda y consiguen una reducción muy importante en el número de distancias calculadas.

Otros algoritmos que se estudian en este capítulo están basados en árboles que se recorren durante la búsqueda del vecino más cercano. En estos algoritmos, la fase de aproximación es la selección del nodo a visitar en el árbol, y la eliminación es la poda de una rama de dicho árbol.

## 2.3. Algoritmos rápidos de búsqueda del vecino más cercano

La mayoría de los algoritmos rápidos utilizan una estructura de datos para almacenar el conjunto de entrenamiento, generalmente un árbol. En esta sección se presentan algunos de los algoritmos basados en árboles más conocidos: k-d tree (Bentley, 1975; Friedman et al., 1977), Fukunaga y Naren-

[3]Determinar exactamente si un prototipo está dentro o fuera de la hiperesfera puede implicar calcular su distancia a la muestra, por lo que muchos algoritmos lo que hacen es obtener de alguna manera una cota inferior de dicha distancia y eliminar aquellos prototipos cuya cota inferior sea mayor que $d$.

dra (Fukunaga y Narendra, 1975), vp-tree (Yianilos, 1993) y GNAT (Brin, 1995). El artículo de Chavez et al. (2001) contiene un estudio más exhaustivo de estos algoritmos y otros similares, principalmente de aquellos que son válidos para espacios métricos en general.

Todos los algoritmos de esta sección se han utilizado en los experimentos de esta tesis (véase el capítulo 4) y todos encajan en un esquema de aproximación y eliminación; el k-d tree utiliza las coordenadas para la aproximación y no selecciona candidatos a vecino más cercano hasta que llega a las hojas en el árbol, pero también puede considerarse que encaja en este esquema. Por otro lado, el GNAT se plantea únicamente para la búsqueda por rango (buscar todos los vecinos que estén como mucho a una distancia previamente fijada $r$ de la muestra), pero se ha adaptado para la búsqueda del vecino más cercano simplemente haciendo que el rango sea en cada paso la distancia al vecino más cercano hasta el momento.

Muchos de los algoritmos que se describen en este capítulo realizan la búsqueda del vecino más cercano utilizando una función recursiva que recorre un árbol. Estas funciones van actualizando en cada paso, cuando es posible, el vecino más cercano ($nn$ en las figuras) y su distancia a la muestra ($d_{nn}$); en los dos casos se ha considerado que son parámetros de entrada y salida, aunque en el caso de $nn$ se trata de un valor únicamente de salida de las funciones, que va variando en las distintas llamadas recursivas.[4] En el caso de $d_{nn}$, su valor se utiliza para dirigir la búsqueda, lo cual obliga a darle un valor inicial, que en todos los casos debe ser $\infty$; esto implica que hasta que el algoritmo no calcule la primera distancia (en muchos casos cuando visita una hoja), no se realiza ningún tipo de poda en la búsqueda.

En general, los algoritmos para la búsqueda del vecino más cercano se pueden extender fácilmente para obtener los $k$ vecinos más cercanos. Los algoritmos que se presentan en esta sección se han extendido como proponen sus respectivos autores para encontrar los $k$ vecinos más cercanos haciendo unas sencillas modificaciones (veáse la sección 2.5 para más detalles). En las figuras correspondientes a los algoritmos de búsqueda se señalan con $(a)$ y $(b)$ los puntos en los que habría que hacer dichas modificaciones, que consisten en:

$(a)$ Cuando se calcula la distancia de la muestra desconocida a un prototipo, en lugar de actualizar si es necesario el vecino más cercano se almacenan ordenadamente en un vector $\mathbf{v}$ los $k$ vecinos más cercanos que se haya encontrado hasta el momento (y sus distancias).[5] Cuando la búsqueda termine tendrá los $k$ vecinos más cercanos a la muestra.

[4] La implementación resulta más sencilla si es un parámetro de entrada/salida en lugar de ser el valor devuelto por la función.

[5] Evidentemente, en los primeros pasos de la búsqueda el vector tendrá menos de $k$ componentes, pero en cuanto se hayan calculado $k$ distancias tendrá los $k$ más cercanos hasta el momento.

(*b*) Cuando se intenta aplicar algún criterio de eliminación en el que en el algoritmo original se utiliza la distancia al vecino más cercano, se debe usar en su lugar la distancia al último elemento válido de $\mathbf{v}$.

### 2.3.1. K-Dimensional Tree (k-d tree)

El k-d tree (Bentley, 1975; Friedman et al., 1977) es un árbol clásico entre los que se utilizan para la búsqueda del vecino más cercano. El nombre k-d tree viene de *k-dimensional tree*, en el que la $k$ representa la dimensión de los datos del espacio de representación. El k-d tree es un árbol binario que contiene en cada nodo intermedio información acerca de una coordenada que divide en dos el conjunto de datos del subárbol correspondiente al nodo, y en las hojas contiene puñados (*buckets*) de prototipos. Durante la fase de clasificación se recorre el árbol siguiendo un esquema de ramificación y poda para encontrar el vecino más cercano. Tanto en el preproceso (construcción del árbol) como en la clasificación propiamente dicha se utilizan las coordenadas de los prototipos, por lo que esta estructura de datos y el algoritmo de búsqueda necesitan un espacio de representación con vectores (espacio de vectores). Aunque se han desarrollado muchas mejoras y optimizaciones, es el algoritmo de referencia cuando se utilizan distancias euclídeas. Sin embargo, tiene un problema conocido: cuando la dimensionalidad de los datos aumenta, el algoritmo visita prácticamente todos los nodos del árbol (se dice que el algoritmo degenera con la dimensionalidad).

#### Construcción del k-d tree

La idea principal del algoritmo para la construcción del k-d tree a partir de un conjunto de prototipos $P$ es la siguiente: encontrar un hiperplano que divida el conjunto $P$ en dos subconjuntos y proceder recursivamente con los subconjuntos. El principal aspecto a resolver es la elección del hiperplano y de la coordenada que va a servir para dirigir la búsqueda a un lado u otro del hiperplano, la *coordenada discriminante*.

Para intentar conseguir que cualquier prototipo tenga la misma probabilidad de estar a un lado o a otro del hiperplano y por tanto que el árbol resulte lo más equilibrado posible, se suele elegir el hiperplano de forma que se sitúe en la mediana de los valores de la coordenada discriminante. Además, la coordenada discriminante debe ser aquella que tenga una mayor amplitud, es decir, aquella para la que la diferencia entre la coordenada mínima y máxima sea la mayor en valor absoluto. El árbol se construye de la siguiente forma: en cada nodo, que representa un conjunto de prototipos (el nodo raíz representa a todo el conjunto de entrenamiento), se elige la coordenada discriminante y se obtiene la mediana de los valores de dicha coordenada en los prototipos del conjunto; a continuación, se divide dicho conjunto en dos subconjuntos utilizando la mediana, situando en un subconjunto aquellos prototipos para los que el valor de la coordenada discriminante sea menor o igual que el de la mediana, y en el otro subconjunto los prototipos cuya coordenada discriminante sea mayor que la mediana. A continuación, se crean recursivamente los árboles asociados a cada uno de los subconjuntos. El proceso termina cuando el tamaño del conjunto de prototipos es menor o igual que el valor fijado como tamaño de un puñado, y en este caso el nodo sería una hoja.

En la figura 2.2 aparecen las funciones necesarias para la construcción del árbol. El tamaño del puñado se ha fijado en 5 en la implementación utilizada[6] en esta tesis, que es un valor pequeño comparado con el propuesto en el algoritmo original (que proponía tamaños del orden de 50) pero nos permite obtener árboles de tamaño comparable a los obtenidos con otros algoritmos, en los que muchas veces las hojas contienen únicamente un prototipo.

#### Búsqueda en el k-d tree

El proceso de búsqueda en el k-d tree es recursivo: dada una muestra desconocida $x$, en un nodo cualquiera del árbol (que no sea una hoja) se compara la coordenada de $x$ que es discriminante para ese nodo (que denotaremos como $c$) con el valor de corte $v$ (la mediana de las coordenadas discriminantes), y se procede en la dirección más cercana según esa coordenada. Si $x[c] + d_{nn} \leq v$ (donde $d_{nn}$ es la distancia al vecino más cercano hasta el momento), el hijo derecho de ese nodo no puede contener al vecino más cercano y por tanto no es necesario buscarlo en ese nodo; de forma similar, si $x[c] - d_{nn} \geq v$ el hijo izquierdo no es necesario visitarlo. Si el nodo es una hoja, la muestra se compara con todos los prototipos contenidos en ella. En la figura 2.3 se muestra la función recursiva de búsqueda.

### 2.3.2. Algoritmo de Fukunaga y Narendra

Este algoritmo (Fukunaga y Narendra, 1975) fue uno de los primeros en ser ideado para espacios métricos generales, y se basa, como el k-d tree y muchos otros, en la construcción de un árbol a partir del conjunto de entrenamiento. Dicho conjunto se divide en $l$ subconjuntos, que a su vez son divididos en $l$ subconjuntos, etc, hasta llegar a un cierto nivel; por tanto, cada hoja del árbol contiene un subconjunto de prototipos. Cada nodo $p$ del árbol tiene $l$ hijos (si no es una hoja) y representa a uno de estos subconjuntos ($S_p$); además, contiene un representante $M_p$, y un radio $R_p$ definido como el máximo de las distancias de $M_p$ a los elementos del subconjunto.

Para la construcción de los subconjuntos se sugiere la utilización de un algoritmo de agrupamiento como puede ser el de las $k$-medias. Debido a esta

[6] La implementación se ha realizado a partir de un paquete llamado Ranger (Murphy y Skiena, 1996), que también contiene otras variantes del k-d tree y una implementación del vp-tree.

**funcion** CrearArbol [k-d tree]
  **entrada** $P$ (conjunto de prototipos)
  **salida** $nodo$ (árbol que representa a $P$)
**comienzo**
  **si** $|P| \leq$ TamañoPuñado **entonces**
    $nodo$ := NodoHoja($P$)
  **si no**
    $nodo$ := Nuevo()
    $coordDiscrim$ := MaximaAmplitud($P$)
    $nodo.coordDisc := coordDiscrim$
    $mediana$ := $\text{Mediana}_{p \in P}\ p[coordDiscrim]$
    $Iz := \{ p \in P \mid p[i] \leq mediana \}$
    $De := \{ p \in P \mid p[i] > mediana \}$
    $nodo.valorDeCorte := mediana$
    $nodo.iz$ := CrearArbol($Iz$)
    $nodo.de$ := CrearArbol($De$)
  **fsi**
  **devuelve** $nodo$
**fin** CrearArbol [k-d tree]

**funcion** MaximaAmplitud [k-d tree]
  **entrada** $P$ (conjunto de prototipos)
  **salida** $c$ (coordenada de máxima amplitud)
**comienzo**
  $maxAmplitud := -\infty$
  **para** $1 \leq i \leq$ NumCoordenadas **hacer**
    $max := -\infty$
    $min := \infty$
    **para** $p \in P$ **hacer**
      **si** $p[i] < min$ **entonces**
        $min := p[i]$
      **fsi**
      **si** $p[i] > max$ **entonces**
        $max := p[i]$
      **fsi**
      **si** $|max - min| > maxAmplitud$ **entonces**
        $maxAmplitud := |max - min|$
        $c := i$
      **fsi**
    **fpara**
  **fpara**
  **devuelve** $c$
**fin** MaximaAmplitud [k-d tree]

Figura 2.2: Construcción del k-d tree.

**funcion** buscar [k-d tree]
    **entrada** $n$ (árbol)
        $x$ (muestra a clasificar)
    **entrada/salida** $d_{nn}$ (distancia al vecino más cercano)
        $nn \in P$ (vecino más cercano de $x$)
**comienzo**
    **si** EsHoja($n$) **entonces**
        **para todo** $p$ contenido en $n$ **hacer**
$(a)$             $d := d(x,p)$
            **si** $d < d_{nn}$ **entonces**
                $d_{nn} := d$ ; $nn := p$
            **fsi**
        **fpara**
    **si no**
        **si** $x[n.coordDisc] < n.valorDeCorte$ **entonces**
            buscar($n.iz$,$x$,$d_{nn}$,$nn$)
$(b)$             **si** $x[n.coordDisc] + d_{nn} > n.valorDeCorte$ **entonces**
                buscar($n.de$,$x$,$d_{nn}$,$nn$)
            **fsi**
        **si no**
            buscar($n.de$,$x$,$d_{nn}$,$nn$)
$(b)$             **si** $x[n.coordDisc] - d_{nn} < n.valorDeCorte$ **entonces**
                buscar($n.iz$,$x$,$d_{nn}$,$nn$)
            **fsi**
        **fsi**
    **fsi**
**fin** buscar [k-d tree]

Figura 2.3: Búsqueda en un k-d tree optimizado. Los argumentos $d_{nn}$ y $nn$ son de entrada y salida, se utilizan para elegir el camino en el árbol y se actualizan cuando se llega a las hojas, lo cual puede ocurrir en cada llamada recursiva. El valor inicial de $nn$ no es importante puesto que no se utiliza en el proceso de búsqueda, pero el valor de $d_{nn}$ sí se utiliza y debería inicializarse a $\infty$. De esta manera, la llamada inicial debería ser: $d_{nn} := \infty$ ; buscar($raiz$,$x$,$d_{nn}$,$nn$) .

sugerencia, se considera que este algoritmo solamente es válido para espacios de vectores; sin embargo, el algoritmo no hace uso de las coordenadas, luego utilizando un algoritmo de agrupamiento válido para espacios métricos, el algoritmo de Fukunaga y Narendra es aplicable a este tipo de espacios, más generales que los de vectores. En concreto, en la implementación que se ha utilizado en esta tesis se ha usado el algoritmo de las $k$-medianas. Este algoritmo elige inicialmente los centros de cada agrupamiento de forma arbitraria, redistribuye los prototipos en cada agrupamiento según el centro más cercano a cada uno de ellos, y calcula el nuevo centro de cada agrupamiento como aquel prototipo que minimiza la suma de las distancias a todos los demás prototipos, para a continuación volver a redistribuir los prototipos. El proceso termina cuando en una iteración los agrupamientos no cambian.

En general, el árbol tiene aridad $l$, pero en esta tesis se ha utilizado un árbol binario para que los resultados fueran comparables con los de otros algoritmos similares, casi todos basados en árboles binarios (excepto el GNAT, como se explica más adelante); además, las hojas solamente contienen un único prototipo, como sucede también en otros algoritmos (excepto el k-d tree, que contiene 5 prototipos).

La fase de clasificación consiste en el recorrido, utilizando ramificación y poda, del árbol hasta que no queden ramas por explorar.

### Búsqueda (recursiva) en el algoritmo de Fukunaga y Narendra

En el trabajo original de Fukunaga y Narendra (1975) se explica el algoritmo utilizando una lista por cada nivel para realizar el recorrido del árbol; sin embargo, una formulación recursiva del algoritmo es mucho más natural y sencilla.[7] La figura 2.6 muestra una función recursiva que realiza la búsqueda en el árbol, siguiendo los siguientes pasos:

1. Dado un nodo del árbol $t$ (que no es una hoja), calcular la distancia de la muestra al representante de cada hijo de $t$, y actualizar el vecino más cercano.

2. De todos los hijos de $t$ que no se hayan eliminado ni visitado anteriormente, tomar aquel cuya distancia a la muestra sea la menor, $p$, y comprobar si se puede eliminar: si la distancia del representante de $p$ ($M_p$) a la muestra es mayor que la distancia al vecino más cercano hasta el momento más el radio de $p$ ($R_p$), seguro que en el súbarbol de $p$ no se encuentra el vecino más cercano y por tanto se puede podar (primera regla de eliminación, véase la figura 2.4). Más formalmente, un nodo $p$ se puede eliminar si se cumple:

$$d(x, M_p) > d(x, n) + R_p$$

[7] El hecho de que en el artículo de Fukunaga y Narendra (1975) aparezca la formulación iterativa se debe probablemente a que se utilizaba FORTRAN para la implementación, que no permitía por aquella época (1975) la recursividad.

3. Si $p$ es una hoja, eliminar aquellos prototipos $e$ contenidos en $p$ que no puedan estar más cerca de la muestra que el vecino más cercano actual (segunda regla de eliminación, véase la figura 2.5), es decir, aquellos cuya distancia al representante de $p$, $M_p$, sumada a la distancia al más cercano sea menor que la distancia de $M_p$ a la muestra. Formalmente, un prototipo $e$ de una hoja $p$ se puede eliminar si:

$$d(e, M_p) + d(x, n) < d(x, M_p)$$

   Para los prototipos de $p$ no eliminados se debe calcular su distancia a la muestra y actualizar el vecino más cercano. Si $p$ no es una hoja, buscar recursivamente en $p$.

4. Repetir los pasos 2 y 3 hasta que no queden hijos de $t$ que no hayan sido eliminados o visitados.

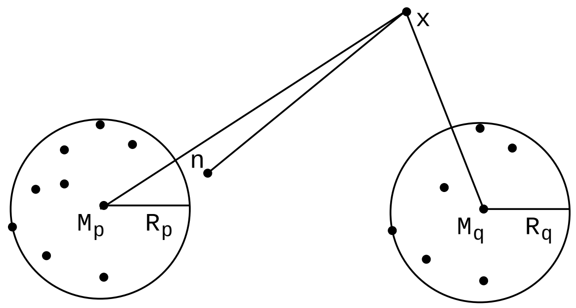


Figura 2.4: Primera regla de eliminación en el algoritmo de Fukunaga y Narendra: dado un nodo $p$ con representante $M_p$ y radio $R_p$, y siendo $n$ el actual candidato a vecino más cercano, si $d(x, M_p) > d(x, n) + R_p$, el nodo $p$ se puede eliminar puesto que ninguno de los prototipos contenidos en el subconjunto asociado $S_p$ puede ser el vecino más cercano. El nodo $q$ no se puede eliminar puesto que no cumple claramente la condición ($d(x, M_q)$ es menor que $d(x, n) + R_q$).

### 2.3.3. Vantage Point Tree (vp-tree)

El vp-tree (Yianilos, 1993) es un árbol ideado para la búsqueda del vecino más cercano en espacios métricos generales. Como otros algoritmos, construye un árbol binario en el que cada nodo representa un subconjunto $S$ de prototipos del conjunto de entrenamiento, y utiliza un elemento especial del conjunto llamado *pivote* (*vantage point*) para dividir el conjunto $S$ en dos subconjuntos, uno por cada hijo. En este epígrafe solamente se describe el llamado vp-tree optimizado, aunque existen otras variantes descritas en

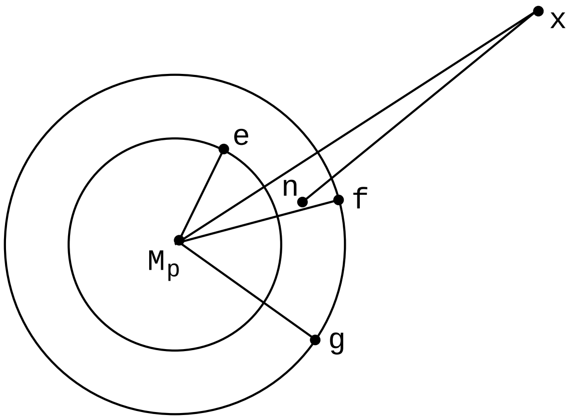


Figura 2.5: Segunda regla de eliminación en el algoritmo de Fukunaga y Narendra: dada una hoja $p$ que contiene varios prototipos (en el dibujo, $M_p$, $e$, $f$ y $g$), la regla funciona de la siguiente manera: si $d(x, M_p) > d(x, n) + d(M_p, e)$, el prototipo $e$ no puede estar más cerca de la muestra que el vecino más cercano actual $n$ y se puede eliminar; sin embargo, los prototipos $f$ y $g$ no cumplen la condición anterior y por tanto podrían estar más cerca de $x$ que $n$, por lo que no pueden ser eliminados. En este caso $f$ sería el siguiente candidato a vecino más cercano y $g$ se descartaría después de haber calculado su distancia a $x$.

**funcion** buscar [Fukunaga y Narendra recursivo]
    **entrada** $t$ (árbol)
        $x$ (muestra a clasificar)
    **entrada/salida** $d_{nn}$ (distancia al vecino más cercano)
        $nn \in P$ (vecino más cercano de $x$)
**comienzo**
    **para todo** $p = \text{Hijo}(t)$ **hacer**
        sea $M_p$ el representante de $p$, y $R_p$ el radio de $p$
$(a)$        $d_p := d(M_p, x)$
        **si** $d(x, M_p) < d_{nn}$ **entonces**
            $d_{nn} := d(x, M_p)$ ; $nn := p$
        **fsi**
    **fpara**
    **mientras** queden hijos de $t$ por visitar **hacer**
        $p := \min_{q=\text{HijoNoVisitado}(t)} d_q$
        visitado[$p$] := cierto
$(b)$        **si** $d_p \leq d_{nn} + R_p$ **entonces**        *// no es posible podarlo*
            **si** Hoja($p$) **entonces**
                **para todo** prototipo $x_i \in S_p$ **hacer**
$(b)$                    **si** $d_p <= d(x_i, M_p) + d_{nn}$ **entonces**
$(a)$                        $d_{x_i} := d(x, x_i)$
                        **si** $d_{x_i} < d_{nn}$ **entonces**
                            $d_{nn} := d_{x_i}$ ; $nn := x_i$
                        **fsi**
                    **fsi**
                **fpara**
            **si no**
                buscar($p$,$x$,$d_{nn}$,$x$)
            **fsi**
        **fsi**
    **fmientras**
**fin** buscar [Fukunaga y Narendra recursivo]

Figura 2.6: Algoritmo de búsqueda de Fukunaga y Narendra formulado como una función recursiva. Como en el kd-tree y otros algoritmos recursivos, inicialmente $d_{nn}$ debería ser $\infty$.

el artículo de Yianilos (1993) y posteriores mejoras utilizando más de un pivote (*multi vantage point tree, mvp-tree* (Bozjaya y Ozsoyoglu, 1997)), y eliminando elementos intermedios del árbol (*excluded middle vantage point forest, vpf* (Yianilos, 1999)).

### Construcción del árbol

Cada nodo del árbol contiene un pivote y dos hijos: el hijo izquierdo contiene el subárbol correspondiente al subconjunto de $S$ que está a una distancia del pivote menor que un cierto valor $mu$, y el hijo derecho contiene el subárbol correspondiente al resto de $S$ (aquellos prototipos que están a una distancia mayor o igual que $mu$ del pivote). Es importante que ambos hijos representen a subconjuntos de tamaño similar, para que el árbol resulte lo más balanceado posible y se eviten los árboles degenerados (aquellos que son más parecidos a una lista simple que a un árbol); de esta manera el proceso de búsqueda en el árbol podría en el mejor caso ser logarítmico. La figura 2.7 es un ejemplo de partición del espacio según un pivote.

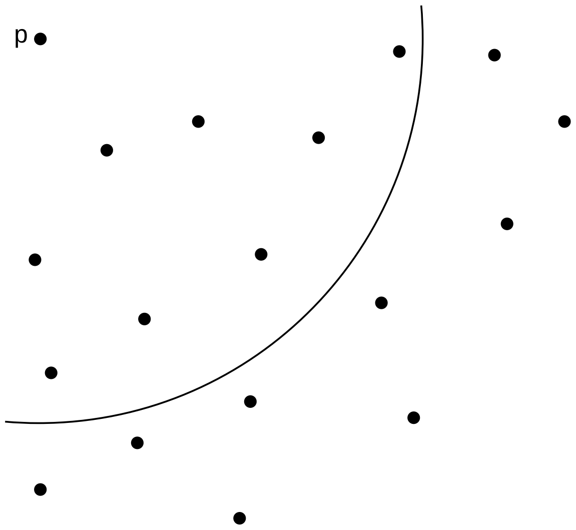


Figura 2.7: Ejemplo de partición de un conjunto en dos subconjuntos en función de su distancia a un pivote $p$, en la construcción de un vp-tree.

La construcción del árbol se realiza de forma recursiva, eligiendo un pivote para cada hijo y descomponiéndolo a su vez en dos hijos, hasta llegar a las hojas, que contendrán únicamente un prototipo. La figura 2.8 muestra el algoritmo para la construcción del árbol, en el que en cada nodo se almacenan, junto con el pivote y el valor de $mu$, la cota inferior y la superior de las distancias del pivote a los prototipos de cada subconjunto.

El pivote puede elegirse al azar, pero los resultados son mejores (Yiani-

**funcion** CrearArbol [vp-tree]
  **entrada** $S$ (conjunto de prototipos)
  **salida** $nodo$ (árbol que representa a $S$)
**comienzo**
  **si** $|S| = 1$ **entonces**
    $nodo$ := Hoja($S$)
  **sino**
    $nodo$ := NuevoNodo()
    $p$ := ElegirPivote($S$)
    $nodo.pivote := p$
    $nodo.mu :=$ Mediana$_{s \in S}$ $d(p, s)$
    $Iz := \{ s \in S - \{ p \} \mid d(p, s) < mu \}$
    $nodo.IzCotaInf := \min_{i \in Iz} d(p, i)$
    $nodo.IzCotaSup := \max_{i \in Iz} d(p, i)$
    $De := \{ s \in S - \{ p \} \mid d(p, s) \geq mu \}$
    $nodo.DeCotaInf := \min_{d \in De} d(p, d)$
    $nodo.DeCotaSup := \max_{d \in De} d(p, d)$
    $nodo.iz$ := CrearArbol($Iz$)
    $nodo.de$ := CrearArbol($De$)
  **fsi**
  **devolver** $nodo$
**fin** CrearArbol [vp-tree]

Figura 2.8: Construcción del árbol vp-tree.

los, 1993) si se elige de la siguiente manera (véase la figura 2.9): dado un conjunto $S$ de prototipos, se toma un subconjunto $P$ de $S$ de forma aleatoria; los puntos de $P$ serán candidatos a pivote. A continuación, para cada $p \in P$ se elige al azar un subconjunto $C \subset S$ y se calcula la mediana $mu$ de las distancias de $p$ a cada prototipo $c \in C$. Considerando $d(p,c) - mu$ una variable aleatoria, se calcula el momento de segundo orden centrado en la media (que coincide con la varianza), para estimar la amplitud del subconjunto $C$:

$$\begin{aligned}\text{Varianza}_{c\in C} \quad & (d(p,c)-\mu) \approx \\ & \tfrac{1}{|C|}\textstyle\sum_{c\in C}(d(p,c)-\mu)^2 - \left(\tfrac{1}{|C|}\textstyle\sum_{c\in C}(d(p,c)-\mu)\right)^2\end{aligned}$$

En la implementación del algoritmo utilizada en esta tesis (procedente del paquete Ranger (Murphy y Skiena, 1996)) se estima la amplitud como:

$$\begin{aligned}\text{Amplitud}_{c\in C} \quad & (d(p,c)-\mu) = \\ & \textstyle\sum_{c\in C}(d(p,c)-\mu)^2 - \left(\textstyle\sum_{c\in C}(d(p,c)-\mu)\right)^2\end{aligned}$$

Esta forma de calcular la amplitud no coincide exactamente con el momento de segundo orden (con la varianza), pero puesto que lo que se busca es elegir el prototipo $p$ cuya amplitud sea máxima como pivote, esta expresión de la amplitud sirve para ese propósito de forma similar a como lo haría la varianza.[8] Intuitivamente, el pivote es el elemento que mejor separa los prototipos en dos subconjuntos. En esta implementación del algoritmo se ha tomado $P$ como el conjunto $S$ y $C$ como $S - \{\ p\ \}$.

En el artículo de Yianilos (1993) se mencionan otras formas de construir el árbol de forma que cada nodo almacene además las distancias a sus antecesores en el árbol (padre, abuelo, etc). También se explica una construcción del árbol en la que los nodos contienen más de un prototipo, con el fin de ahorrar espacio.

### Búsqueda en el vp-tree

La búsqueda en el vp-tree se realiza siguiendo el esquema clásico de ramificación y poda, utilizando las cotas almacenadas en cada nodo para dirigir la búsqueda. La figura 2.10 muestra la función recursiva que implementa la búsqueda en el vp-tree.

[8] Hay un par de diferencias entre el valor de la amplitud y el de la varianza: un factor $\frac{1}{|C|}$ que podría sacarse como factor común y eliminarse puesto que sólo influye en el valor, no en qué prototipo $p$ tiene el máximo valor. Por otro lado, el otro factor $\frac{1}{|C|}$, que quedaría multiplicando únicamente al segundo término, tampoco influye en cual sería el prototipo de mayor valor: se puede demostrar fácilmente que el prototipo con la máxima amplitud es también el prototipo con la máxima varianza, y viceversa.

**funcion** ElegirPivote [vp-tree]
    **entrada** $S$ (conjunto de prototipos)
    **salida** $p \in S$ (pivote)
**comienzo**
    $P :=$ Muestra aleatoria de $S$
    $maximaAmplitud := 0$
    **para todo** $p \in P$ **hacer**
        $C :=$ Muestra aleatoria de $S - \{ p \}$
        $mu := \text{Mediana}_{c \in C}\ d(p, c)$
        $amplitud := \text{Amplitud}_{c \in C}\ (d(p, c) - mu)$
        **si** $amplitud > maximaAmplitud$ **entonces**
            $maximaAmplitud := amplitud$
            $pivote := p$
        **fsi**
    **fpara**
    **devolver** *pivote*
**fin** ElegirPivote [vp-tree]

Figura 2.9: Elección del pivote en el vp-tree.

### 2.3.4. Geometric Near-neighbor Access Tree (GNAT)

El GNAT (Brin, 1995) es un árbol que intenta estar balanceado y a la vez reflejar la estructura geométrica del conjunto de prototipos. Esta estructura de datos y el algoritmo de búsqueda asociado son especialmente adecuados para espacios de dimensionalidad alta o para espacios en los que el tiempo de cálculo de una distancia es un factor importante en el proceso de búsqueda. En la mayoría de los algoritmos basados en árboles anteriores a éste, el árbol siempre es binario (excepto en el de Fukunaga y Narendra); sin embargo, en este algoritmo el número de hijos de cada nodo es variable (entre un máximo y un mínimo), para de esta forma reflejar la geometría del subconjunto de prototipos que representa el nodo y mantener balanceado el árbol. En este árbol, las hojas contienen únicamente un prototipo.

#### Construcción del árbol GNAT

La construcción de este árbol es relativamente sencilla, y se explica de la siguiente manera:

1. Elegir $k$ *puntos de partición* (*split points* en inglés) $p_1, p_2, \ldots, p_k$ del conjunto de datos. El valor $k$ se denomina *grado* y puede variar en cada nodo del árbol. Los puntos de partición se pueden elegir al azar, pero los mejores resultados se obtienen si se eligen de forma que estén máximamente separados. Para ello, se elige el primer punto al azar y el segundo como aquel que esté más lejos del primero; el tercero

**funcion** buscar [vp-tree]
    **entrada** $n$ (nodo del árbol)
        $x$ (muestra a clasificar)
    **entrada/salida** $d_{nn}$ (distancia al vecino más cercano)
        $nn \in P$ (vecino más cercano de $x$)
**comienzo**
$(a)$ $d := d(x, n.pivote)$
    **si** $d < d_{nn}$ **entonces**
        $d_{nn} := d$ ; $nn := n.pivote$
    **fsi**
    **si** NoEsHoja($n$) **entonces**
        $medio := (n.IzCotaSup + n.DeCotaInf)/2$
        **si** $d < medio$ **entonces**
$(b)$             **si** $n.IzCotaInf - d_{nn} < d < n.IzCotaSup + d_{nn}$ **entonces**
                buscar($n.iz$,$x$,$d_{nn}$,$nn$)
            **fsi**
$(b)$             **si** $n.DeCotaInf - d_{nn} < d < n.DeCotaSup + d_{nn}$ **entonces**
                buscar($n.de$,$x$,$d_{nn}$,$nn$)
            **fsi**
        **si no entonces**
$(b)$             **si** $n.DeCotaInf - d_{nn} < d < n.DeCotaSup + d_{nn}$ **entonces**
                buscar($n.de$,$x$,$d_{nn}$,$nn$)
            **fsi**
$(b)$             **si** $n.IzCotaInf - d_{nn} < d < n.IzCotaSup + d_{nn}$ **entonces**
                buscar($n.iz$,$x$,$d_{nn}$,$nn$)
            **fsi**
        **fsi**
    **fsi**
**fin** buscar [vp-tree]

Figura 2.10: Búsqueda en el vp-tree. Inicialmente, $d_{nn}$ debería ser $\infty$.

será aquel cuya distancia al más cercano de los anteriores sea la mayor de todos los puntos restantes, y así sucesivamente hasta elegir los $k$ puntos de partición.

2. Dividir el conjunto inicial en subconjuntos $D_{p_i}$, cada uno asociado a un punto de partición $p_i$. Cada prototipo $p$ estará en el subconjunto asociado al punto de partición más cercano a él.

3. Para cada par de puntos de partición $(p_i, p_j)$, se calcula y almacena el rango de $p_i$ al conjunto $D_{p_j}$, $rango(p_i, D_{p_j})$, que es un par de valores: el mínimo y el máximo de $d(p_i, p)$ para $p \in D_{p_j} \cup \{ p_j \}$, denotados como $min\_d(p_i, D_{p_j})$ y $max\_d(p_i, D_{p_j})$, respectivamente.

4. Construir recursivamente el árbol para cada $D_{p_i}$, utilizando el mismo o diferente grado.

La elección del grado de cada nodo es un aspecto fundamental en la eficiencia del algoritmo de búsqueda. Inicialmente, la raíz del árbol tiene grado $k$ , y cada hijo del nodo tendrá un grado proporcional al número de prototipos que contiene, de forma que el grado medio de todos los hijos es $k$. En la implementación realizada en esta tesis se ha fijado el grado inicial a 10, el grado mínimo a 2 y el grado máximo a $5k$ o 200, la cantidad que sea menor; estos valores son los utilizados en los experimentos del artículo de Brin (1995).

La construcción del árbol se realiza en la fase de preproceso, como sucede con otros algoritmos o estructuras de datos. En el caso del GNAT, esta fase es especialmente costosa en tiempo y es su principal inconveniente, tal y como reconoce su autor en su artículo.

### Búsqueda en el árbol GNAT

Aunque en el artículo original solamente se menciona la búsqueda por rango, la búsqueda del vecino más cercano es simplemente una búsqueda por rango en la que el rango es siempre la distancia al vecino más cercano hasta el momento (inicialmente es infinito), y por tanto va cambiando según cambia el candidato a vecino más cercano. Esta adaptación de la búsqueda por rango para la búsqueda del vecino más cercano es habitual en este tipo de algoritmos.

El algoritmo de búsqueda se formula como una función recursiva (véase la figura 2.12), y sigue los siguientes pasos.

1. Sea $P$ el conjunto de puntos de partición de un nodo (inicialmente, de la raíz del árbol). Cada elemento de $P$ se corresponde con un hijo del nodo en el árbol, y representa un conjunto de puntos.

2. Elegir un prototipo $p$ de $P$ (que no haya sido elegido anteriormente) y calcular su distancia a la muestra $x$, $d(x,p)$. Actualizar si es necesario el vecino más cercano hasta el momento y el rango de la búsqueda, $r$.

3. Para todo $q \in P, q \neq p$, si la intersección del rango $[d(x,p)-r, d(x,p)+r]$ con $rango(p, D_q)$ es vacía, es decir, si $d(x,p)+r < min\_d(p, D_q)$ o $d(x,p)-r > max\_d(p, D_q)$, eliminar $q$ de P(véase la figura 2.11). Esta eliminación se basa en la desigualdad triangular, y se explica de la siguiente forma. Sea un prototipo $y \in D_q$,

    a) Si $d(y,p) < d(x,p) - r$, como $d(x,y) + d(y,p) \geq d(x,p)$ (desigualdad triangular), entonces $d(x,y) > r$, luego $y$ no puede ser el vecino más cercano.

    b) Si $d(y,p) > d(x,p)+r$, como $d(y,x)+d(x,p) \geq d(y,p)$ (desigualdad triangular), entonces se concluye también que $d(x,y) > r$ y por tanto $y$ no puede ser el vecino más cercano.

4. Repetir los pasos 2 y 3 hasta que todos los prototipos de $P$ se hayan elegido o eliminado. Para los $p_i \in P$ no eliminados, repetir la búsqueda recursivamente en $D_{p_i}$ si el nodo de $p_i$ no es una hoja.

Quizá por el hecho de estar pensado para la búsqueda por rango se pueden encontrar algunos aspectos mejorables en el algoritmo de búsqueda, como por ejemplo que no tiene ningún orden especial para elegir los hijos, calcular su distancia a la muestra y seguir la búsqueda recursiva; en otros algoritmos como el de Fukunaga y Narendra, por ejemplo, se calculan las distancias de los representantes de los hijos a la muestra y la búsqueda continua por aquellos hijos más cercanos, de manera que se encuentre el vecino más cercano lo antes posible, lo cual permite podar más el resto del árbol y por tanto hacer que la búsqueda sea más rápida. Cuando se busca por rango no es importante el orden en que se recorre el árbol, ya que se deben encontrar todos aquellos prototipos que se encuentren como mucho a una distancia fija o rango $r$ de la muestra, y esa distancia es la que determina la rapidez de la búsqueda: si es relativamente grande, la búsqueda será lenta; si es pequeña, la búsqueda será más rápida. En la búsqueda del vecino más cercano el rango va cambiando (siempre es la distancia al vecino más cercano hasta el momento) y es un factor determinante en la eficacia de la búsqueda: cuanto menor sea, más nodos se podarán y más rápida será la búsqueda. Por tanto, cuanto antes se encuentre el vecino más cercano (que marca el valor mínimo para el rango), más rápida será la búsqueda.

## 2.4. La familia AESA

Los algoritmos de la familia AESA (Vidal, 1986, 1994; Micó et al., 1994, 1996) se caracterizan principalmente por calcular un número muy reducido

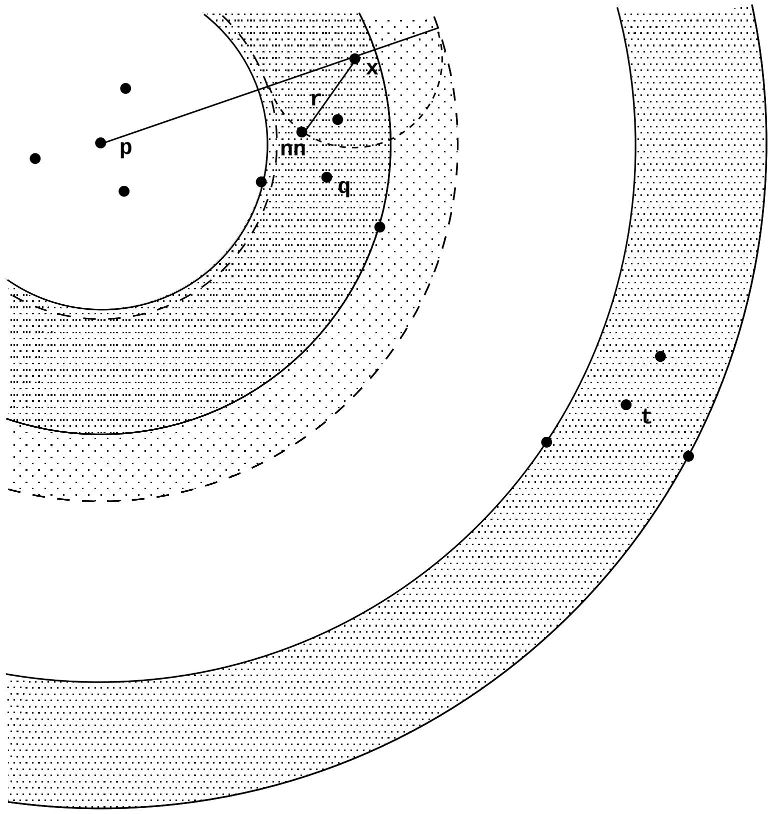


Figura 2.11: Ejemplo de eliminación en el algoritmo GNAT. Dado un prototipo $p$, y suponiendo que la distancia al vecino más cercano a la muestra es $r$, se puede observar que la intersección del rango $[d(x,p) - r, d(x,p) + r]$ (líneas punteadas en el dibujo) con $rango(p, D_q)$ (líneas continuas) es no vacía, luego no podría eliminarse de forma segura el nodo de $q$ (de hecho, uno de los prototipos de $D_q$ está más cerca que el actual vecino más cercano). Sin embargo, la intersección con $rango(p, D_t)$ sí resulta vacía, por lo que el nodo asociado a $t$ puede eliminarse porque seguro que no contiene al vecino más cercano.

**funcion** buscar [GNAT]
 **entrada**   $n$ (nodo, contiene puntos de partición)
   $x$ (muestra a clasificar)
 **entrada/salida**   $r$ (rango de búsqueda y distancia al vecino más cercano)
   $nn \in P$ (vecino más cercano de $x$)
**comienzo**
 Sea $T$ el conjunto de puntos de partición del nodo $n$
 **para todo** punto de partición $p \in T$ (no eliminado) **hacer**
$(a)$    $d := d(x, p)$
  **si** $d < r$ **entonces**
   $r := d$ ; $nn := p$
  **fsi**
  **para todo** $q \in T$ no eliminado, $q \neq p$ **hacer**
$(b)$     **si** $d + r < min\_d(p, D_q)$ **o** $d - r > max\_d(p, D_q)$ **entonces**
    $T := T - \{ q \}$
   **fsi**
  **fpara**
 **para todo** $p \in T$ no eliminado **hacer**
  $h :=$ Hijo($n$,$p$)    *// hijo de $n$ correspondiente a $D_p$*
  **si** NoEsHoja($h$) **entonces**
   buscar($h$,$x$,$r$,$nn$)
  **fsi**
 **fpara**
**fin** buscar [GNAT]

Figura 2.12: Función de búsqueda del vecino más cercano en un árbol GNAT. Inicialmente, $r$ debería ser $\infty$.

de distancias que no depende del tamaño del conjunto de entrenamiento; esta característica, demostrada experimentalmente, los hace especialmente interesantes cuando el cálculo de la distancia entre dos elementos es muy costoso. Sin embargo, cuando se trabaja con distancias rápidas de calcular, el procesamiento adicional hace que estos algoritmos no sean los más rápidos.

Estos algoritmos pueden trabajar en espacios métricos generales y están basados en un esquema de aproximación y eliminación: para cada prototipo del conjunto de entrenamiento, se obtiene una cota inferior de la distancia real entre dicho prototipo y la muestra a clasificar. Cuando dicha cota es mayor que la distancia del vecino más cercano hasta el momento, el prototipo ya no puede ser el vecino más cercano y por tanto se puede eliminar (eliminación). El siguiente candidato a vecino más cercano (aproximación) será aquel cuya cota sea la menor de entre aquellos prototipos que queden después de la eliminación. La diferencia entre los tres algoritmos de búsqueda reside en la forma de calcular la cota inferior y en la forma de recorrer el conjunto de entrenamiento para eliminar prototipos y elegir el siguiente candidato a vecino más cercano.

### 2.4.1. Approximation and Elimination Search Algorithm (AESA)

El algoritmo AESA (véase la figura 2.15) tiene una fase de preproceso en la que se calculan todas las distancias entre todos los prototipos y se almacenan en una matriz $D$.

En la fase de clasificación, inicialmente se elige de forma arbitraria un candidato a vecino más cercano $s$ y se calcula su distancia a la muestra $d_s$ (paso $(a)$), eliminándolo del conjunto de entrenamiento y actualizando el vecino más cercano. A continuación esta distancia se usa para obtener una cota inferior $G$ de la distancia de cada prototipo $p$ a la muestra, utilizando para ello la propiedad de la desigualdad triangular (véase la figura 2.13). Si la cota resulta ser superior a la distancia al vecino más cercano hasta el momento, el prototipo se elimina (paso $(b)$).

En la figura 2.14 se puede observar un ejemplo: en este caso los prototipos $p_1$ y $p_2$ no se pueden eliminar porque sus cotas son inferiores a la distancia de $x$ a $s$; sin embargo, $p_3$ y $p_4$ pueden ser eliminados con toda seguridad, ya que nunca serán el vecino más cercano.

El siguiente candidato a vecino más cercano se elige utilizando las cotas, y se repite todo el proceso (actualizando la cota si es mayor que la obtenida hasta el momento) hasta que no quedan prototipos en el conjunto de entrenamiento. La desigualdad triangular asegura que nunca se elimina el vecino más cercano.

Este algoritmo es probablemente el algoritmo que menos distancias calcula para obtener el vecino más cercano, pero tiene un gran inconveniente: la matriz de distancias que se calcula en el preproceso puede llegar a tener

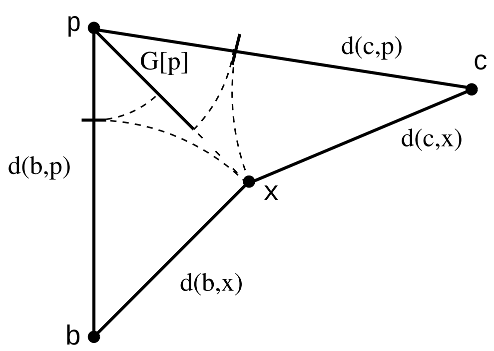


Figura 2.13: Dados dos prototipos $b$ y $c$ cuya distancia a la muestra $x$ ya se ha calculado, la cota inferior de la distancia de $p$ a $x$ (en el dibujo, el tramo continuo denotado como $G[p]$) se calcula como el máximo de $|d(x,c)-d(c,p)|$ y $|d(x,b) - d(b,p)|$.

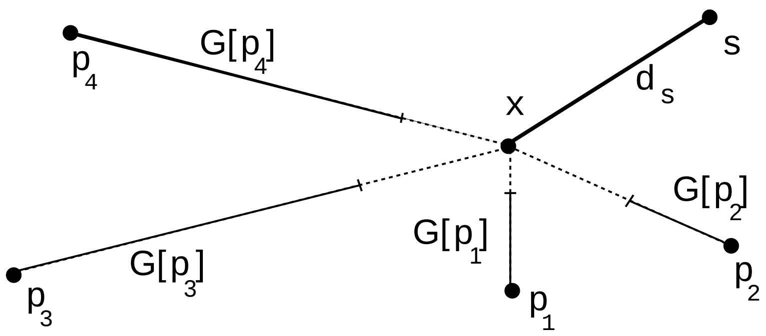


Figura 2.14: Eliminación de prototipos utilizando una cota inferior de la distancia a la muestra $x$; cada cota está dibujada como un segmento de trazo continuo en la línea que va de cada prototipo hacia $x$. En el dibujo, $p_3$ y $p_4$ se pueden eliminar puesto que su cota inferior ($G[p_3]$ y $G[p_4]$, respectivamente) es mayor que la distancia del vecino más cercano hasta el momento $s$ a $x$ ($d_s$); sin embargo, $p_1$ y $p_2$ no se pueden eliminar y podrían ser el vecino más cercano.

```
algoritmo AESA (Approximating and Eliminating Search Algorithm)
   entrada  P ⊂ E (conjunto de prototipos)
            x (muestra a clasificar)
   usa
            D ∈ ℝ^{|P|×|P|} (matriz de distancias)
   salida   nn ∈ P (vecino más cercano de x en P)
comienzo
    para todo p ∈ P hacer G[p] := 0 fpara
    nn := desconocido; d_nn := ∞
    s := ElementoArbitrario(P)
    mientras |P| > 0 hacer
(a)     d_s := d(x, s)
        P := P − {s}
        si d_s < d_nn entonces                          // actualización de nn
            nn := s
            d_nn := d_s
        fsi
        siguiente := desconocido; gmin := ∞
        para todo p ∈ P hacer
            G[p] := maximo(G[p],|D[p, s] − d_s|)        // actualización de cotas
(b)         si G[p] > d_nn entonces
                P := P − {p}                            // eliminación
            si no
                si G[p] < gmin entonces
                    gmin := G[p]
                    siguiente := p                      // aproximación
                fsi
            fsi
        fpara
        s := siguiente
    fmientras
fin AESA
```

Figura 2.15: Algoritmo AESA.

un tamaño excesivo si el conjunto de entrenamiento es lo suficientemente grande (la complejidad espacial es cuadrática con respecto al tamaño del conjunto de entrenamiento).

Por otro lado, cuando la distancia no es excesivamente costosa en tiempo, el hecho de recalcular las cotas en cada paso hace que este algoritmo no sea el más rápido. Por tanto, el AESA necesita para ser competitivo que el conjunto de entrenamiento tenga un tamaño razonable (para poder almacenar la matriz de distancias) y que el cálculo de la distancia entre dos elementos sea costoso en tiempo.

Como los algoritmos de la sección anterior, el AESA se ha extendido para obtener los $k$ vecinos más cercanos (Aibar et al., 1993), manteniendo todas sus características. En el artículo de Aibar et al. (1993) se explica una versión del $k$-AESA, el $k$-AESAU, que permite obtener los $k$ vecinos más cercanos de forma eficiente utilizando el AESA. Sin embargo, y por motivos de uniformidad, la extensión del AESA para obtener los $k$ vecinos más cercanos se ha implementado en esta tesis como en los demás algoritmos: en el paso $(a)$ se almacenan de forma ordenada los $k$ vecinos más cercanos que se hayan encontrado hasta el momento, y en el paso $(b)$ se compara la cota con la del último vecino almacenado en el paso $(a)$, que en las primeras iteraciones puede no ser el $k$-ésimo ya que no se habrán calculado todavía $k$ distancias.

### 2.4.2. Linear AESA (LAESA)

La principal motivación de este algoritmo (Micó et al., 1994; Micó, 1996) es superar el problema de la complejidad espacial cuadrática del AESA. Para ello, en el preproceso se elige a partir del conjunto de entrenamiento un conjunto $B$ de *prototipos base*, se calcula la distancia de cada prototipo base a todos los demás prototipos del conjunto de entrenamiento y se almacena dicha distancia en una matriz que ya no es cuadrada, como en el AESA. De esta forma, la complejidad espacial del LAESA es lineal con respecto al tamaño del conjunto de entrenamiento.

Durante la clasificación, se seleccionan como candidatos a vecino más cercano los prototipos base, y se calcula su distancia a la muestra, utilizando dicha distancia y las distancias calculadas en el preproceso para obtener una cota inferior de la distancia de cada prototipo a la muestra, de forma similar a como se hacía en el AESA (utilizando la desigualdad triangular). Cuando se han seleccionado (y eliminado) todos los prototipos base, las cotas ya no se actualizan y comienza la búsqueda del vecino más cercano con un procedimiento parecido al del AESA (con la diferencia de que no se actualizan las cotas).

La figura 2.16 presenta una versión simplificada del LAESA; tal y como aparece en (Micó et al., 1994), el LAESA se puede considerar como una pequeña familia de algoritmos en función de cómo se realiza la fase de aproximación y eliminación. De todas las versiones del LAESA, la más eficiente en número de distancias calculadas es la que aparece en la figura 2.16, y es la que se ha tomado como base para las modificaciones que se han realizado en esta tesis (véase el capítulo 3).

El número de distancias que calcula el LAESA es siempre mayor que el que calcula el AESA (para que fueran iguales el conjunto de prototipos base tendría que ser todo el conjunto de entrenamiento), pero tiene un comportamiento similar en ambos algoritmos: crece muy lentamente con respecto al tamaño del conjunto de entrenamiento, y parece estar acotado por una constante; por tanto, se puede conjeturar que no depende del tamaño del conjunto de entrenamiento. Esta conclusión no ha podido ser demostrada teóricamente y se basa en observaciones empíricas.

Un factor del que depende el número de distancias calculadas por el LAESA es el tamaño del conjunto de prototipos base $B$ y el algoritmo para la selección de dichos prototipos. En el trabajo de Micó (1996) se hace un estudio exhaustivo de estas cuestiones y se llega a la conclusión de que el número óptimo de prototipos base depende de la dimensionalidad de los datos, y se deben seleccionar de forma que estén separados lo máximo posible.

El LAESA se puede extender de forma muy sencilla para encontrar los $k$ vecinos más cercanos (Moreno-Seco et al., 2002), almacenando en el paso $(a)$ los $k$ vecinos más cercanos que se han encontrado hasta el momento, y tomando en el paso $(b)$ la distancia al último de ellos en lugar de la distancia al más cercano.

### 2.4.3. Tree LAESA (TLAESA)

La mayoría de los algoritmos rápidos (véanse por ejemplo los que aparecen en la sección 2.3) para la búsqueda del vecino más cercano se basan en la construcción de un árbol durante el preproceso, y en el recorrido del árbol durante la clasificación siguiendo un esquema de ramificación y poda. La principal motivación de esta estrategia es obtener una complejidad temporal sublineal con respecto al tamaño del conjunto de entrenamiento. El TLAESA (Micó et al., 1996; Micó, 1996) es uno de estos algoritmos, y está basado (como su nombre indica) en el LAESA, lo cual le permite obtener un reducido número de distancias calculadas (aunque superior a los de LAESA y AESA) con una complejidad temporal sublineal.

Durante la fase de preproceso, primero se seleccionan los prototipos base y se calculan las distancias a todos los demás prototipos (incluidos los demás prototipos base), como en el LAESA. A continuación se construye un árbol binario de la siguiente manera: se elige un prototipo arbitrario $\rho$ como representante de la raíz del árbol, y se construyen dos subárboles recursivamente a partir de dos particiones del conjunto de prototipos. El hijo de la izquierda tendrá como representante $(t_i)$ el prototipo cuya distancia al representante del padre sea la mayor, mientras que el representante del hijo

```
algoritmo LAESA (Linear Approximating and Eliminating Search Algorithm)
    entrada  P ⊂ E (conjunto de prototipos)
             x (muestra a clasificar)
    usa      B ⊂ P (conjunto de prototipos base)
             D_B ∈ ℝ^{|B|×|P|} (distancias a prot. base)
    salida   nn ∈ P (vecino más cercano de x en P)
comienzo
     para todo p ∈ P hacer G[p] := 0 fpara
     nn := desconocido; d_nn := ∞
     s := ElementoArbitrario(B)            // el primer prototipo siempre es de B
     mientras |P| > 0 hacer
(a)       d_s := d(x, s)
          P := P − {s}
          si d_s < d_nn entonces   nn := s ; d_nn := d_s   fsi // actualización de nn
          siguiente_B := desconocido; gmin_B := ∞
          siguiente := desconocido; gmin := ∞
          para todo p ∈ P hacer
               si s ∈ B entonces
                    G[p] := maximo(G[p],|D_B[s, p] − d_s|)   // actualización de cotas
               fsi
(b)            si G[p] > d_nn entonces
                    P := P − {p}                               // eliminación
               si no
                    si p ∈ B entonces
                         si G[p] < gmin_B entonces
                              gmin_B := G[p] ; siguiente_B := p // aproximación en B
                         fsi
                    si no
                         si G[p] < gmin entonces
                              gmin := G[p] ; siguiente := p // aproximación en P − B
                         fsi
                    fsi
               fsi
          fpara
          si siguiente_B ≠ desconocido entonces
               s := siguiente_B                       // selección en B, si es posible
          si no
               s := siguiente                                   // selección en P − B
          fsi
     fmientras
fin LAESA
```

Figura 2.16: Algoritmo LAESA (versión simplificada).

**algoritmo** TLAESA (Tree LAESA)
  **entrada** $T$ (árbol con raiz $\rho$)
    $x$ (muestra a clasificar)
  **usa** $B \subset P$ (conjunto de prototipos base)
    $D_B \in \mathbb{R}^{|B| \times |P|}$ (matriz de distancias)
  **salida** $nn \in P$ (vecino más cercano de $x$ en $P$)
**comienzo**
  $nn :=$ desconocido; $d_{nn} := \infty$
  **para todo** $b \in B$ **hacer**
    $D[b] := d(x, b)$
    **para todo** $p \in P$ **hacer**
      $G[p] :=$ maximo($G[p]$,$|D_B[b,p] - D[b]|$) *// actualización de cotas*
    **fpara**
  **fpara**
  buscarTLAESA($x$,$\rho$,$G[\rho]$,$nn$,$d_{nn}$)
**fin** TLAESA

Figura 2.17: Algoritmo TLAESA.

de la derecha ($t_d$) será el mismo que el del padre. El conjunto de prototipos del hijo de la derecha es el formado por los prototipos que están más cerca de $t_d$ que de $t_i$, y el conjunto de prototipos del hijo de la izquierda será el resto del conjunto del padre. Este procedimiento termina cuando se llega a un conjunto formado por un único prototipo, que será una hoja en el árbol. Para cada nodo del árbol conteniendo un conjunto $S$ de prototipos y con un representante $m$, se calcula su radio $r$ como $max_{p \in S} d(p, m)$.

Las figuras 2.17 y 2.18 muestran el procedimiento de búsqueda del algoritmo TLAESA. El principio de la fase de clasificación es similar al del LAESA: se calculan las distancias de la muestra a los prototipos base y se utilizan para obtener una cota inferior de la distancia. A continuación, se busca recursivamente en el árbol utilizando las cotas y los radios asociados a cada nodo, eliminando aquellos subárboles que no puedan contener al vecino más cercano. Cuando se llega a una hoja, si su cota es inferior a la distancia al más cercano hasta el momento, se calcula la distancia a la muestra y se actualiza el vecino más cercano.

Al contrario que otros algoritmos basados en árboles, las distancias solamente se calculan en las hojas; esto hace que la poda del árbol no comience hasta que se haya llegado a la primera hoja, pero también permite obtener un número reducido de distancias calculadas: si se calculase la distancia al representante de cada nodo (como hacen otros algoritmos) se podría empezar antes a podar, pero el número de distancias sería probablemente superior.

Como en los algoritmos anteriores, la extensión del TLAESA para obtener los $k$ vecinos más cercanos se puede hacer almacenando los $k$ vecinos

```
algoritmo buscarTLAESA
    entrada          x (muestra a clasificar)
                     t ∈ T (nodo del árbol)
                     g_t (cota de t)
    entrada/salida   nn ∈ P (vecino más cercano de x en P)
                     d_nn (distancia al vecino más cercano)
comienzo
     si EsHoja(t) entonces
         si g_t < d_nn entonces
(a)          d_t := d(x,t)
             si d_t < d_nn entonces                      // actualización de nn
                 d_nn := d_t
                 nn := t
             fsi
         fsi
     si no
         t_i := HijoIzquierda(t) ; t_d := HijoDerecha(t)
         g_i := G[t_i] ; gd := G[t_d]
         r_i := Radio(t_i) ; r_d := Radio(t_d)
         si g_i ≤ g_t entonces                           // búsqueda en izq-der.
(b)          si d_nn + r_i > g_i entonces
                 buscar(x,t_i,g_i,nn,d_nn)
             fsi
(b)          si d_nn + r_d > g_d entonces
                 buscar(x,t_d,g_d,nn,d_nn)
             fsi
         si no                                           // búsqueda en der-izq.
(b)          si d_nn + r_d > g_d entonces
                 buscar(x,t_d,g_d,nn,d_nn)
             fsi
(b)          si d_nn + r_i > g_i entonces
                 buscar(x,t_i,g_i,nn,d_nn)
             fsi
         fsi
     fsi
fin buscarTLAESA
```

Figura 2.18: Búsqueda en el árbol del vecino más cercano en el TLAESA.

más cercanos hasta el momento en el paso $(a)$, y utilizando la distancia al último de ellos en lugar de la distancia al más cercano en los pasos etiquetados con $(b)$. En la implementación debe tenerse en cuenta que en cada llamada recursiva al procedimiento buscar puede cambiar tanto el vecino más cercano como el último de los $k$ vecinos, luego antes de los pasos $(b)$ de la figura 2.18 debe actualizarse el valor que se utiliza en la condición ($d_{nn}$ o la distancia al último de los $k$ vecinos).

## 2.5. Búsqueda rápida de los $k$ vecinos más cercanos

En general, la clasificación utilizando los $k$ vecinos más cercanos produce tasas de error más bajas que utilizando únicamente el vecino más cercano. La búsqueda de los $k$ vecinos más cercanos de forma exhaustiva (calculando todas las distancias de la muestra a los prototipos del conjunto de entrenamiento) es costosa, pero no mucho más que la búsqueda exhaustiva del vecino más cercano, en el caso general en el que el valor de $k$ es mucho menor que el tamaño del conjunto de entrenamiento. El número de distancias es exactamente el mismo, y únicamente existe un pequeño coste adicional de almacenar en cada paso los $k$ vecinos más cercanos.

Una forma eficiente de encontrar los $k$ vecinos más cercanos es extender alguno de los algoritmos rápidos de búsqueda como se comenta a lo largo de este capítulo, realizando dos cambios:

1. Cada vez que se calcula una distancia (en el paso marcado con $(a)$ en los algoritmos), se almacena (si es posible) en un vector o lista que contiene los $k$ vecinos más cercanos hasta el momento. Esta modificación no es un factor importante en el tiempo final de clasificación (siempre que $k$ sea mucho menor que el tamaño del conjunto de entrenamiento, por supuesto). Solamente se requiere la inserción de un elemento en un vector ordenado, y esto requiere un tiempo proporcional al valor de $k$.

2. Cuando en las condiciones para la eliminación o poda de prototipos de entrenamiento se utiliza la distancia al vecino más cercano hasta el momento, se debe utilizar en su lugar la distancia al último de los (como mucho) $k$ vecinos más cercanos encontrados hasta el momento (paso $(b)$). Esta distancia es mayor que la del vecino más cercano y por tanto se puede suponer que el algoritmo eliminará menos prototipos, lo cual le llevará a calcular más distancias y por tanto a consumir más tiempo en la clasificación. Además, cuanto mayor sea el valor de $k$ parece razonable pensar que mayor será el tiempo de búsqueda, hasta alcanzar el tiempo de la búsqueda exhaustiva.

Para comprobar que se cumple la hipótesis de que el número de distancias y el tiempo de clasificación de los $k$ vecinos más cercanos aumentan en estos algoritmos con respecto a los del vecino más cercanos se ha realizado un experimento preliminar, con la base de datos CROMOSOMAS (que se describe con más detalle en el apéndice A). Se trata de dos conjuntos de 2200 cadenas de caracteres que codifican características de cromosomas humanos, en los que se utiliza la distancia de edición de cadenas o distancia de Levenshtein para calcular la distancia entre dos prototipos. Las gráficas muestran los resultados medios de dos experimentos en los que se ha utilizado uno de los conjuntos para entrenamiento y el otro para test.

En las figuras 2.19 y 2.20 se muestra para cada algoritmo (excepto para el k-d tree, que requiere un espacio de vectores) el número medio de distancias calculadas en función del valor de $k$ (en la gráfica de la izquierda), y el tiempo medio de clasificación (en la de la derecha). Como se puede observar, en todos los casos tanto el número de distancias como el tiempo de clasificación aumentan cuando se incrementa el valor de $k$. La principal aportación de esta tesis consiste en una regla de clasificación que obtiene tasas de error similares a las de la regla de los $k$ vecinos más cercanos, pero que no sufre un incremento en el número de distancias ni en el tiempo de clasificación (al menos de forma apreciable) con respecto a la regla del vecino más cercano.

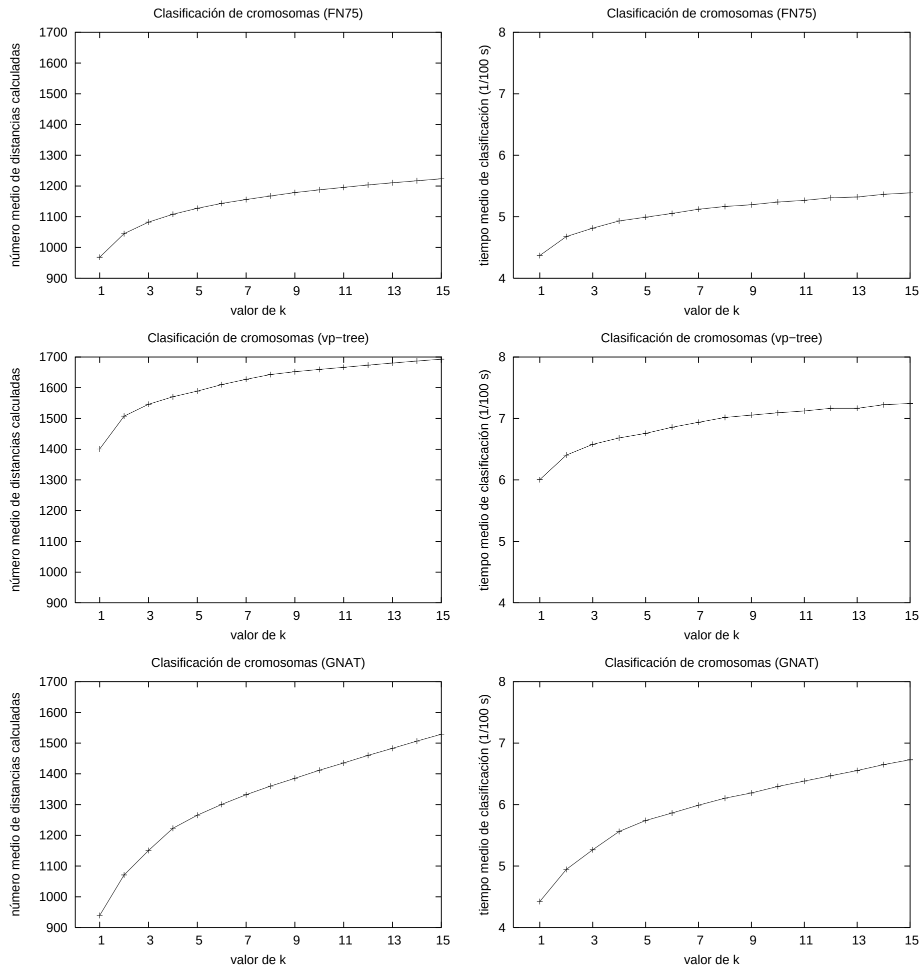


Figura 2.19: Coste adicional de la búsqueda de los $k$-NN, con diferentes algoritmos.

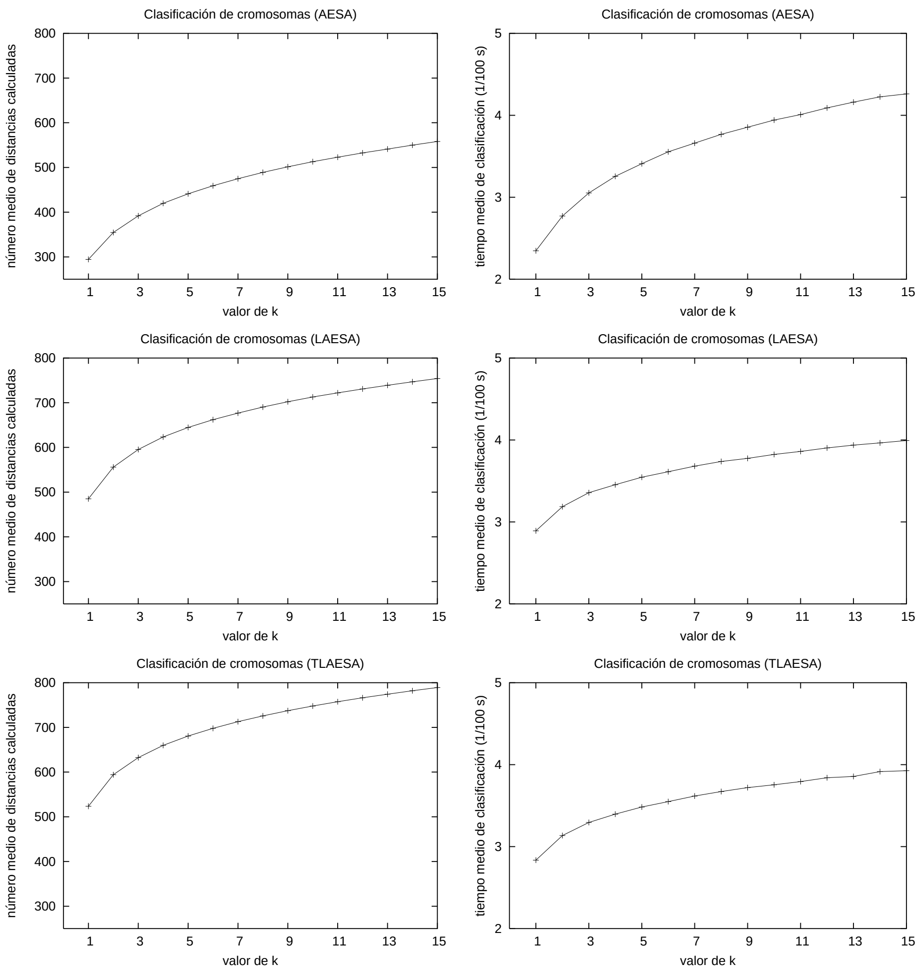


Figura 2.20: Coste adicional de la búsqueda de los $k$-NN, con diferentes algoritmos (2).

# Parte II

# Aportaciones

# Capítulo 3

# Mejoras del LAESA para clasificación utilizando $k$ vecinos cercanos

El algoritmo LAESA (véase el epígrafe 2.4.2) encuentra el vecino más cercano a la muestra calculando un número muy reducido de distancias, lo cual lo hace especialmente adecuado para tareas de clasificación en las que el cálculo de la distancia es costoso. En tareas de clasificación, el objetivo de encontrar el vecino más cercano es clasificar la muestra como perteneciente a la misma clase que su vecino más cercano. Por tanto, si el objetivo final es clasificar la muestra en una clase, puede resultar interesante buscar alternativas a la regla de clasificación del vecino más cercano. En este sentido han aparecido varios trabajos (Arya et al., 1998; Kleinberg, 1997; Indyk y Motwani, 1998; Clarkson, 1999) que clasifican la muestra de forma aproximada, utilizando un vecino cercano que no es necesariamente el más cercano.

En este capítulo se presentan varias versiones de una mejora del LAESA para la clasificación aproximada, denominada A$k$-LAESA (del inglés *Approximate $k$-LAESA*), que clasifica la muestra utilizando varios vecinos cercanos, incluyendo o no el vecino más cercano (depende de la versión). La elección del LAESA se debe a que es un algoritmo desarrollado por los directores de esta tesis junto con Enrique Vidal (Micó et al., 1994; Micó, 1996), que han estudiado detenidamente su comportamiento y sus posibilidades. Este algoritmo calcula muy pocas distancias, tiene un coste espacial y temporal lineal con el tamaño del conjunto de entrenamiento, y está basado en un esquema de aproximación y eliminación. Con posterioridad, la técnica empleada en la última versión del A$k$-LAESA, que es la que mejores resultados obtiene, se ha extendido con éxito a otros algoritmos basados en el esquema de aproximación y eliminación (véase el capítulo 4).

Los epígrafes sucesivos se dedican a cada una de las versiones del A$k$-LAESA, por orden de desarrollo. En cada epígrafe, después de describir las motivaciones y el funcionamiento del algoritmo se presentan unos resultados preliminares (excepto en el caso de la versión 3), en los que el algoritmo se compara con el LAESA y con la regla de los $k$ vecinos más cercanos. Para facilitar la comparación de las distintas versiones entre sí, se incluye otro apartado de experimentos y resultados utilizando las mismas bases de datos para todas las versiones del A$k$-LAESA.

## 3.1. A$k$-LAESA (versión 1)

Como se explica en el epígrafe 2.4.2, la fase de clasificación en el algoritmo LAESA tiene dos partes: en la primera de ellas, utilizando las distancias de la muestra a los prototipos base, se obtiene una cota inferior de la distancia de cada prototipo a la muestra, a la vez que se intenta eliminar algún prototipo del conjunto de entrenamiento. En la segunda parte, en cada paso se selecciona un candidato a vecino más cercano $s$ que ya no es un prototipo base, se elimina del conjunto de prototipos (para no volver a seleccionarlo), se calcula su distancia a la muestra y se actualiza el vecino más cercano hasta el momento $nn$, si es necesario; la distancia de $nn$ a la muestra, $d_{nn}$, se utiliza para eliminar aquellos prototipos cuya cota inferior sea mayor y por tanto no puedan ser el vecino más cercano; además, se selecciona el siguiente candidato a vecino más cercano como aquel cuya cota inferior sea la menor de entre los prototipos restantes.

La primera de las modificaciones al algoritmo LAESA para la clasificación aproximada (Moreno-Seco et al., 1999) se realizó partiendo de la suposición de que, si las cotas se aproximan correctamente a la distancia real, en los últimos pasos del algoritmo los prototipos que aún no se han eliminado son prototipos que están muy cerca de la muestra y por tanto podrían servir para clasificar la muestra de forma parecida al vecino más cercano. Esta primera modificación, denominada A$k$-LAESA (versión 1), consiste básicamente en terminar la fase de búsqueda cuando quedan $k$ o menos prototipos sin eliminar en el conjunto de entrenamiento, siempre y cuando se hayan eliminado todos los prototipos base.[1]

La figura 3.1 es básicamente la figura 2.16 con las modificaciones mínimas (marcadas en el margen izquierdo) para obtener el A$k$-LAESA. Como se puede observar, cuando el conjunto $P$ se queda vacío se recupera el vecino más cercano hasta el momento para clasificar la muestra. En otro caso, se votaría con los vecinos ($k$ o menos) que no se hubieran eliminado de $P$; puesto que los candidatos a vecino más cercano se eliminan inmediatamente después de calcular su distancia a la muestra, el vecino más cercano hasta el momento no se utiliza en esta votación. El procedimiento de votación

[1] De no poner esta restricción se podría en algún caso utilizar para la clasificación algunos de los prototipos base, y dado que se eligen de forma que estén máximamente separados, las tasas de acierto en la clasificación obtenidas serían probablemente peores.

es el habitual cuando se tiene más de un prototipo: se toma la clase con mayor número de votos, y en caso de empate se toma la primera clase que se encuentre con el máximo número de votos (otra posibilidad sería tomar la clase del vecino más cercano).

Por otro lado, para mantener el mismo comportamiento que el LAESA cuando $k = 1$, se tiene este caso en cuenta y se utiliza 0 en lugar de 1 como tamaño límite para finalizar la búsqueda, lo cual hace que la condición del bucle sea exactamente la misma que en el LAESA. Si no se tuviera en cuenta este caso, para $k = 1$ el A$k$-LAESA tomaría la mayoría de las veces el vecino más cercano (porque $|P| = 0$), pero en otros casos tomaría un vecino cercano, ignorando el vecino más cercano (en esta primera versión), lo cual produce peores tasas de acierto. Por tanto, por un lado se consigue la propiedad de que cuando $k = 1$, el A$k$-LAESA y el LAESA son exactamente iguales, y por otro lado se evita un caso que produce malos resultados.

### 3.1.1. Resultados preliminares

En (Moreno-Seco et al., 1999) se presentan los resultados de esta primera versión del A$k$-LAESA utilizando datos artificiales, en concreto procedentes de dos clases gaussianas con distinta media[2] pero con la misma matriz de covarianza (la matriz unidad). Los datos empleados tienen diferentes dimensiones: 4, 6, 8 y 10, que se han distribuido en conjuntos de entrenamiento de tamaño creciente y conjuntos de prueba de 1000 prototipos. Para cada tamaño del conjunto de entrenamiento se han realizado varias repeticiones con distintos conjuntos de entrenamiento para obtener una mejor estimación de las tasas de error; concretamente, en algunos experimentos se han realizado 20 repeticiones y en otros 30 repeticiones. Por tanto, cada punto dibujado en las gráficas corresponde a la media de 20 valores diferentes en unos casos, y de 30 valores en otros.

Los resultados más interesantes son los que comparan las tasas de error del A$k$-LAESA con las de un clasificador basado en el vecino más cercano y un clasificador basado en los $k$ vecinos más cercanos; la figura 3.2 muestra estos resultados, en los que se puede apreciar que la tasa de error del A$k$-LAESA decrece según aumentan el tamaño del conjunto de entrenamiento y la dimensionalidad. También se comentan en (Moreno-Seco et al., 1999) los resultados obtenidos en cuanto a distancias calculadas por el A$k$-LAESA, que son siempre inferiores (en estas primeras versiones) que las del LAESA, como se puede observar en la figura 3.3. Además, al aumentar el valor de $k$ el número de distancias tiende a disminuir, mientras que si se buscan los $k$ vecinos más cercanos con el LAESA el número de distancias aumenta con el valor de $k$ (véase la figura 2.20 en el epígrafe 2.5).

[2] En un caso la media es el vector unitario, y en el otro es el mismo vector pero con un 2 en la primera coordenada.

```
algoritmo Ak-LAESA (versión 1)
    entrada  P ⊂ E (conjunto de prototipos)
             k (vecinos a utilizar para la clasificación)
             x (muestra a clasificar)
    usa      B ∈ P (conjunto de prototipos base)
             D_B ∈ ℝ^{|B|×|P|} (distancias a prot. base)
    salida   c (clase asignada a la muestra x)
comienzo
     para todo p ∈ P hacer G[p] := 0 fpara
     nn := desconocido; dnn := ∞
     s := ElementoArbitrario(B)          // el primer prototipo siempre es de B
>    si k = 1 entonces
>        TamLimite := 0                  // compatibilidad con LAESA
>    si no
>        TamLimite := k
>    fsi
>    mientras |P| > TamLimite o B ∩ P ≠ ∅ hacer
         d_s := d(x, s)
         P := P − {s}
         si d_s < d_nn entonces  nn := s ; d_nn := d_s  fsi // actualización de nn
         siguiente_B := desconocido; gmin_B := ∞
         siguiente := desconocido; gmin := ∞
         para todo p ∈ P hacer
             si s ∈ B entonces
                 G[p] := maximo(G[p],|D_B[s, p] − d_s|)   // actualización de cotas
             fsi
             si G[p] > dnn entonces
                 P := P − {p}                              // eliminación
             si no
                 si p ∈ B entonces
                     si G[p] < gmin_B entonces
                         gmin_B := G[p] ; siguiente_B := p // aproximación en B
                     fsi
                 si no
                     si G[p] < gmin entonces
                         gmin := G[p] ; siguiente := p // aproximación en P − B
                     fsi
                 fsi
             fsi
         fpara
         si siguiente_B ≠ desconocido entonces
             s := siguiente_B                // selección en B, si es posible
         si no
             s := siguiente                  // selección en P − B
         fsi
     fmientras
>    si |P| = 0 entonces
>        P := P ∪ {nn}                       // para evitar un conjunto vacío
>    fsi
>    c := Votacion(P)
fin Ak-LAESA (v1)
```

Figura 3.1: Algoritmo A$k$-LAESA (versión 1).

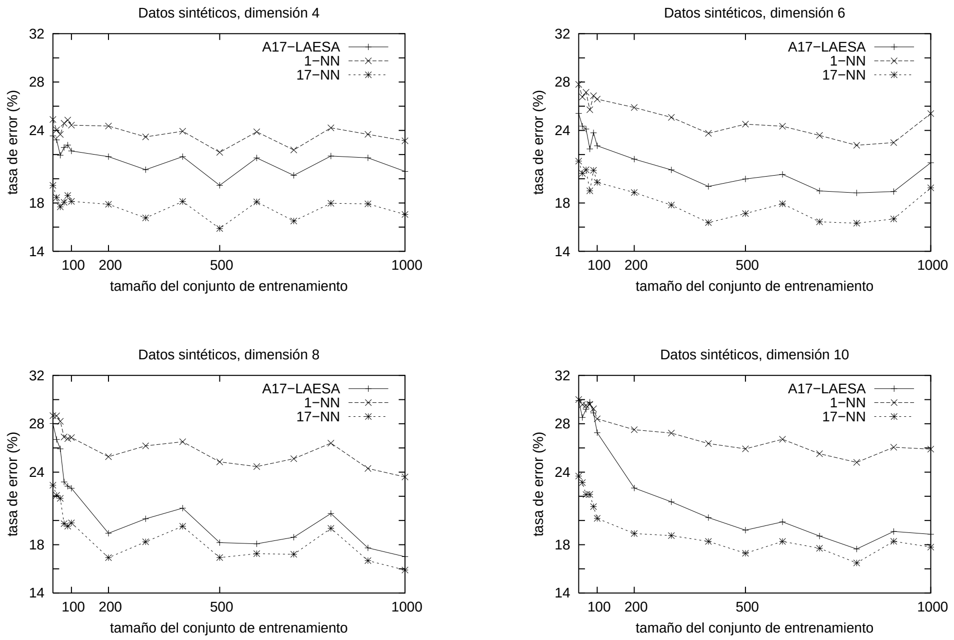


Figura 3.2: Tasas de error del A$k$-LAESA (versión 1), para $k = 17$ comparadas con las de clasificadores basados en el vecino más cercano (NN) y en los $k$ vecinos más cercanos (17-NN), para datos sintéticos de diferentes dimensiones.

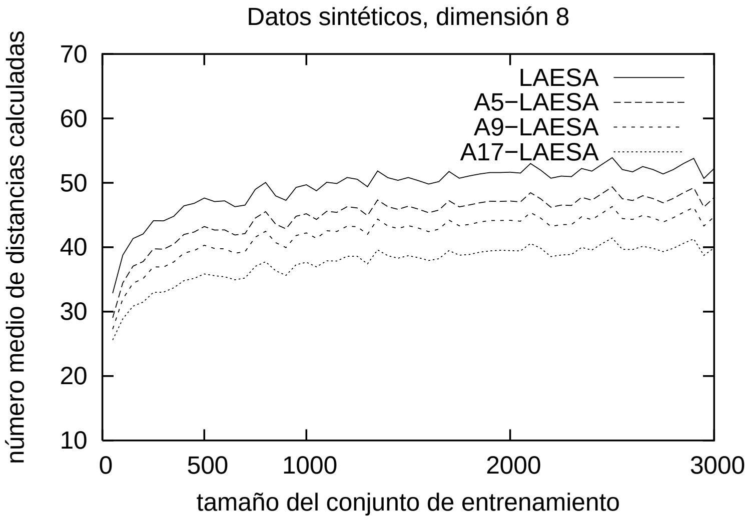


Figura 3.3: Comparación entre el número de distancias calculadas del LAESA y los del A$k$-LAESA (versión 1), para varios valores de $k$.

La conclusión que se extrae de (Moreno-Seco et al., 1999) es la siguiente: el A$k$-LAESA obtiene mejores tasas de acierto que el LAESA con la misma complejidad temporal y un número ligeramente menor de distancias calculadas y, además, la tasa de acierto del A$k$-LAESA se acerca a la de un clasificador basado en los $k$ vecinos más cercanos. Esta última cuestión ha sido el objetivo principal de las sucesivas mejoras realizadas al A$k$-LAESA: obtener una tasa de acierto lo más cercana posible a la de la regla de clasificación de los $k$ vecinos más cercanos, manteniendo un número de distancias calculadas similar al del LAESA.

### 3.1.2. Experimentos y resultados

Con el objetivo de poder comparar los resultados de todas las versiones del A$k$-LAESA con los mismos conjuntos de datos, se han realizado una serie de experimentos con bases de datos sintéticas y reales, cuyas características se describen con detalle en el apéndice A. En todos los casos se comparan las tasas de error del A$k$-LAESA con las del vecino más cercano y las de los $k$ vecinos más cercanos.

#### Resultados con la base de datos SINTE

El cuadro 3.1 resume las principales características de esta base de datos. En las figuras 3.4 y 3.5 se muestran las tasas de error del A$k$-LAESA para valores crecientes de $k$, y se incluye en ellas las tasas de error del $k$-LAESA, que obtiene exactamente los $k$ vecinos más cercanos, como referencia. Las tasas de error obtenidas con datos de dimensión 10 son similares a las obtenidas en los experimentos preliminares, pero con datos de dimensión 20 los resultados son sorprendentemente malos.[3] Estos últimos resultados indican que esta versión del A$k$-LAESA no es comparable con un algoritmo que utilice exactamente los $k$ vecinos más cercanos (como el $k$-LAESA) cuando la dimensión de los datos es elevada.

Los resultados referentes a número de distancias calculadas son similares a los mostrados en la figura 3.3, y se pueden consultar en el epígrafe 3.2.2, donde se comparan con el LAESA y el $k$-LAESA.

#### Resultados con la base de datos CROMOSOMAS

En el cuadro 3.2 se muestra un resumen de las principales características de esta base de datos. Los resultados de la versión 1 del A$k$-LAESA con esta base de datos son también muy malos, y se debe fundamentalmente a

[3] En experimentos posteriores (que no se reproducen en esta tesis por su escasa relevancia) se dedujo que la causa principal de estos resultados es el reducido tamaño del conjunto de entrenamiento en relación con la dimensionalidad (el número de prototipos base, fijado en 40, también influye aunque en menor medida). Para que esta primera versión del A$k$-LAESA obtenga buenos resultados se requieren muchos más prototipos.

| | |
|---|---|
| Tipo de prototipo | vector |
| Dimensiones | 10 y 20 |
| Número de clases | 8 y 32 |
| Tamaño del conjunto de entrenamiento | 1024, 4096 y 16384 |
| Tamaño del conjunto de test | 1024 |
| Número de pares entrenamiento/test | 10 |

Cuadro 3.1: Principales características de la base de datos SINTE.

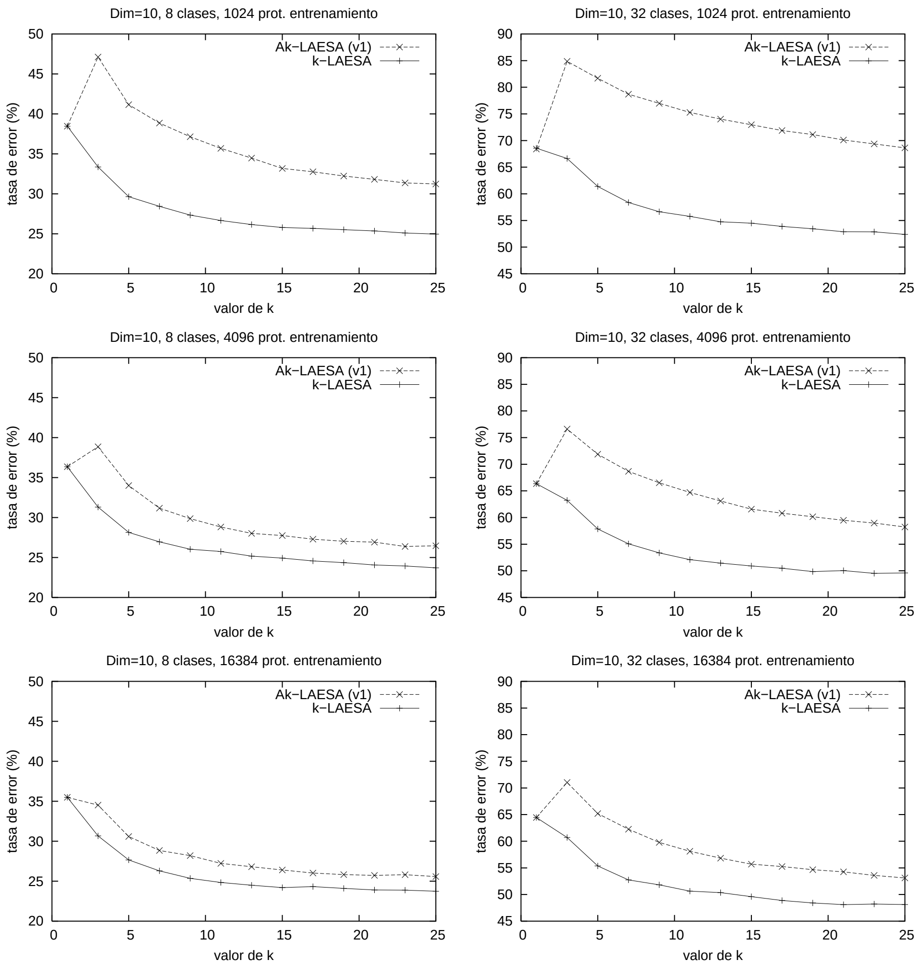


Figura 3.4: Tasas de error del A$k$-LAESA (versión 1) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 10 (base de datos SINTE). Las gráficas de la izquierda corresponden a datos de 8 clases, y las de la derecha a 32 clases. Las gráficas aparecen por orden creciente del tamaño del conjunto de entrenamiento (1024, 4096 y 16384).

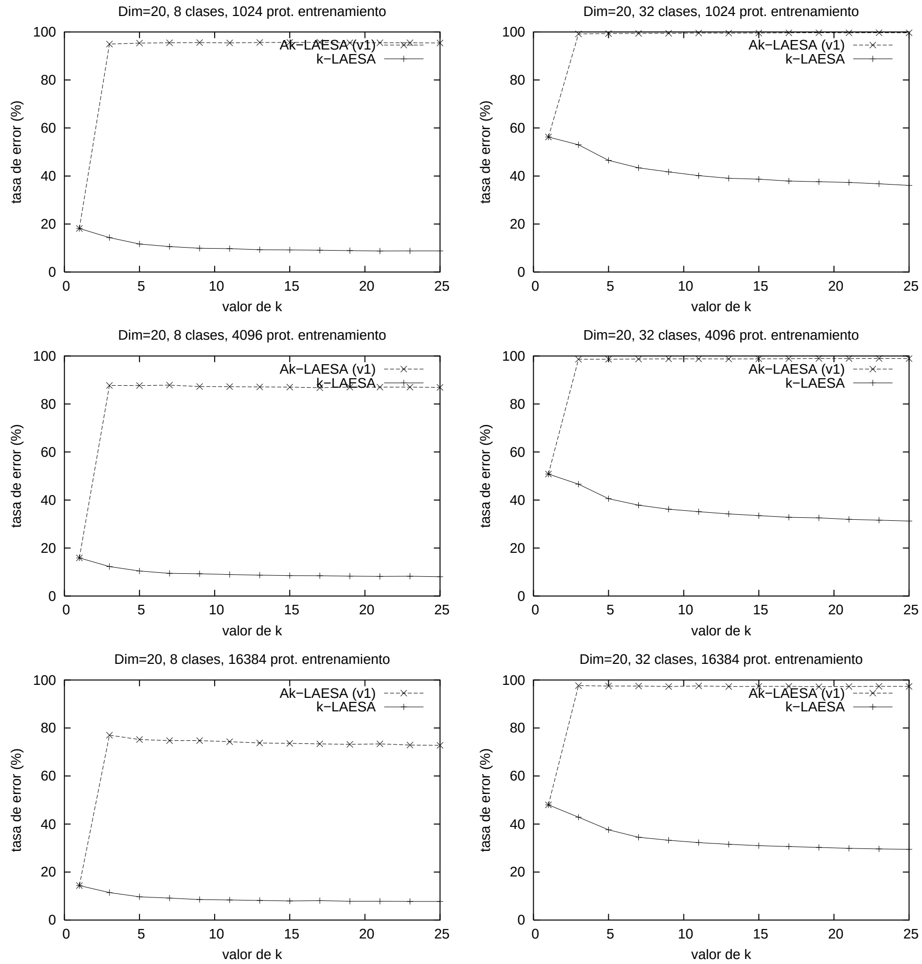


Figura 3.5: Tasas de error del A$k$-LAESA (versión 1) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 20 (base de datos SINTE). Las gráficas de la izquierda corresponden a datos de 8 clases, y las de la derecha a 32 clases. Las gráficas aparecen por orden creciente del tamaño del conjunto de entrenamiento (1024, 4096 y 16384).

| | |
|---|---|
| Tipo de prototipo | cadena de caracteres |
| Número de clases | 22 |
| Tamaño del conjunto de entrenamiento | 2200 |
| Tamaño del conjunto de test | 2200 |
| Número de pares entrenamiento/test | 2 |

Cuadro 3.2: Principales características de la base de datos CROMOSOMAS.

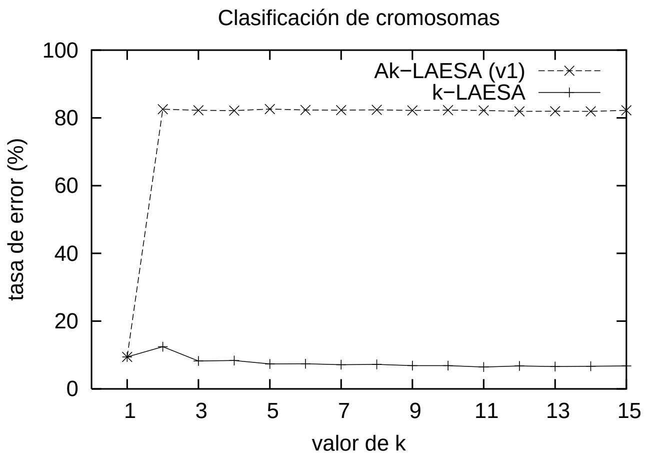


Figura 3.6: Tasa de error del A$k$-LAESA (versión 1) en la clasificación de cromosomas, en comparación con la tasa de error del $k$-LAESA.

la alta dimensionalidad (intrínseca) de los datos en relación con el tamaño del conjunto de datos, limitado y generalmente pequeño como suele ser habitual en conjuntos de datos reales. La figura 3.6 muestra la tasa de error de la versión 1 del A$k$-LAESA según aumenta el tamaño de $k$, y muestra también la tasa de error de los $k$ vecinos más cercanos ($k$-LAESA, concretamente).

## 3.2. A$k$-LAESA (versión 2)

La segunda de las mejoras sobre el algoritmo LAESA (Moreno-Seco et al., 2000, 2001) para la clasificación se basa en recuperar siempre el vecino más cercano hasta el momento para la votación, lo cual puede llevar a casos en los que se utilicen $k+1$ prototipos en la votación. La figura 3.7 muestra la mínima modificación que habría que realizar en la parte final del A$k$-LAESA (versión 1) de la figura 3.1 para obtener la versión 2.

La motivación de esta segunda versión consiste en que, en la versión 1, en muchos casos el vecino más cercano se eliminaba de la votación al ser seleccionado, ya que el LAESA elimina inmediatamente los prototipos selec-

**algoritmo** A$k$-LAESA (versión 2)

... *(exactamente igual que la versión 1)*

**fmientras**
$P := P \cup \{nn\}$ *// recuperar siempre el vecino más cercano*
$c :=$ Votacion($P$)
**fin** A$k$-LAESA (v2)

Figura 3.7: Algoritmo A$k$-LAESA (versión 2).

cionados, una vez ha calculado su distancia a la muestra y ha actualizado el vecino más cercano hasta el momento. Cuando el valor de $k$ es grande, se puede pensar que el vecino más cercano todavía no ha sido seleccionado, y por tanto si la fase de búsqueda termina en ese momento resultará incluido en la votación. Sin embargo, para valores de $k$ pequeños, el vecino más cercano muy probablemente ya haya sido seleccionado y guardado como vecino más cercano hasta el momento, y a la vez se habrá eliminado del conjunto de entrenamiento (como todos los prototipos seleccionados). Por tanto, si la votación se realiza únicamente con los prototipos que quedan por seleccionar (de los cuales ninguno va a ser probablemente el vecino más cercano), es previsible que los resultados no sean tan buenos como si se incluye el vecino más cercano en la votación. Como se comenta más adelante, las tasas de acierto en la clasificación aumentaron al incluir siempre el vecino más cercano hasta el momento en la votación.

### 3.2.1. Resultados preliminares

En este epígrafe solamente se muestran resultados de esta segunda versión en comparación con clasificadores basados en los $k$ vecinos más cercanos y en los $k$ vecinos de centroide más cercano (Sánchez et al., 2000), que son los resultados presentados en (Moreno-Seco et al., 2000, 2001); la comparación con la versión 1 del A$k$-LAESA se realiza en el siguiente epígrafe.

Para probar el comportamiento del A$k$-LAESA (versión 2) se han realizado (Moreno-Seco et al., 2000) experimentos con datos sintéticos generados utilizando el algoritmo de Jain y Dubes (1988), que permite generar datos aleatorios procedentes de gaussianas con un cierto solapamiento máximo.[4] En este caso, se han generado muestras de dimensiones 6 y 10, utilizando un solapamiento de 0.04 (para obtener tasas de error bajas), para obtener conjuntos de entrenamiento de hasta 8192 prototipos. En un primer experimento

[4] En los experimentos preliminares de la versión 1 del A$k$-LAESA se fijaron las medias de forma arbitraria, mientras que en esta versión se ha utilizado el algoritmo de Jain y Dubes, que elige aleatoriamente las medias para conseguir el solapamiento especificado.

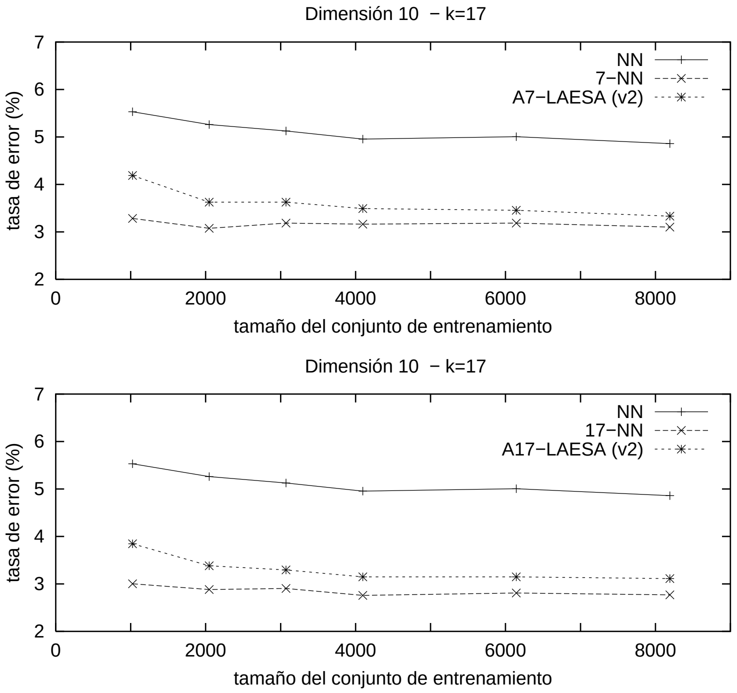


Figura 3.8: Tasas de error del A$k$-LAESA (versión 2), para $k = 17$ y $k = 7$, comparadas con las de clasificadores basados en el vecino más cercano (NN) y en los $k$ vecinos más cercanos.

se han realizado 16 repeticiones con diferentes conjuntos de entrenamiento de cada tamaño, con conjuntos de test de 512 prototipos. Los resultados con estos datos (algunos de los cuales se muestran en la figura 3.8) confirman los obtenidos con la versión 1 del A$k$-LAESA: la tasa de error del A$k$-LAESA es siempre mejor que la de un clasificador basado en el vecino más cercano, y se acerca según aumentan la dimensión y el tamaño del conjunto de entrenamiento a la tasa de error de un clasificador basado en los $k$ vecinos más cercanos.

En (Moreno-Seco et al., 2000) también se presentan unos experimentos[5] donde se estudia el comportamiento del A$k$-LAESA según aumenta el valor de $k$. En estos experimentos se ha comprobado que a partir de cierto valor de $k$ la tasa de error se mantiene estable, sin variaciones significativas (véase la figura 3.9). Este es un comportamiento que se observa también en los clasificadores basados en los $k$ vecinos más cercanos en problemas reales, si bien es cierto que a partir de un valor generalmente muy elevado de $k$ la tasa de error empieza a aumentar, lo cual no sucede de forma tan acusada

[5]Como en el experimento anterior, se han realizado 16 repeticiones de cada experimento, con 4096 prototipos de entrenamiento y 512 de test.

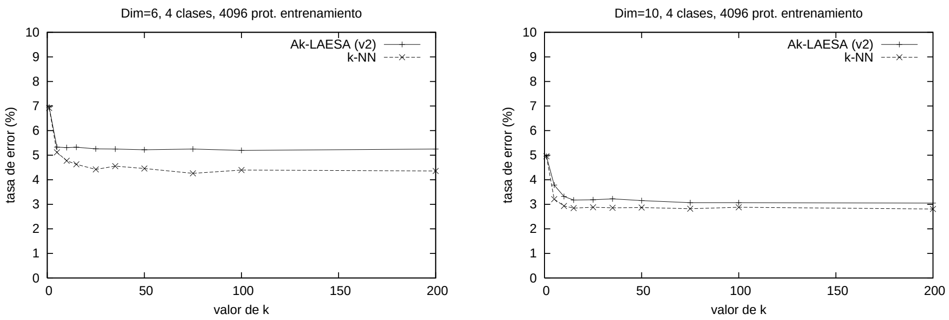


Figura 3.9: Tasas de error del A$k$-LAESA (versión 2), para valores crecientes de $k$ con datos de dimensiones 6 y 10.

| Conjunto | No. de prototipos | Dimensionalidad | No. de clases |
|---|---|---|---|
| CERVICAL | 64 | 10 | 5 |
| DORSAL | 186 | 14 | 12 |
| LUMBAR | 68 | 14 | 5 |

Cuadro 3.3: Características de los datos extraídos de vértebras humanas (Moreno-Seco et al., 2001).

en el A$k$-LAESA por las restricciones propias del algoritmo, ya que a partir de cierto valor de $k$, utiliza siempre los mismos prototipos para clasificar: aquellos que quedan sin eliminar después de haber seleccionado y eliminado todos los prototipos base. En algunos casos, este comportamiento del A$k$-LAESA puede ser una ventaja frente a los clasificadores basados en los $k$ vecinos más cercanos.

Por otro lado, en (Moreno-Seco et al., 2001) se presentan experimentos con datos reales extraídos a partir de imágenes de vértebras humanas. En este caso concreto, los conjuntos de datos son pequeños (algo habitual cuando se trabaja con datos reales procedentes del campo de la medicina) y se ha utilizado la técnica de *leave-one-out* para obtener resultados más significativos. Los datos originales se han clasificado en tres conjuntos de datos, los correspondientes a las vértebras cervicales, dorsales y lumbares, cuyas características aparecen en el cuadro 3.3. Cada prototipo se representa como un vector de características, y la distancia utilizada ha sido la distancia euclídea, como en los experimentos con datos sintéticos.

El A$k$-LAESA se compara en este caso con un clasificador basado en los $k$ vecinos más cercanos y con un clasificador basado en los $k$ vecinos de centroide más cercano, $k$-NCN (Sánchez et al., 2000), que es un clasificador especialmente adecuado para conjuntos reducidos de datos. Los experimen-

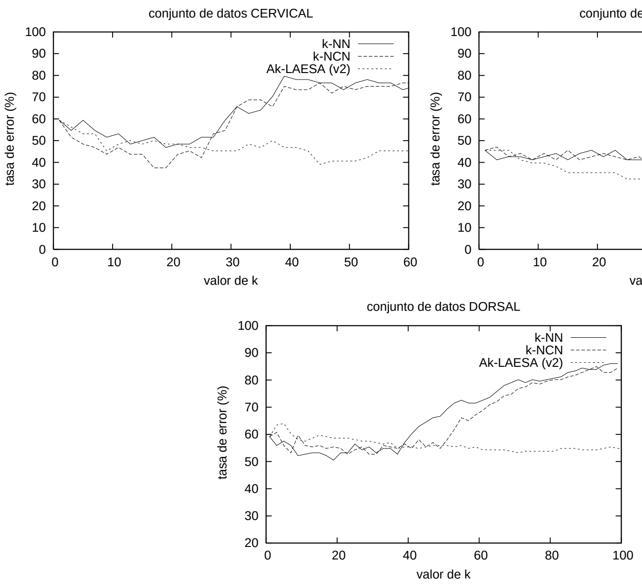


Figura 3.10: Tasas de error del A$k$-LAESA (versión 2), comparadas con $k$-NN y $k$-NCN con datos de vértebras humanas.

| Clasificador | CERVICAL | DORSAL | LUMBAR |
|---|---|---|---|
| k-NN | 46.875 (19) | **50.53** (19) | 41.17 (3) |
| k-NCN | **37.5** (17) | 52.68 (23) | 38.23 (29) |
| Ak-LAESA | 39.06 (45) | 53.22 (71) | **32.35** (29) |

Cuadro 3.4: Mejores resultados del A$k$–LAESA (v2), $k$–NN y $k$–NCN, con datos de vértebras humanas. El valor entre paréntesis corresponde al valor de $k$ para el que se obtiene el mejor resultado.

tos se han realizado para un valor creciente de $k$, y los resultados obtenidos se muestran en la figura 3.10. Los tres clasificadores obtienen resultados similares, y cada clasificador es el mejor (eligiendo un valor adecuado de $k$) en uno de los conjuntos de datos (véase el cuadro 3.4), lo cual nos lleva a la conclusión de que el A$k$-LAESA puede obtener resultados similares o a veces mejores que los de otros clasificadores con datos reales.

### 3.2.2. Experimentos y resultados

Para comparar los resultados de la versión 2 del A$k$-LAESA con los de la versión 1 se han reproducido los experimentos realizados en el epígrafe 3.1.2 con la versión 2. Los experimentos se han realizado utilizando las bases de datos SINTE (datos sintéticos procedentes de gaussianas) y CROMOSO-

MAS (características de cromosomas humanos codificadas como cadenas de caracteres).

### Resultados con la base de datos SINTE

Las figuras 3.11 y 3.12 muestran los resultados con esta base de datos de las versiones 1 y 2 del A$k$-LAESA, y también los del $k$-LAESA como referencia. En todos los casos los resultados de la versión 2 son mejores que los de la versión 1, especialmente para valores pequeños de $k$; cuando el valor de $k$ aumenta las diferencias son mínimas. En ambas versiones las tasas de error se acercan a las tasas de los $k$ vecinos más cercanos, pero éstas últimas son todavía mejores (más bajas); la diferencia con respecto a los $k$ vecinos más cercanos es importante en estas primeras versiones del A$k$-LAESA.

El número de distancias calculadas de la versión 2, como sucede en la versión 1, es inferior al del LAESA, y desciende al aumentar el valor de $k$. En la figura 3.13 se muestra el número medio de distancias calculadas por estas dos versiones (que coincide, como era de esperar) con respecto al número calculado por el LAESA (para $k = 1$). Solamente se muestran los resultados con 4096 prototipos de entrenamiento por no repetir gráficas muy parecidas con otros tamaños. Además, la figura 3.14 muestra el número de distancias calculadas por la versión 2 del A$k$-LAESA en comparación con las calculadas por el $k$-LAESA, con los mismos datos.

### Resultados con la base de datos CROMOSOMAS

Como ocurre con la versión 1, los resultados de la versión 2 son muy pobres con datos de dimensionalidad alta como los de esta base de datos. La figura 3.15 muestra las tasas de error de las dos versiones del A$k$-LAESA según aumenta el valor de $k$; en este caso la tasa de error de la versión 2 es mejor, pero sigue siendo muy mala en comparación con la de los $k$ vecinos más cercanos.

## 3.3. A$k$-LAESA (versión 3)

En una tercera fase constatamos que los candidatos a vecino más cercano que no resultaban ser el vecino más cercano (cuya distancia a la muestra ya se había calculado) podrían ser unos buenos candidatos para entrar en la votación para clasificar la muestra, probablemente mejores que aquellos prototipos no eliminados y cuya distancia no se había calculado todavía. De esta idea surgió la tercera versión del A$k$-LAESA, en la que no se eliminan nunca los candidatos a vecino más cercano, salvo cuando resultan eliminados normalmente, es decir, cuando su cota inferior es mayor que la distancia al más cercano hasta el momento; el algoritmo termina cuando quedan como

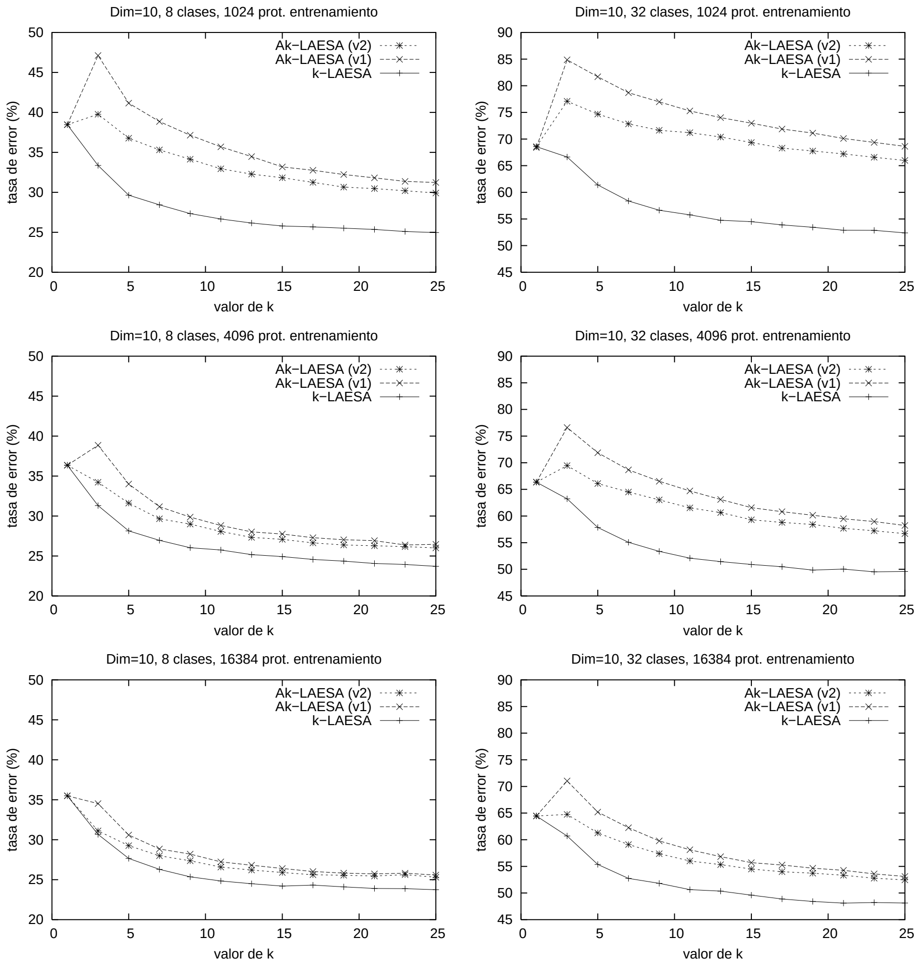


Figura 3.11: Tasas de error del A$k$-LAESA (versiones 1 y 2) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 10.

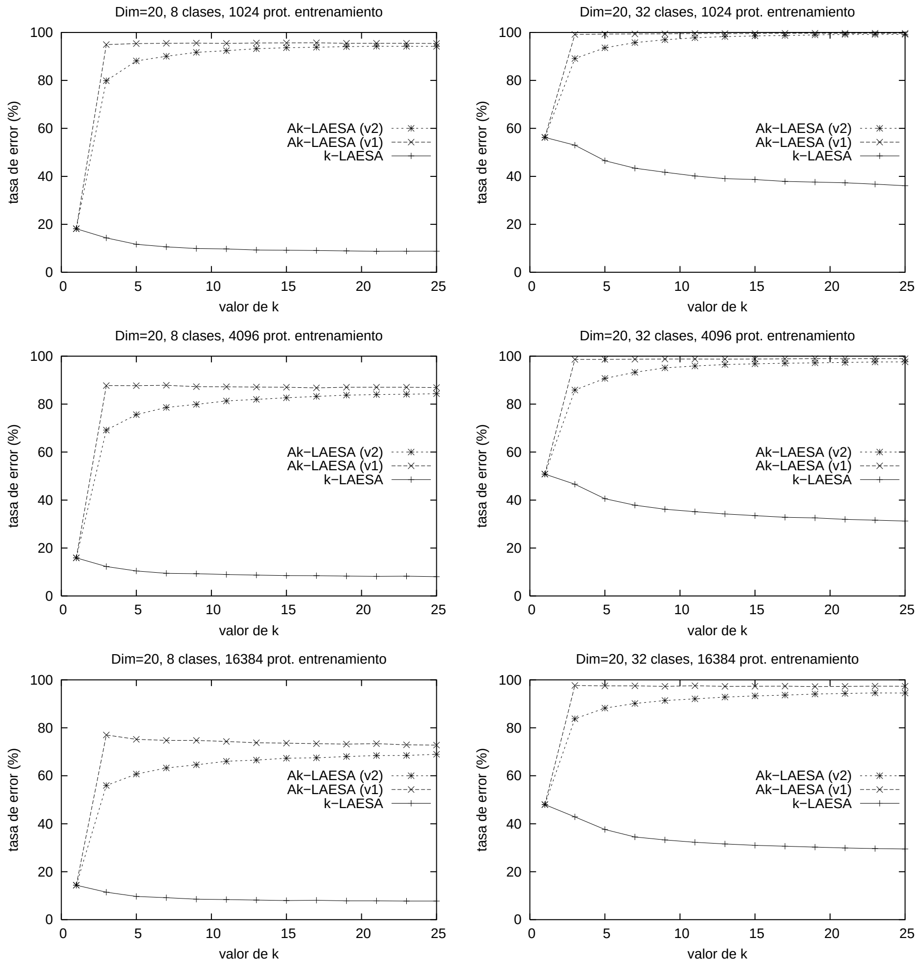


Figura 3.12: Tasas de error del A$k$-LAESA (versiones 1 y 2) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 20.

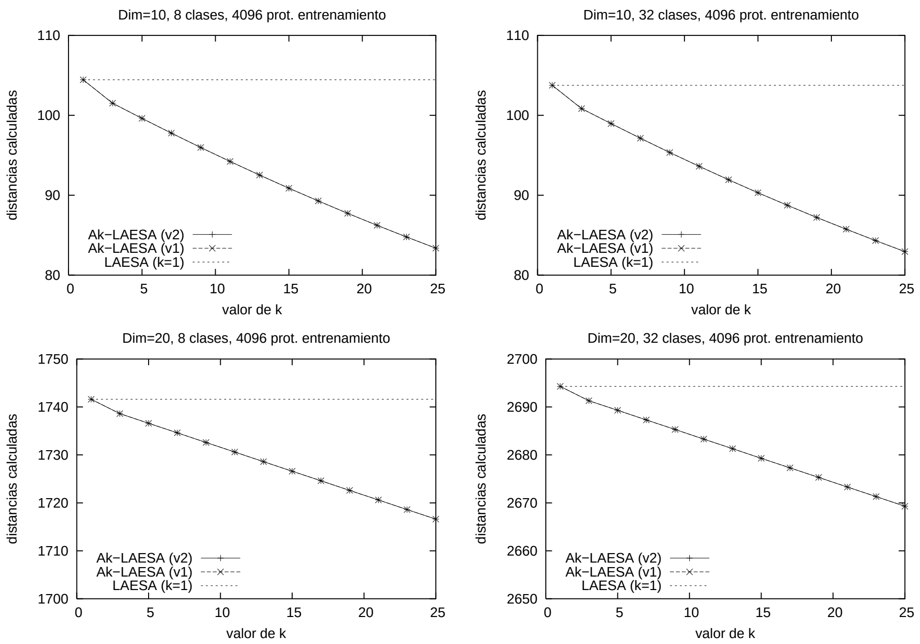


Figura 3.13: Distancias calculadas por el A$k$-LAESA (versiones 1 y 2) en comparación con las del LAESA, para datos sintéticos.

mucho $k$ prototipos sin eliminar o no queda ningún prototipo por seleccionar como candidato a vecino más cercano.

La figura 3.16 contiene esta tercera versión del A$k$-LAESA, en la que se han marcado en el margen izquierdo las principales diferencias con el LAESA. En esta versión la condición de parada es un poco más compleja, y la votación ya no se realiza con todos los prototipos que quedan sin eliminar en $P$ sino que se utilizan los $k$ mejores prototipos cuya distancia se ha calculado y, si es necesario, se utilizan también los demás prototipos no eliminados, tomando aquellos con menor cota. En la práctica, muy pocas veces se utilizan prototipos de los que no se tenga calculada su distancia a la muestra (prototipos no seleccionados).

### 3.3.1. Experimentos y resultados

La versión 3 del A$k$-LAESA es la primera que obtiene resultados prácticamente idénticos a los de los $k$ vecinos más cercanos, aunque siempre o casi siempre ligeramente peores. En cualquier caso, esta versión obtiene tasas de error muy parecidas a las de los $k$ vecinos más cercanos con el mismo tiempo de clasificación (prácticamente) que el LAESA, mientras que el tiempo de clasificación del $k$-LAESA aumenta con el valor de $k$ (véase la figura 2.20 en el epígrafe 2.5).

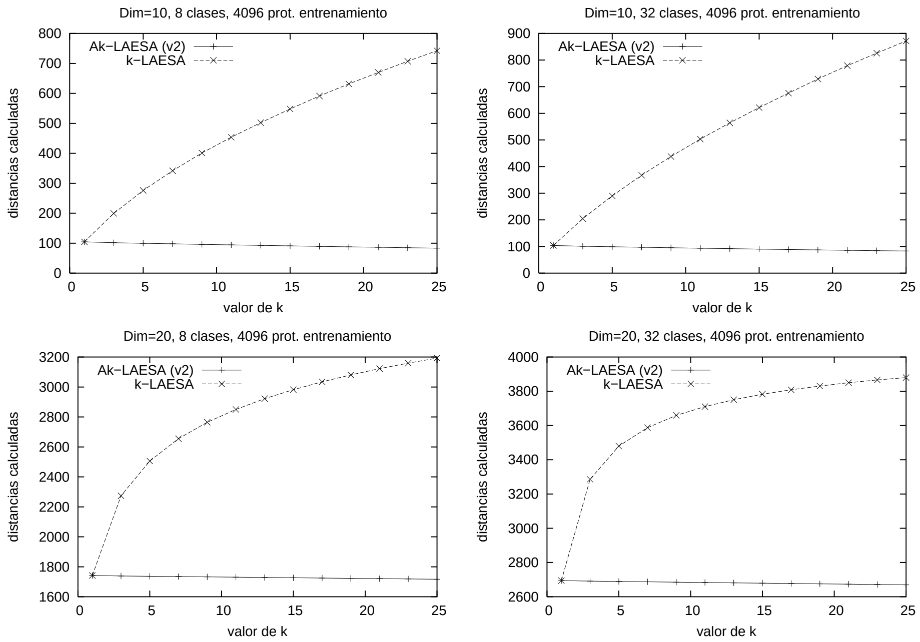


Figura 3.14: Distancias calculadas por el A$k$-LAESA (versión 2) en comparación con las del $k$-LAESA, para datos sintéticos.

### Resultados con la base de datos SINTE

Las figuras 3.17 y 3.18 muestran los resultados con esta base de datos de las versiones 1, 2 y 3 del A$k$-LAESA, y también los del $k$-LAESA como referencia. Para los datos de dimensión 20 (figura 3.18) se ha ajustado la escala para apreciar la casi nula diferencia entre la tasa de error del A$k$-LAESA (versión 3) y la de los $k$ vecinos más cercanos, lo cual hace que las tasas de las versiones anteriores no aparezcan en las gráficas (se pueden consultar en epígrafes anteriores).

Como se puede apreciar en las figuras, las diferencias entre la tasa de error del A$k$-LAESA (versión 3) y la de un clasificador basado en los $k$ vecinos más cercanos (como el $k$-LAESA) son muy pequeñas (prácticamente inexistentes en dimensión 20), aunque suben ligeramente al aumentar el valor de $k$. Al contrario que las versiones anteriores, la versión 3 obtiene tasas de error similares a las de los $k$ vecinos más cercanos incluso con dimensionalidades altas.

### Resultados con la base de datos CROMOSOMAS

La figura 3.19 muestra las tasas de error de las tres versiones del A$k$-LAESA según aumenta el valor de $k$; en este caso, la tasa de error de la

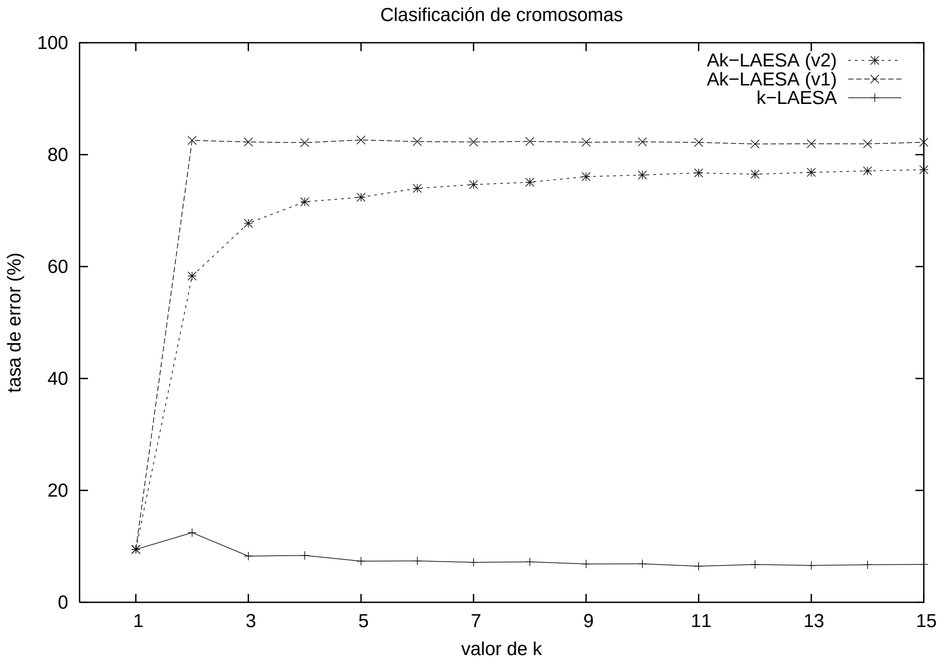


Figura 3.15: Tasa de error del A$k$-LAESA (versiones 1 y 2) en la clasificación de cromosomas, en comparación con la tasa de error del $k$-LAESA.

```
algoritmo Ak-LAESA (versión 3)
    entrada  P ⊂ E (conjunto de prototipos)
             k (vecinos a utilizar para la clasificación)
             x (muestra a clasificar)
    usa      B ∈ P (conjunto de prototipos base)
             D_B ∈ ℝ^{|B|×|P|} (distancias a prot. base)
    salida   c (clase asignada a la muestra x)
comienzo
    para todo p ∈ P hacer G[p] := 0 ; YaSeleccionado[p] := falso fpara
    nn := desconocido; dnn := ∞
    s := ElementoArbitrario(B)          // el primer prototipo siempre es de B
>   terminar := falso
>   mientras |P| > 0 y no terminar hacer
        d_s := d(x, s)
>       YaSeleccionado[s] := cierto
        si d_s < d_nn entonces   nn := s ; d_nn := d_s   fsi // actualización de nn
        siguiente_B := desconocido; gmin_B := ∞
        siguiente := desconocido; gmin := ∞
        para todo p ∈ P hacer
            si s ∈ B entonces
                G[p] := maximo(G[p],|D_B[s, p] − d_s|)   // actualización de cotas
            fsi
            si G[p] > dnn entonces
                P := P − {p}                              // eliminación
            si no
                si p ∈ B entonces
>                   si G[p] < gmin_B y no YaSeleccionado[p] entonces
                        gmin_B := G[p] ; siguiente_B := p  // aproximación en B
                    fsi
                si no
>                   si G[p] < gmin y no YaSeleccionado[p] entonces
                        gmin := G[p] ; siguiente := p // aproximación en P − B
                    fsi
                fsi
            fsi
        fpara
        si siguiente_B ≠ desconocido entonces
            s := siguiente_B                      // selección en B, si es posible
        si no
            s := siguiente                        // selección en P − B
        fsi
>       si |P| ≤ 0 o s = desconocido entonces
>           terminar := cierto
>       fsi
    fmientras
>   c := VotacionConKMejores(P)
fin Ak-LAESA (v1)
```

Figura 3.16: Algoritmo A$k$-LAESA (versión 3).

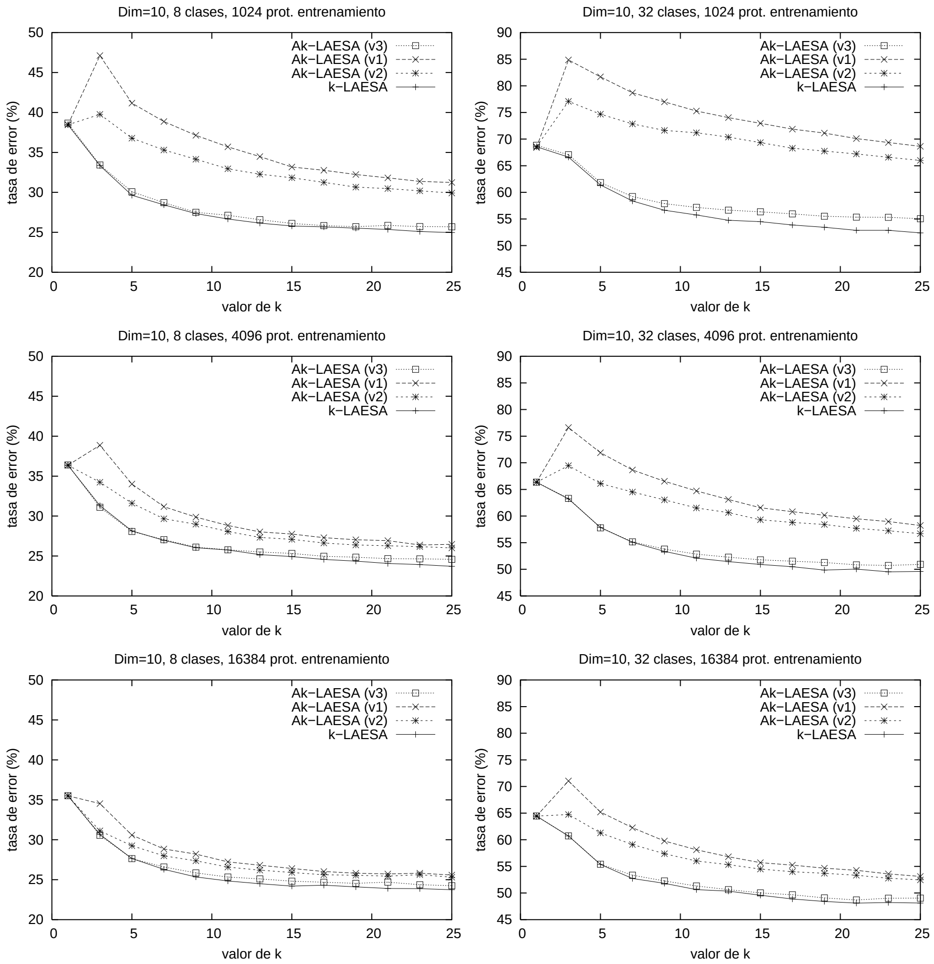


Figura 3.17: Tasas de error del A$k$-LAESA (versiones 1, 2 y 3) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 10.

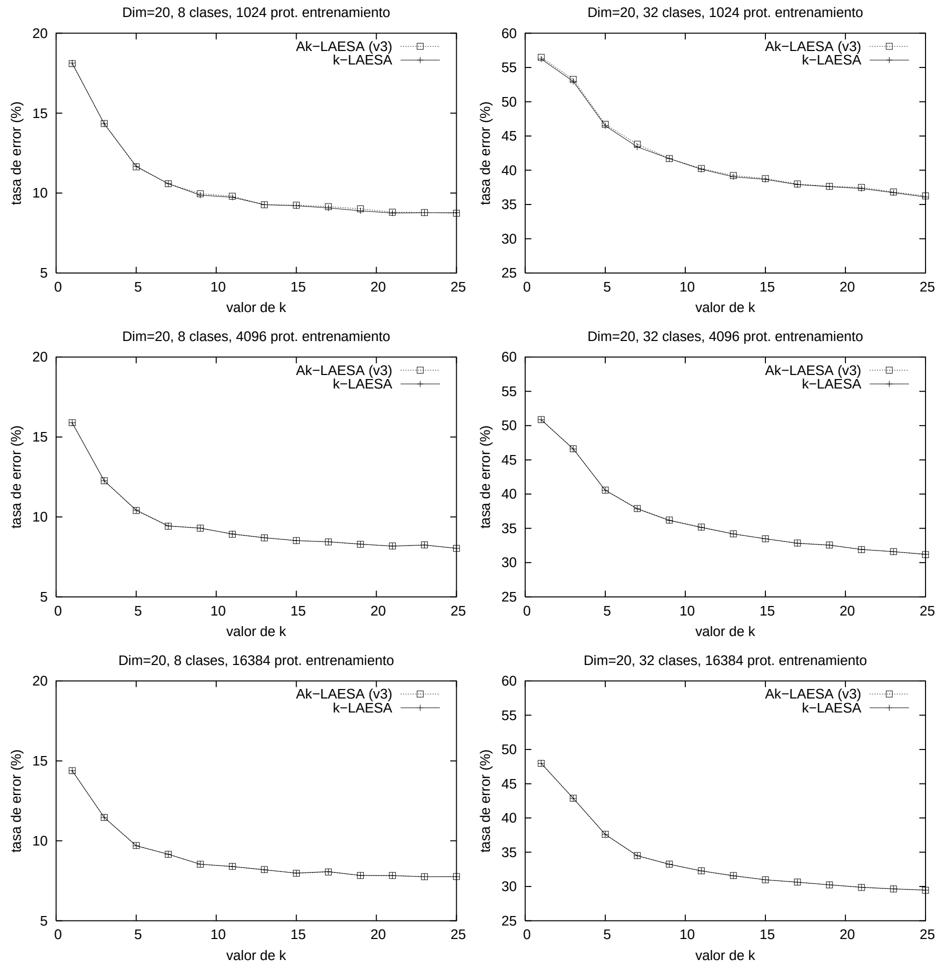


Figura 3.18: Tasas de error del A$k$-LAESA (versiónes 3) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 20. Los resultados de las versiones 1 y 2 del A$k$-LAESA se salen de la gráfica (véase la figura 3.12).

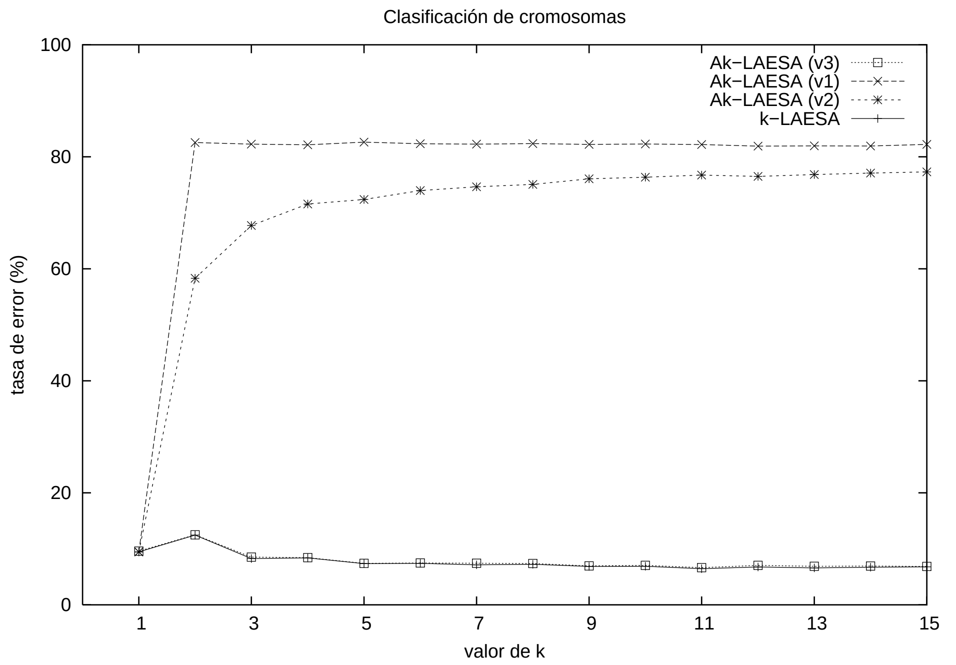


Figura 3.19: Tasa de error del A$k$-LAESA (versiones 1, 2 y 3) en la clasificación de cromosomas, en comparación con la tasa de error del $k$-LAESA.

versión 3 del A$k$-LAESA es prácticamente idéntica a la de los $k$ vecinos más cercanos, y por tanto es mucho mejor que la de las versiones anteriores.

## 3.4. A$k$-LAESA (versión 4)

En la práctica, todos o casi todos los prototipos que se utilizaban en las votaciones en la versión 3 del A$k$-LAESA eran prototipos que previamente habían sido candidatos, y cuya distancia ya se había calculado. Además, de todas las posibles variantes del LAESA (véase el epígrafe 2.4.2) siempre se ha utilizado aquella en la que primero se seleccionan prototipos base, y solamente cuando todos ellos ya han sido seleccionados se eligen prototipos del resto del conjunto de entrenamiento. Esto último nos llevó a reformular la versión simplificada de la figura 2.16 de la siguiente manera (véase la figura 3.20):

1. En una primera parte, se calculan las distancias de cada prototipo base a la muestra y se calcula la cota inferior de la distancia para todos los prototipos.
2. En la segunda parte se van obteniendo candidatos a vecino más cercano, de menor a mayor cota. Aunque hay implementaciones más efi-

```
algoritmo LAESA (versión simplificada y reformulada)
    entrada  P ⊂ E (conjunto de prototipos)
             x (muestra a clasificar)
    usa      B ∈ P (conjunto de prototipos base)
             D_B ∈ ℝ^{|B|×|P|} (distancias a prot. base)
    salida   nn (vecino más cercano a la muestra x)
comienzo
     para todo p ∈ P hacer G[p] := 0 fpara
     para todo b ∈ B hacer
          dxb := d(x, b)
          G[p] := maximo(G[p],|D_B[b, p] − dxb|)          // actualización de cotas
     fpara
     ordenar(G)
     nn := desconocido; dnn := ∞
     i := 1
     mientras G[i] < dnn hacer
          p := Prototipo(i)
(a)       dxp := d(x, p)
          si dxp < dnn entonces                           // actualización de nn
               nn := p ; dnn := dxp
          fsi
          i := i + 1                       // siguiente prototipo según las cotas
     fmientras
fin LAESA (versión simplificada y reformulada)
```

Figura 3.20: Versión reformulada del LAESA.

cientes, la más sencilla consiste en almacenar las cotas en un vector y ordenarlo, de forma que en esta segunda parte simplemente se recorre el vector previamente ordenado.

3. Para cada candidato se calcula su distancia a la muestra y se actualiza el vecino más cercano hasta el momento.

4. El algoritmo termina cuando la siguiente cota en el vector es mayor que la distancia al vecino más cercano hasta ese momento.

Esta versión simplificada se puede modificar para la clasificación utilizando $k$ vecinos simplemente almacenando de manera ordenada los $k$ mejores candidatos a vecino más cercano, cuyas distancias a la muestra se han calculado. Esta última modificación constituye el A$k$-LAESA en su versión 4 (Moreno-Seco et al., 2003c), que obtiene mejores resultados que las versiones 1 y 2 y resultados equivalentes en prácticamente todos los casos a los de la tercera versión del LAESA.[6] Esta última versión (véase la figura 3.21)

[6] Puesto que en la práctica todos o casi todos los prototipos utilizados por la versión 3 habían sido seleccionados (y por tanto serían los utilizados por la versión 4), es compren-

**algoritmo** A$k$-LAESA (versión 4)
  **entrada** $P \subset E$ (conjunto de prototipos)
    $x$ (muestra a clasificar)
  **usa** $B \in P$ (conjunto de prototipos base)
    $D_B \in \mathbb{R}^{|B|\times|P|}$ (distancias a prot. base)
  **salida** $c$ (clase asignada a la muestra $x$)
**comienzo**
  **para todo** $p \in P$ **hacer** $G[p] := 0$ **fpara**
  **para todo** $b \in B$ **hacer**
    $dxb := d(x, b)$
    $G[p] :=$ maximo($G[p]$,$|D_B[b, p] - dxb|$) *// actualización de cotas*
  **fpara**
  ordenar($G$)
  $nn :=$ desconocido; $dnn := \infty$ ; $V := \emptyset$
  $i := 1$
  **mientras** $G[i] < dnn$ **hacer**
    $p :=$ Prototipo($i$)
    $dxp := d(x, p)$
    **si** $dxp < dnn$ **entonces** *// actualización de nn*
      $nn := p$ ; $dnn := dxp$
    **fsi**
>   $V :=$ GuardarKMejores($V \cup \{ p \}$)
    $i := i + 1$ *// siguiente prototipo según las cotas*
  **fmientras**
>  $c :=$ Votacion($V$)
**fin** A$k$-LAESA (v4)

Figura 3.21: Algoritmo A$k$-LAESA (versión 4). El conjunto $V$ se puede implementar como un vector ordenado por las distancias.

calcula exactamente el mismo número de distancias que el LAESA, mientras que las anteriores obtenían en general un número inferior de distancias (véase la figura 3.13); sin embargo, este aumento se compensa con una mejor tasa de clasificación en todos los casos.

Como veremos en el siguiente capítulo, la idea de utilizar para la clasificación los $k$ mejores candidatos a vecino más cercano que se obtienen mientras se busca el vecino más cercano, incluyendo el vecino más cercano (que es el último candidato y el mejor), se ha extendido con éxito a otros algoritmos basados, como el LAESA, en un esquema de aproximación y eliminación.

### 3.4.1. Resultados preliminares

En (Moreno-Seco et al., 2003c) se presenta una nueva formulación del LAESA (similar a la figura 3.20) y la modificación necesaria para el A$k$-

sible que los resultados sean prácticamente iguales.

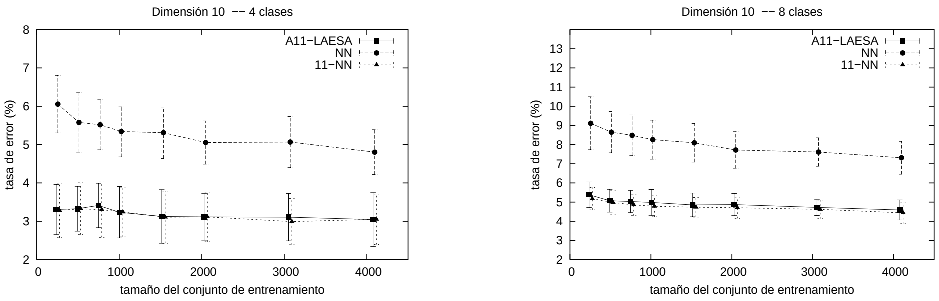


Figura 3.22: Tasas de error del A$k$-LAESA (versión 4), para $k = 11$ comparadas con las de clasificadores basados en el vecino más cercano (NN) y en los $k$ vecinos más cercanos (11-NN).

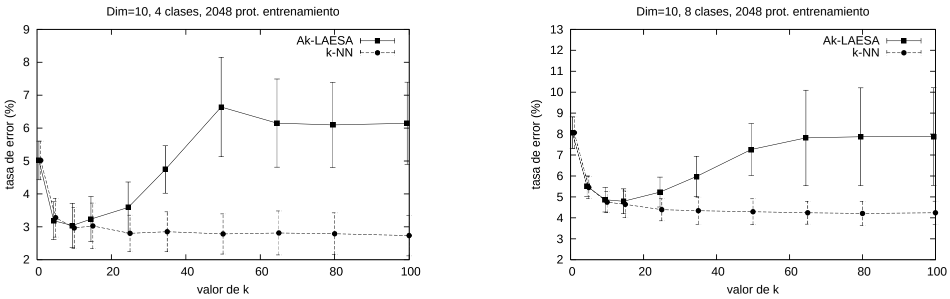


Figura 3.23: Tasas de error del A$k$-LAESA (versión 4), para valores de $k$ crecientes.

LAESA (versión 4). Se han realizado experimentos con datos sintéticos de dimensión 10 (de 4 y 8 clases) para comparar la tasa de error de un clasificador basado en el vecino más cercano con la de un clasificador basado en los $k$ vecinos más cercanos y con la del A$k$-LAESA, resultando que estos últimos obtienen tasas similares de error (véase la figura 3.22). Además, se ha estudiado la evolución de estas tasas de error según aumenta el valor de $k$ (véase la figura 3.23), observándose que en el A$k$-LAESA la tasa de error sube para valores grandes de $k$ (a partir de 20); este resultado se debe a lo rápido que se aproxima el A$k$-LAESA al vecino más cercano, con lo que los candidatos a vecino más cercano seleccionados en los últimos pasos del algoritmo están realmente cerca de la muestra, pero los primeros no están tan cerca y si se utilizan para votar los resultados empeoran.

También se han realizado experimentos con datos reales, en concreto con dígitos manuscritos de cuyo contorno se han extraído cadenas de caracteres (los detalles de la técnica para obtener las cadenas se describen en (Gómez et al., 1995; Micó y Oncina, 1998)). La distancia utilizada ha sido la distancia

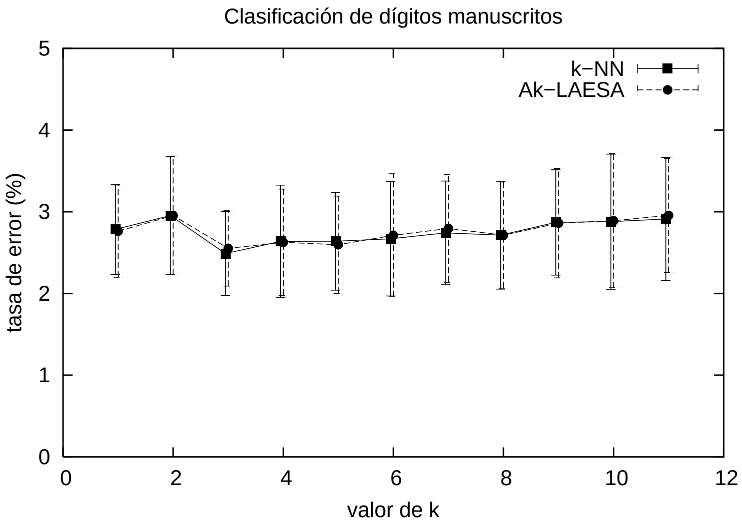


Figura 3.24: Comparación de A$k$-LAESA con $k$-NN para dígitos manuscritos.

de edición de cadenas o distancia de Levenshtein, y los resultados obtenidos son similares a los obtenidos con datos sintéticos (véase la figura 3.24): la tasa de error del A$k$-LAESA es prácticamente idéntica a la de un clasificador basado en los $k$ vecinos más cercanos, incluso cuando no es mucho menor que la del vecino más cercano, como es este caso.

### 3.4.2. Experimentos y resultados

Los resultados de la versión 4 del A$k$-LAESA son prácticamente idénticos a los de la versión 3 (en lo referente a tasas de error), tanto con datos sintéticos como con datos reales. En las gráficas que se muestran a continuación solamente aparecen los resultados de las versiones 3 y 4 del A$k$-LAESA, los de las versiones anteriores son en todos los casos peores.

Para ilustrar la ventaja de esta última versión del A$k$-LAESA con respecto a la búsqueda exacta de los $k$ vecinos más cercanos, hemos realizado una comparación entre el número de distancias que calcula el A$k$-LAESA y el número de distancias que calcula el $k$-LAESA, y en el caso de la base de datos CROMOSOMAS también se muestran los tiempos medios de clasificación[7]; para no repetir resultados, solamente hemos dibujado las gráficas correspondientes a un conjunto de entrenamiento de 4096 prototipos.

[7] Los tiempos para la base de datos SINTE son muy pequeños y difíciles de medir con precisión.

| | $k=5$ | $k=15$ | $k=25$ |
|---|---|---|---|
| A$k$-LAESA (v3) | 0.28067 | 0.25322 | 0.24578 |
| A$k$-LAESA (v4) | 0.28047 | 0.25302 | 0.24551 |
| $k$-NN | 0.28144 | 0.24932 | 0.2371 |

Cuadro 3.5: Resumen de resultados del A$k$-LAESA (versiones 3 y 4) y $k$-NN (dimensión 10, 8 clases).

| | $k=5$ | $k=15$ | $k=25$ |
|---|---|---|---|
| A$k$-LAESA (v3) | 0.57793 | 0.51779 | 0.50928 |
| A$k$-LAESA (v4) | 0.57783 | 0.51759 | 0.50977 |
| $k$-NN | 0.57862 | 0.50909 | 0.49599 |

Cuadro 3.6: Resumen de resultados del A$k$-LAESA (versiones 3 y 4) y $k$-NN (dimensión 10, 32 clases).

### Resultados con la base de datos SINTE

Como en los epígrafes correspondientes a las versiones anteriores del A$k$-LAESA, en las figuras 3.25 y 3.26 se muestra la evolución de la tasa de error de la versión 4 del A$k$-LAESA según aumenta el valor de $k$, comparándola con la tasa de error de la versión 3 y la de los $k$ vecinos más cercanos. Como ocurre con la versión 3, las diferencias con la tasa de error de los $k$ vecinos más cercanos son mínimas, y las diferencias entre las tasas de las dos versiones del A$k$-LAESA son inapreciables en el dibujo; los cuadros 3.5, 3.6, 3.7 y 3.8 muestran los resultados en tanto por uno para algunos valores de $k$, con 4096 prototipos de entrenamiento.[8]

Aunque en el caso de distancias euclídeas el número de distancias calculadas por un algoritmo no es lo más relevante (influye más el resto del algoritmo), cuando las distancias son caras dicho número de distancias sirve

[8]Al tratarse de valores medios no todas las cifras son significativas, depende del intervalo de confianza.

| | $k=5$ | $k=15$ | $k=25$ |
|---|---|---|---|
| A$k$-LAESA (v3) | 0.103998 | 0.085254 | 0.08037 |
| A$k$-LAESA (v4) | 0.104095 | 0.085254 | 0.08037 |
| $k$-NN | 0.104293 | 0.085156 | 0.08037 |

Cuadro 3.7: Resumen de resultados del A$k$-LAESA (versiones 3 y 4) y $k$-NN (dimensión 20, 8 clases).

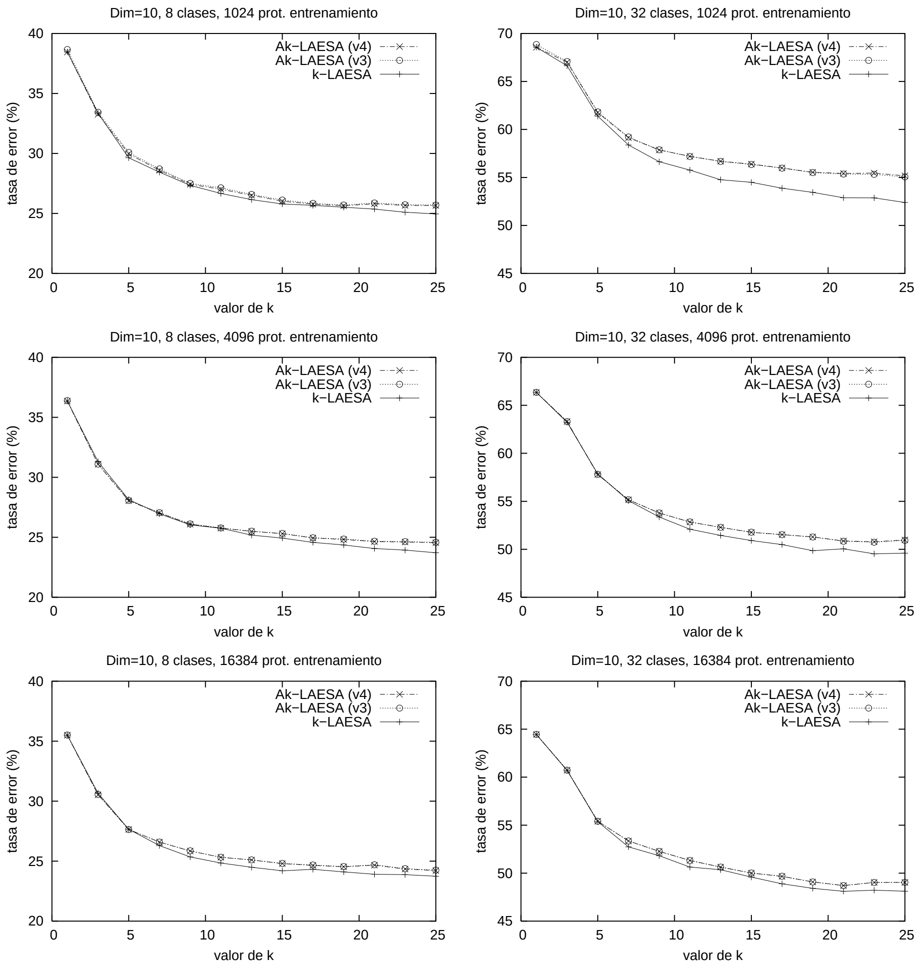


Figura 3.25: Tasas de error del A$k$-LAESA (versiones 3 y 4) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 10.

| | $k = 5$ | $k = 15$ | $k = 25$ |
|---|---|---|---|
| A$k$-LAESA (v3) | 0.40577 | 0.33487 | 0.31162 |
| A$k$-LAESA (v4) | 0.40567 | 0.33497 | 0.31162 |
| $k$-NN | 0.40548 | 0.33487 | 0.31221 |

Cuadro 3.8: Resumen de resultados del A$k$-LAESA (versiones 3 y 4) y $k$-NN (dimensión 20, 32 clases).

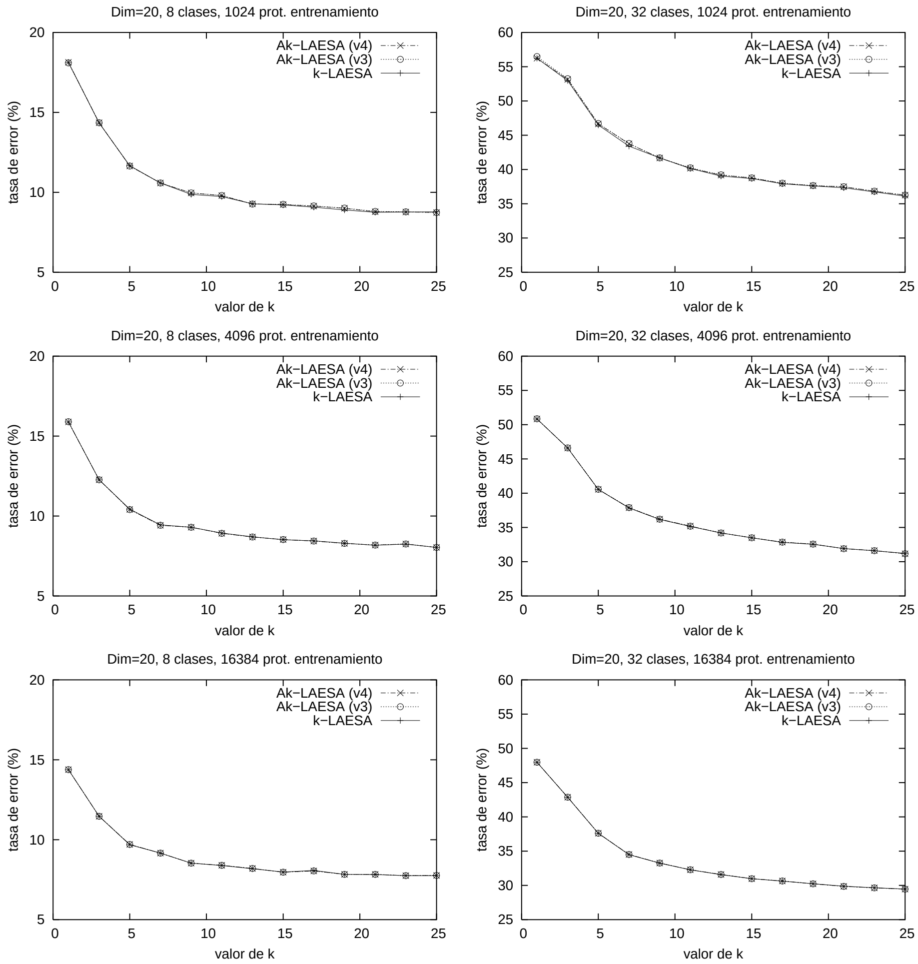


Figura 3.26: Tasas de error del A$k$-LAESA (versiones 3 y 4) en comparación con las del $k$-LAESA, para datos sintéticos de dimensión 20.

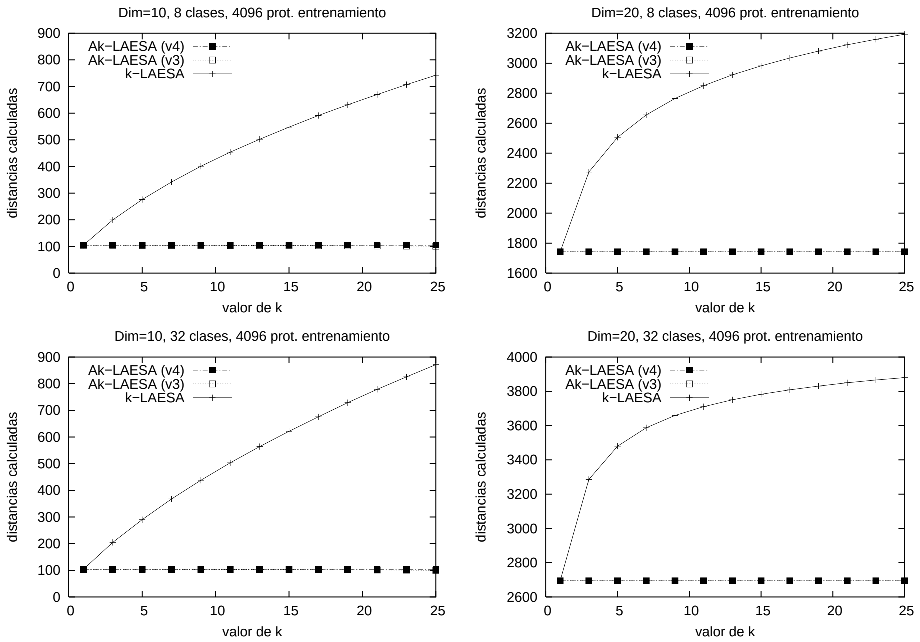


Figura 3.27: Número de distancias calculadas por el A$k$-LAESA (versiones 3 y 4) en comparación con las del $k$-LAESA, para datos sintéticos.

para medir la eficiencia de un clasificador. La figura 3.27 muestra las distancias calculadas por las dos últimas versiones del A$k$-LAESA junto con el número de distancias calculadas por el $k$-LAESA, para conjuntos de entrenamiento de 4096 prototipos. Como se puede apreciar en las gráficas, el número de distancias del A$k$-LAESA no depende del valor de $k$, mientras que en el caso del $k$-LAESA crece considerablemente al aumentar el valor de $k$.

### Resultados con la base de datos CROMOSOMAS

La figura 3.28 muestra las tasas de error, el número medio de distancias calculadas y el tiempo medio de clasificación de las dos últimas versiones del A$k$-LAESA, comparados con los mismos valores del $k$-LAESA (que clasifica utilizando los $k$ vecinos más cercanos). Los resultados son similares a los obtenidos con los datos sintéticos: la tasa de error del A$k$-LAESA es similar a la de los $k$ vecinos más cercanos, pero el número de distancias calculadas del A$k$-LAESA no crece con el valor de $k$ (cosa que sí sucede con los $k$ vecinos más cercanos).

En estas gráficas se puede observar además un resultado curioso: aunque el número de distancias calculadas es prácticamente el mismo en ambas versiones del A$k$-LAESA, el tiempo de clasificación de la versión 3 es superior (incluso al del $k$-LAESA). Esto se debe a que tanto la versión 4 del A$k$-LAESA como el $k$-LAESA están implementados a partir de la versión reformulada del LAESA (véase la figura 3.20), más eficiente. Los tiempos de clasificación de las versiones 1 y 2 del A$k$-LAESA son parecidos a los de la versión 3, aunque inferiores. Por tanto, aunque la distancia empleada para este problema es una distancia costosa en tiempo, no es el único factor que influye en el tiempo de clasificación, también influye el tiempo empleado en la gestión de las estructuras de datos (más sencilla y rápida en la versión 4 que en la versión 3).

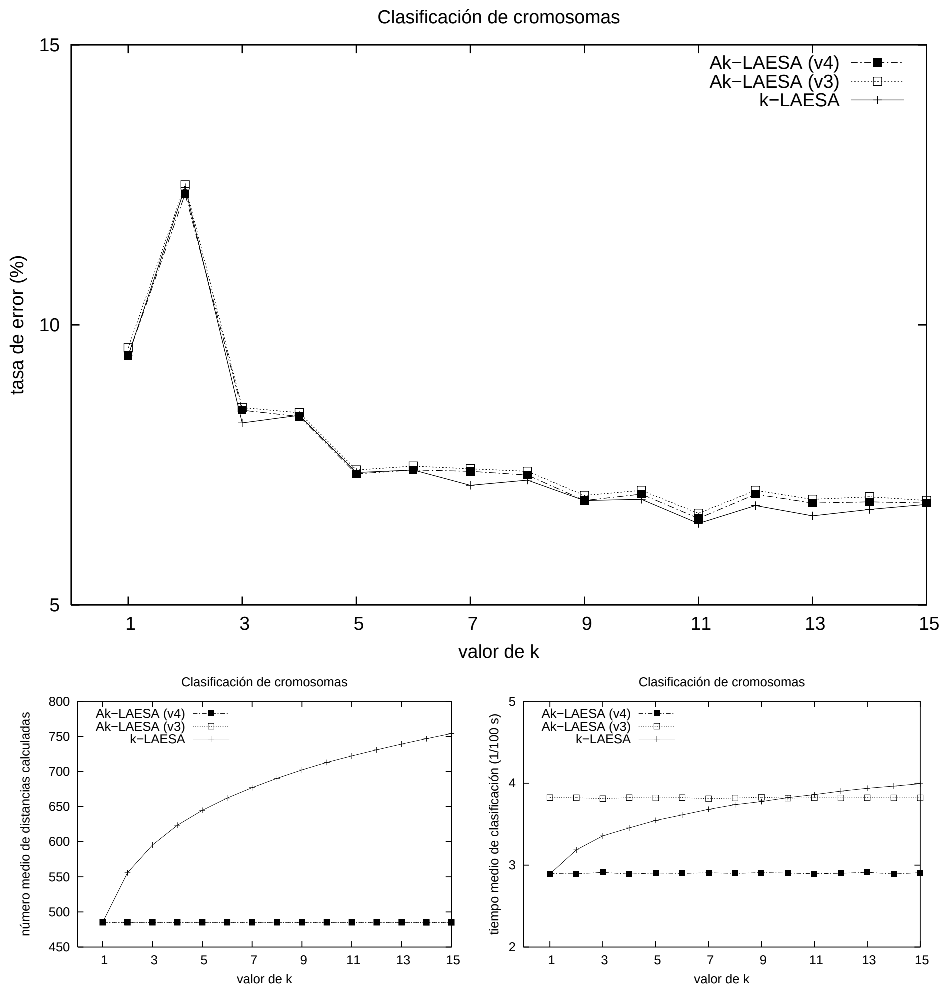


Figura 3.28: Tasa de error, número de distancias y tiempo de clasificación del A$k$-LAESA (versiones 3 y 4), y comparación con el $k$-LAESA en la clasificación de cromosomas.

# Capítulo 4

# La regla de los $k$ vecinos seleccionados más cercanos

Al final del capítulo 3 llegamos a la conclusión de que se puede mejorar considerablemente las tasas de acierto en la clasificación del LAESA utilizando aquellos prototipos que son seleccionados como candidatos a vecino más cercano (versión 4 del A$k$-LAESA). Puesto que son muchos los algoritmos que realizan la búsqueda del vecino más cercano de una manera similar (seleccionando en cada paso un candidato a vecino más cercano y calculando su distancia a la muestra), es posible aplicar la misma idea a todos ellos. En este capítulo se presenta una nueva regla de clasificación basada en los $k$ prototipos (vecinos) más cercanos seleccionados durante la búsqueda del vecino más cercano: los $k$ vecinos seleccionados más cercanos (Moreno-Seco et al., 2003b). Esta regla se ha aplicado a los algoritmos estudiados en el capítulo 2: k-d tree, algoritmo de Fukunaga y Narendra, AESA, TLAESA, vp-tree y GNAT. La regla aplicada al LAESA es exactamente la versión 4 del A$k$-LAESA (capítulo 3).

## 4.1. Aplicabilidad de la regla

En general, la regla de los $k$ vecinos seleccionados más cercanos ($k$-NSN, del inglés *$k$ nearest selected neighbours*) se puede aplicar a todos los algoritmos que encajen en un esquema de búsqueda por aproximación y eliminación (Ramasubramanian y Paliwal, 2000). En la sección 2.2 se describe con detalle dicho esquema, que se podría resumir de esta manera:

- En cada paso del algoritmo, se selecciona un candidato a vecino más cercano mediante una función de *aproximación*, y se calcula su distancia a la muestra, $d$.
- Si dicha distancia $d$ es menor que la del vecino más cercano hasta el momento, se actualiza su valor y se *eliminan* todos aquellos prototipos

del conjunto de entrenamiento que no puedan estar dentro de una hiperesfera de radio $d$ centrada en la muestra; para determinar si un prototipo puede estar o no en la hiperesfera se suele utilizar una cota inferior de su distancia a la muestra.

Para aplicar la regla de los $k$ vecinos seleccionados más cercanos a un algoritmo solamente es necesario realizar dos modificaciones:

1. Cada vez que se calcula una distancia de un prototipo $p$ a la muestra $x$, se inserta el par $(d(x,p),p)$ en la lista de los $k$ vecinos seleccionados más cercanos, si $d(x,p)$ es menor que alguna de las distancias almacenadas. Dicha lista contiene como mucho $k$ prototipos junto con su distancia a la muestra, por lo que la inserción de un nuevo prototipo (si su distancia es menor que alguna de las que ya existen) es una tarea rápida con valores razonables de $k$.

2. Cuando termina la búsqueda, se clasifica la muestra por votación entre las clases de los elementos de dicha lista (como mucho $k$ elementos), en lugar de asignarle la clase del vecino más cercano (que es el primero de los elementos de dicha lista).

La figura 4.1 muestra con una marca en el margen izquierdo las modificaciones necesarias en el esquema de la figura 2.1 para la utilización de la regla de los $k$ vecinos seleccionados más cercanos.

    $d_{nn} := \infty$
$>$ listaNSN := vacía
    **hacer** hasta que el conjunto de entrenamiento $P$ esté vacío
        $p_i := argmin_{p \in P}$Aprox$(x,p)$ // *aproximación*
        $d := d(x,p)$ // *distancia a la muestra*
$>$ Almacenar(listaNSN,$k$,$d(x,p)$,$p$)
        **si** $d < d_{nn}$ **entonces**
            $nn := p$ ; $d_{nn} := d$
            $P := P - \{ q : q \notin E(x, d_{nn}) \}$ // *eliminación*
        **fsi**
    **fhacer**
$>$ claseMuestra := Votación(listaNSN,$k$)

Figura 4.1: La regla de los $k$ vecinos seleccionados más cercanos en el esquema de búsqueda por aproximación y eliminación.

Teniendo en cuenta las modificaciones que se realizan a los algoritmos, la aplicabilidad de la regla se puede extender a otros algoritmos que, aunque no pueda considerarse que utilizan un esquema de aproximación y eliminación, para elegir el vecino más cercano seleccionen varios candidatos y calculen su distancia a la muestra. El k-d tree es un buen ejemplo: es un algoritmo

| NOMBRE | FIGURA | PÁGINA |
|---|---|---|
| k-d tree | 2.3 | 26 |
| Fukunaga y Narendra | 2.6 | 30 |
| vp-tree | 2.10 | 35 |
| GNAT | 2.12 | 39 |
| AESA | 2.15 | 42 |
| LAESA | 3.20 | 78 |
| TLAESA | 2.17 | 46 |

Cuadro 4.1: Algoritmos a los que se ha aplicado la regla $k$-NSN.

difícil de encajar en el esquema de aproximación y eliminación puesto que utiliza las coordenadas para realizar la aproximación y solamente *selecciona* (calcula la distancia) cuando se llega a una hoja, y sin embargo es posible (como veremos a continuación) utilizar la regla de los $k$ vecinos seleccionados más cercanos.

Cuando se extiende un algoritmo rápido de búsqueda del vecino más cercano para encontrar los $k$ vecinos más cercanos (véase el capítulo 2), el número de distancias calculadas por el algoritmo aumenta, ya que la distancia de referencia para terminar la búsqueda es la del $k$-ésimo vecino. Sin embargo, con la regla $k$-NSN estos algoritmos calculan exactamente el mismo número de distancias (la distancia de referencia sigue siendo la del vecino más cercano), y la única tarea adicional es la de almacenar los $k$ vecinos seleccionados más cercanos, cuyo coste temporal es mucho menor (despreciable en la práctica) que el cálculo de varias distancias, especialmente cuando se trata de distancias caras. Esta característica hace que la regla $k$-NSN no necesite más tiempo de clasificación que la regla NN, mientras que la regla $k$-NN sí emplea más tiempo (en experimentos que se describen más adelante en este capítulo se puede comprobar esta afirmación). Además, el tiempo empleado por un algoritmo extendido con la regla $k$-NSN no depende del valor de $k$ (para valores de $k$ razonables), y por tanto puede aumentarse el valor de $k$ todo lo que se desee, por lo que también podría utilizarse esta regla como una aproximación rápida para encontrar el valor óptimo de $k$ en una tarea concreta, independientemente de que después se utilice una regla u otra para la clasificación.

## 4.2. Algoritmos ampliados con la regla $k$-NSN

Los algoritmos que se han extendido con la regla de los $k$-NSN en esta tesis son los que aparecen en el cuadro 4.1. También aparecen en ese cuadro las figuras en las que se describe cada algoritmo, y la página en que aparecen.

En el caso del LAESA, el cuadro 4.1 hace referencia a la versión reformulada del algoritmo que se presenta en el capítulo 3. En todas las figuras está marcado con $(a)$ el punto en el que se calcula la distancia de un candidato a vecino más cercano a la muestra; en ese punto, después de calcular la distancia, habría que almacenarla en la lista de los $k$-NSN (sólo si no es mayor que la del último). Finalmente, todos los algoritmos obtienen el vecino más cercano y posteriormente clasifican la muestra en la misma clase; para la aplicación de la regla $k$-NSN habría que clasificar la muestra en la clase más votada de entre los $k$-NSN.

Las figuras 4.2 y 4.3 contienen un ejemplo de la aplicación de esta regla con diferentes algoritmos, en un caso con 100 prototipos en dimensión 2, con $k = 3$. Los prototipos del conjunto de entrenamiento están marcados con un $+$, y la muestra con el símbolo $\times$ (más grande); los prototipos seleccionados por cada algoritmo están marcados con un círculo, y aquellos que se utilizan en la votación de la regla $k$-NSN están marcados con un círculo más grande. Finalmente, los $k$ vecinos más cercanos están marcados con un cuadrado grande. Como se puede observar, en varios casos los prototipos utilizados por la regla $k$-NSN y la regla $k$-NN son exactamente los mismos; en los algoritmos de la familia AESA no todos coinciden porque calculan muy pocas distancias, es decir, seleccionan muy pocos prototipos (en el caso de LAESA y TLAESA, solamente se selecciona un prototipo).

## 4.3. Experimentos y resultados

En esta sección se presentan los experimentos realizados con la regla $k$-NSN para comprobar su comportamiento según evolucionan distintos parámetros: tamaño del conjunto de entrenamiento, valor de $k$ y dimensionalidad de los datos. En todos los experimentos, las referencias son siempre la regla del vecino más cercano y la regla de los $k$ vecinos más cercanos.

Los experimentos se han realizado con todas las bases de datos del apéndice A, comparando las tasas de error de la regla de los $k$-NSN con la de la regla de los $k$-NN, con todos los algoritmos del cuadro 4.1, excepto el k-d tree en aquellos casos en que los prototipos no se representan como vectores y el AESA en algunos en los que el conjunto de entrenamiento es demasiado grande. También se ha probado la regla $k$-NSN con otros conjuntos de datos en los que la utilización de $k$ vecinos no mejora o mejora muy poco las tasas de acierto en la clasificación, como por ejemplo con dígitos manuscritos codificados como cadenas (para más detalles acerca de la codificación y la distancia empleada se puede consultar (Gómez et al., 1995; Micó y Oncina, 1998)); la figura 4.4 muestra las tasas de error de ambas reglas en la clasificación de dígitos manuscritos, usando conjuntos de entrenamiento con aproximadamente 8000 prototipos y conjuntos de test de aproximadamente 1000 prototipos. Solamente se muestran los resultados de ambas reglas con el vp-tree por mantener la claridad de la gráfica (con otros

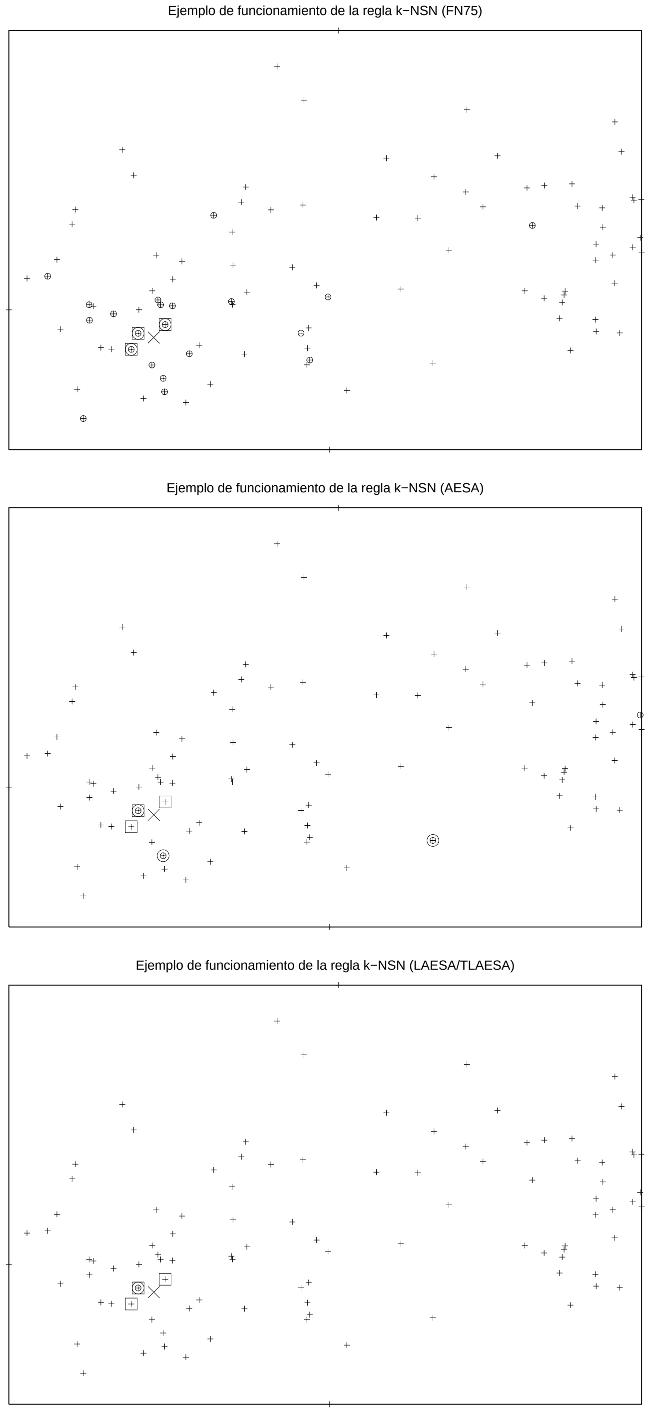


Figura 4.2: Ejemplo en dos dimensiones de la regla $k$-NSN, con $k = 3$. La muestra está marcada con $\times$, los prototipos seleccionados con un círculo pequeño y los utilizados por la regla $k$-NSN con un círculo grande. Los 3 vecinos más cercanos se han marcado con un cuadrado grande.

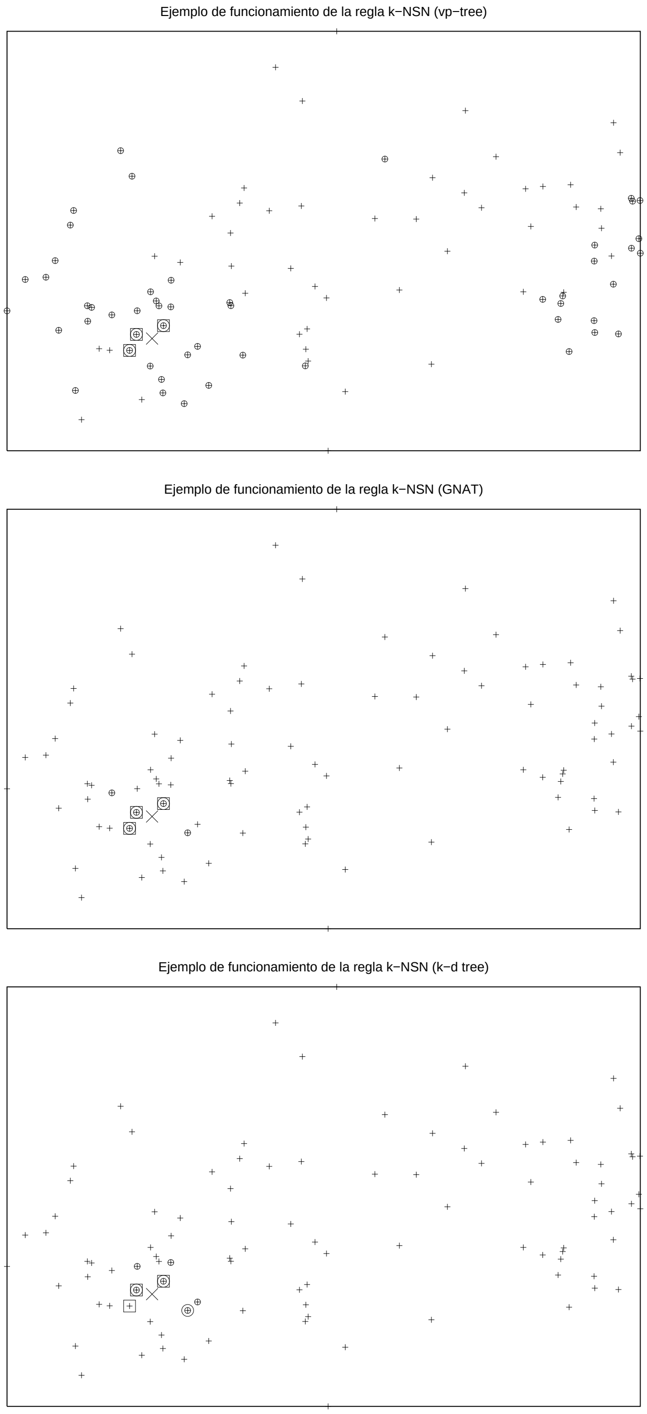


Figura 4.3: Ejemplo en dos dimensiones de la regla $k$-NSN (2).

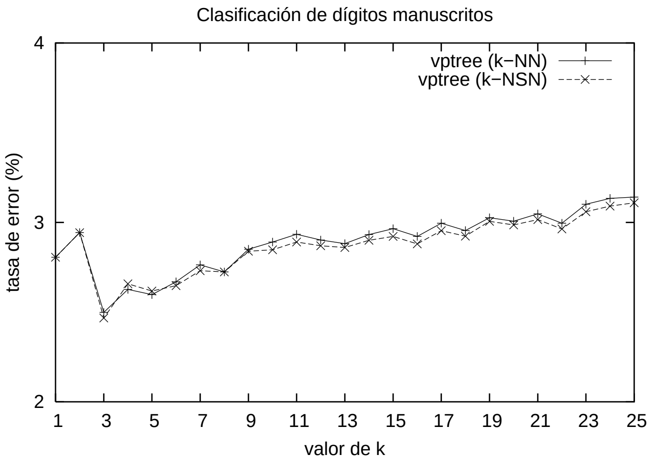


Figura 4.4: Tasa de error de la regla $k$-NSN con respecto a la regla $k$-NN en la clasificación de dígitos manuscritos.

algoritmos el comportamiento es similar), y como se puede observar la regla $k$-NSN obtiene tasas de error similares a las de la regla $k$-NN.

### 4.3.1. Resultados con la base de datos SINTE

Con esta base de datos se han realizado dos conjuntos de experimentos: en el primer conjunto se ha probado el comportamiento de la regla $k$-NSN con datos de dimensiones 10 y 20, y varios tamaños del conjunto de entrenamiento, con el objeto de estudiar la tasa de error según aumenta el valor de $k$, comparándola en todos los casos con la tasa de error de la regla $k$-NN. En las figuras 4.5 y 4.6 se muestran los resultados de la regla $k$-NSN aplicada en varios algoritmos, y como se puede observar, las tasas de error de la regla $k$-NSN son en general muy parecidas a las de la regla $k$-NN, y son prácticamente idénticas cuando aumenta la dimensión; además, al aumentar el tamaño del conjunto de entrenamiento los resultados mejoran, como sucede en la regla $k$-NN. En las gráficas se han dibujado las barras de error correspondientes a la regla $k$-NN, que representan un intervalo de confianza del 95 %. Las tasas de error de la regla $k$-NSN se solapan claramente con la de la regla $k$-NN; solamente en el caso de dimensión 10 y con los algoritmos de la familia AESA se salen del intervalo de confianza de la regla $k$-NN, pero si dibujáramos las barras de error también para la regla $k$-NSN (no se ha hecho por claridad), se observaría que existe un solapamiento, lo cual implica que los resultados son claramente similares.

La gran ventaja de la regla $k$-NSN sobre la regla $k$-NN es que obtiene unas tasas de error similares, pero calculando un número de distancias que no depende del valor que se elija para $k$, como se puede observar en la figura 4.7,

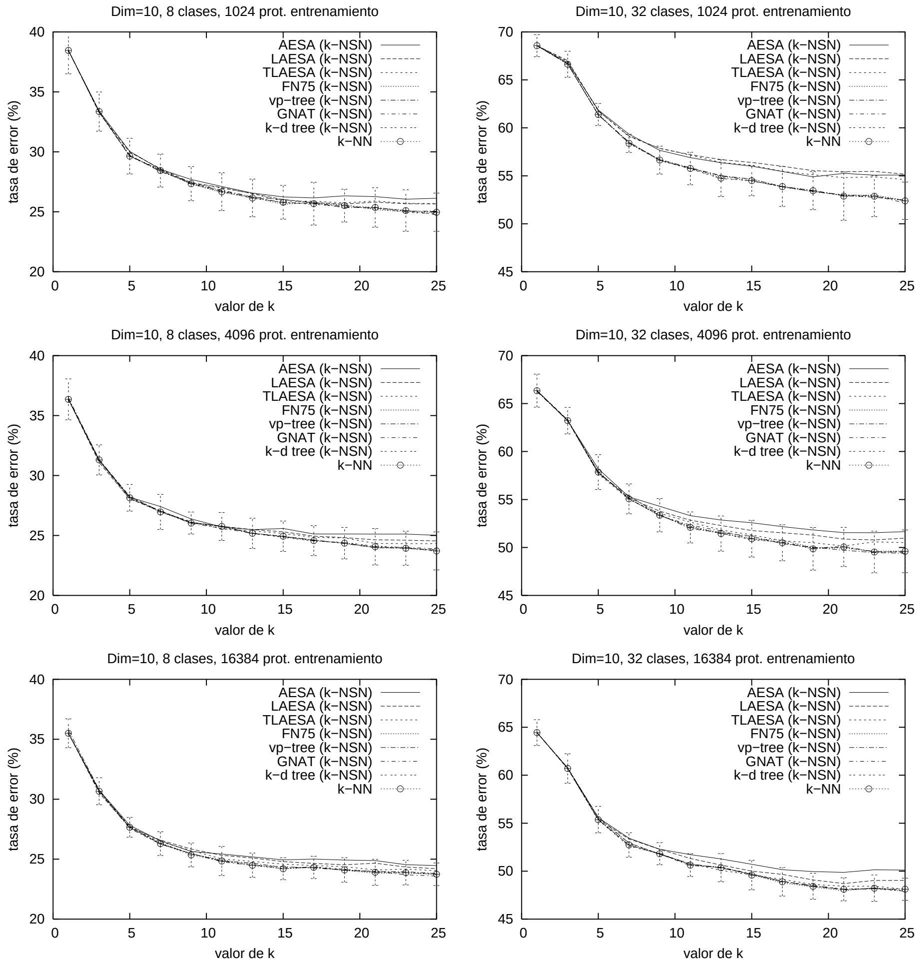


Figura 4.5: Tasas de error de la regla $k$-NSN en comparación con la de la regla $k$-NN, para datos sintéticos de dimensión 10 y valores crecientes de $k$. Las gráficas de la izquierda corresponden a datos de 8 clases, y las de la derecha a 32 clases. Las gráficas aparecen por orden creciente del tamaño del conjunto de entrenamiento (1024, 4096 y 16384).

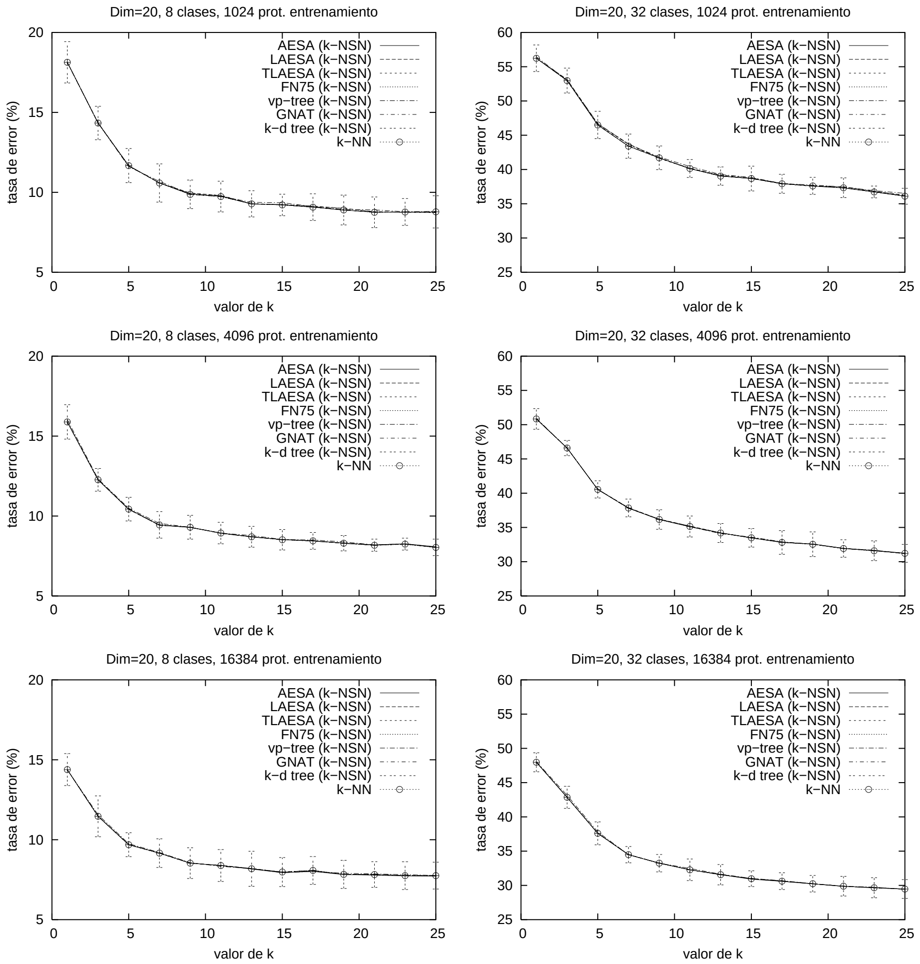


Figura 4.6: Tasas de error de la regla $k$-NSN en comparación con la de la regla $k$-NN, para datos sintéticos de dimensión 20 y valores crecientes de $k$.

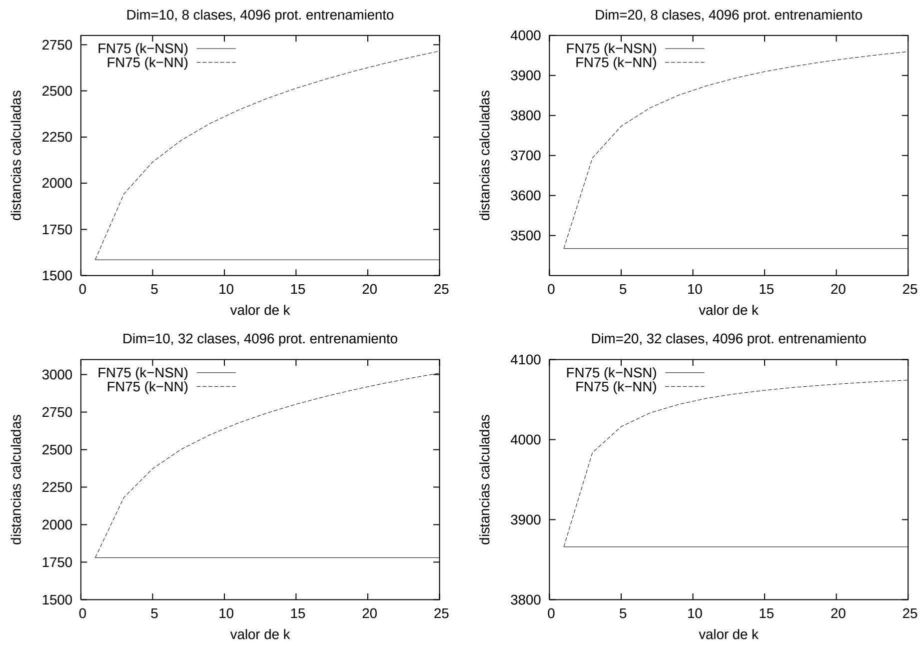


Figura 4.7: Número de distancias calculadas por el algoritmo de Fukunaga y Narendra con las reglas $k$-NN y $k$-NSN, con 4096 prototipos de entrenamiento.

que muestra el número de distancias calculadas por ambas reglas, para un conjunto de entrenamiento de 4096 prototipos. Se muestran únicamente los resultados para el algoritmo de Fukunaga y Narendra, abreviado como FN75, por claridad, ya que los resultados con otros algoritmos son parecidos en el sentido de que el incremento del valor de $k$ no conlleva un aumento en el número de distancias calculadas por la regla $k$-NSN. También se puede observar que el número de distancias que calcula la regla $k$-NN crece con el valor de $k$ (hasta llegar prácticamente al máximo, 4096); teniendo en cuenta que las tasas de error de ambas reglas son similares, la aplicación de la regla $k$-NSN resulta muy aconsejable cuando el cálculo de la distancia es costoso y es el principal factor en el tiempo de clasificación.

Por otro lado, para comprobar el comportamiento de la regla $k$-NSN con valores muy grandes de $k$ se realizó un segundo conjunto de experimentos, cuyos resultados se muestran en la figuras 4.8. Como sucede con la regla $k$-NN, la tasa de error de la regla $k$-NSN empieza a oscilar a partir de un cierto valor de $k$, sin mejorar ni empeorar. Es importante destacar que el valor de $k$ máximo que utiliza la regla $k$-NSN coincide con el número de distancias calculadas por el algoritmo que se utilice para la búsqueda del vecino más cercano, independientemente del valor de $k$ fijado previamente; esta es la causa del mal comportamiento de algunos algoritmos de la familia

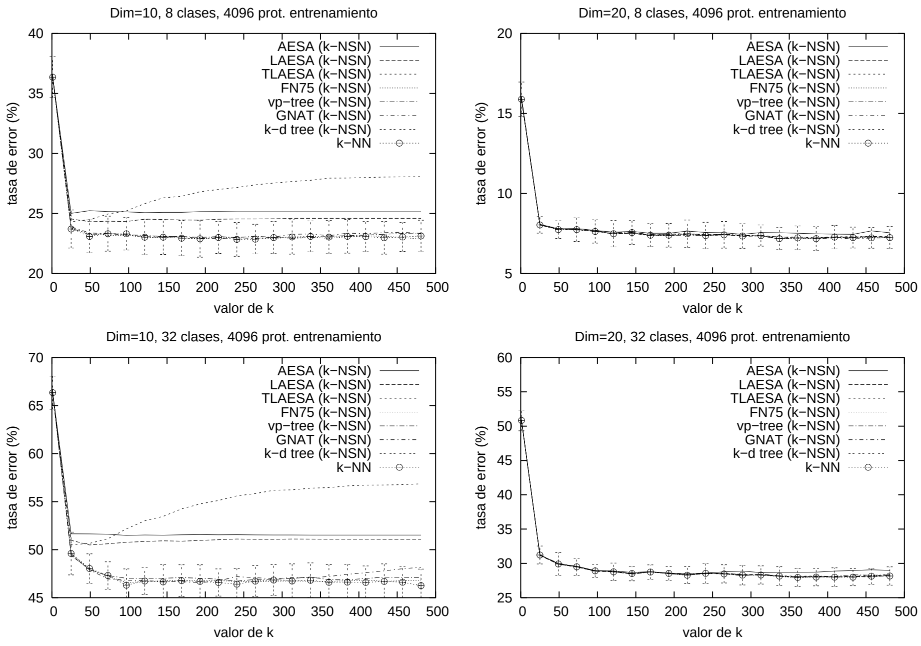


Figura 4.8: Tasas de error de las reglas $k$-NN y $k$-NSN para valores grandes de $k$, con 4096 prototipos de entrenamiento.

AESA para valores grandes de $k$ en dimensión 10, ya que el número medio de distancias que calculan no llega a 50, y entre los prototipos seleccionados se encuentran algunos que no están muy cerca de la muestra.

### 4.3.2. Resultados con la base de datos CROMOSOMAS

En la tarea de clasificación de cromosomas (véase el apéndice A para una descripción de la base de datos) es en la que mejor se aprecian las ventajas de la regla $k$-NSN con respecto a la regla $k$-NN. Como se puede observar en la figura 4.9, las tasas de error de los distintos algoritmos con la regla $k$-NSN son muy parecidas a la tasa de error de la regla $k$-NN; la diferencia más apreciable se produce en los algoritmos de la familia AESA, en los que el número de distancias (y por tanto de prototipos seleccionados) es menor que en el resto de algoritmos, lo cual hace que se utilicen en la votación vecinos no muy cercanos a la muestra. Además, como sucede con los datos sintéticos, el número de distancias y el tiempo de clasificación de la regla $k$-NSN no aumentan con el valor de $k$, como sí sucede con la regla $k$-NN (figuras 4.10 y 4.11).

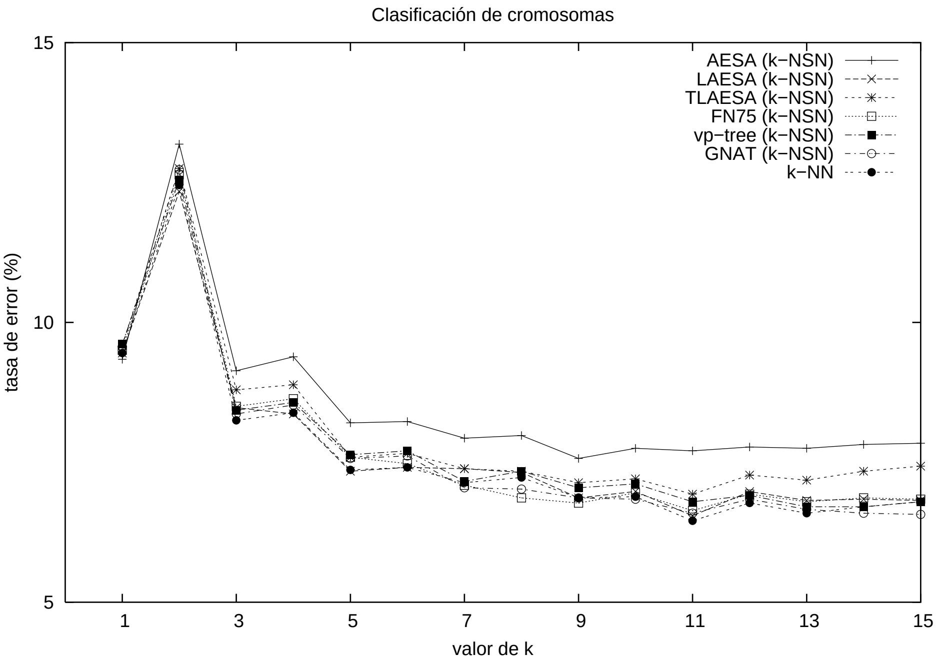


Figura 4.9: Tasa de error de la regla $k$-NSN con respecto a la regla $k$-NN en la clasificación de cromosomas.

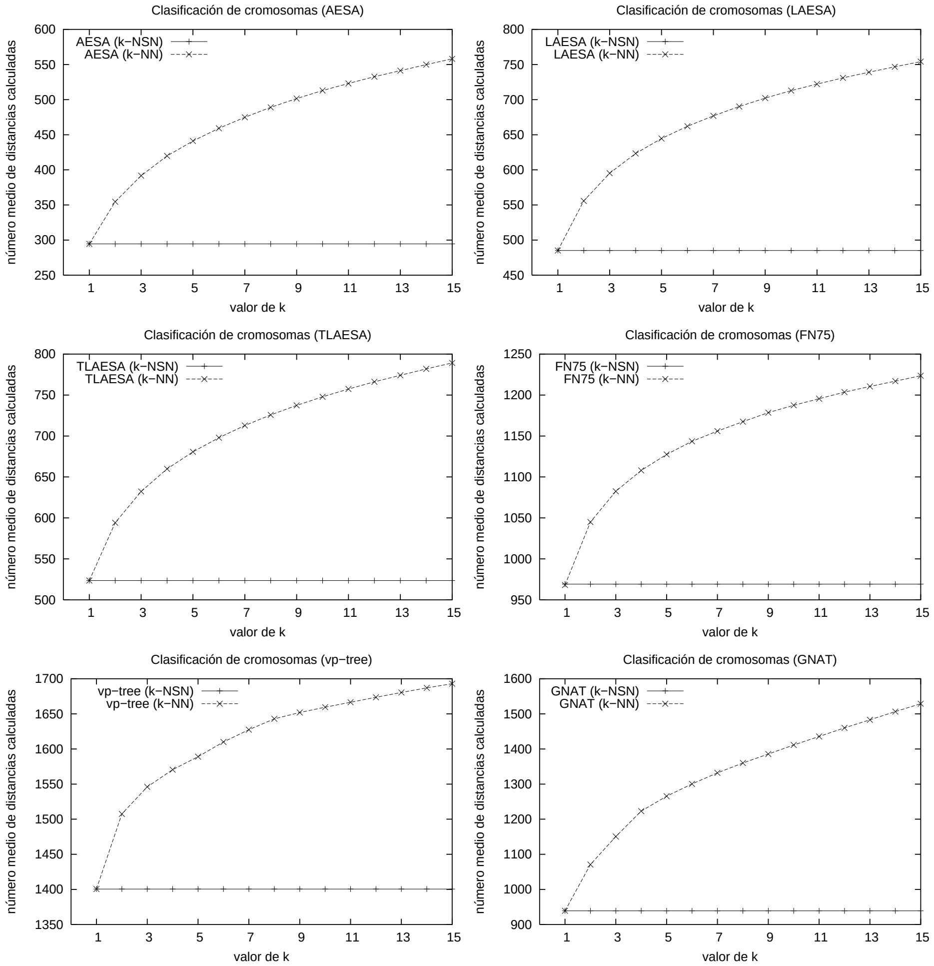


Figura 4.10: Comparación entre el número de distancias de las reglas $k$-NSN y $k$-NN en la clasificación de cromosomas, utilizando diferentes algoritmos.

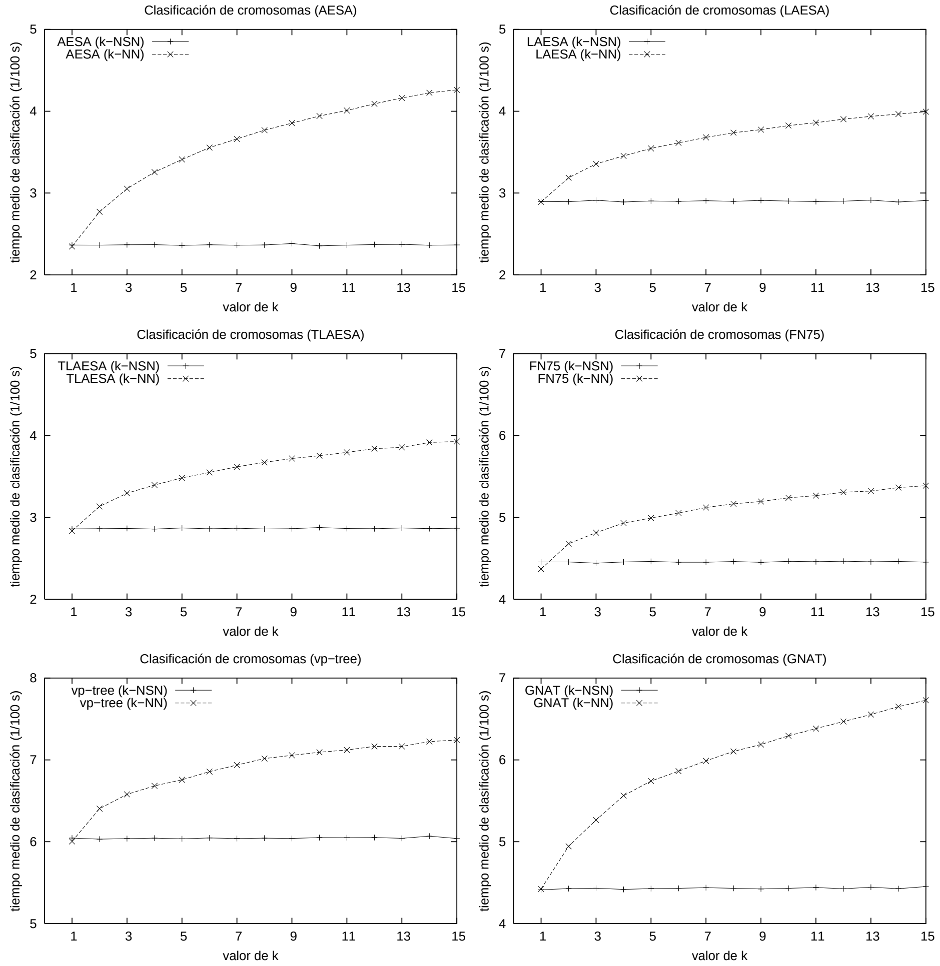


Figura 4.11: Comparación entre el tiempo medio de clasificación por muestra en la clasificación de cromosomas, utilizando diferentes algoritmos.

### 4.3.3. Resultados con la base de datos UCI

De todas las bases de datos presentes en el conjunto de bases de datos para el aprendizaje automático de la U.C.I. (Blake y Merz, 1998), las únicas que se han utilizado para realizar experimentos son aquellas en las que la utilización de $k$ vecinos mejoraba de manera apreciable las tasas de error, ya que es en estos casos en los que la regla $k$-NSN es aplicable, puesto que cuando no se mejoran los resultados de la regla del vecino más cercano no tiene sentido utilizar reglas que usen $k$ vecinos. Las bases de datos utilizadas son: `heart`, `liver-disorders`, `pima-diabetes` y `wdbc`; en el apéndice A se describe con más detalle cada base de datos. En estos conjuntos los prototipos se representan como vectores, por lo que la distancia que se ha empleado ha sido la distancia euclídea. Además, al tratarse de conjuntos de datos pequeños (768 prototipos tiene el conjunto más grande), para calcular el error se ha utilizado la técnica *leave-one-out*. Se ha realizado un único experimento para cada conjunto de datos, comparando las tasas de error obtenidas por la regla $k$-NSN con la de la regla $k$-NN, para valores crecientes de $k$. La figura 4.12 muestra los resultados obtenidos; lo más destacable es el mal comportamiento de la regla $k$-NSN con los algoritmos de la familia AESA, debido al reducido número de distancias que calculan con unos conjuntos de datos tan pequeños. Por ejemplo, con la base de datos `wdbc` el AESA solamente calcula unas 6 distancias como media, por lo que es razonable que a partir de $k = 1$ sus resultados sean malos. Obsérvese además, que a partir de $k = 7$ no varían, porque la regla $k$-NSN utiliza como máximo $k$ prototipos, pero si en la búsqueda del vecino más cercano no se seleccionan más que $m$ prototipos, en la votación se utilizan solamente $m$, aunque $k$ sea mayor que $m$.

En general, con conjuntos pequeños de datos en los que el vecino más cercano se encuentra en pocos pasos, la regla $k$-NSN no funciona tan bien como cuando los conjuntos de datos tienen un tamaño mayor. Por otro lado, la variabilidad tan alta de los resultados al cambiar el valor de $k$ parece indicar que se necesitan más datos para que la regla $k$-NN (y la $k$-NSN) funcionen mejor. En estos casos, la regla de los vecinos de centroide más cercano, $k$-NCN (Sánchez et al., 2000), obtiene mejores resultados.

### 4.3.4. Resultados con la base de datos PHONEME

En esta base de datos los prototipos se representan como vectores, y del conjunto inicial se han obtenido 5 particiones para entrenamiento/test (los detalles se pueden consultar en el apéndice A). Se han realizado experimentos con cada par de conjuntos entrenamiento/test (en total 5 experimentos), comparando las tasas de error de la regla $k$-NSN aplicada a diferentes algoritmos con la tasa de error de la regla $k$-NN, para valores crecientes de $k$. La figura 4.13 muestra los resultados obtenidos, en los que se aprecia que los

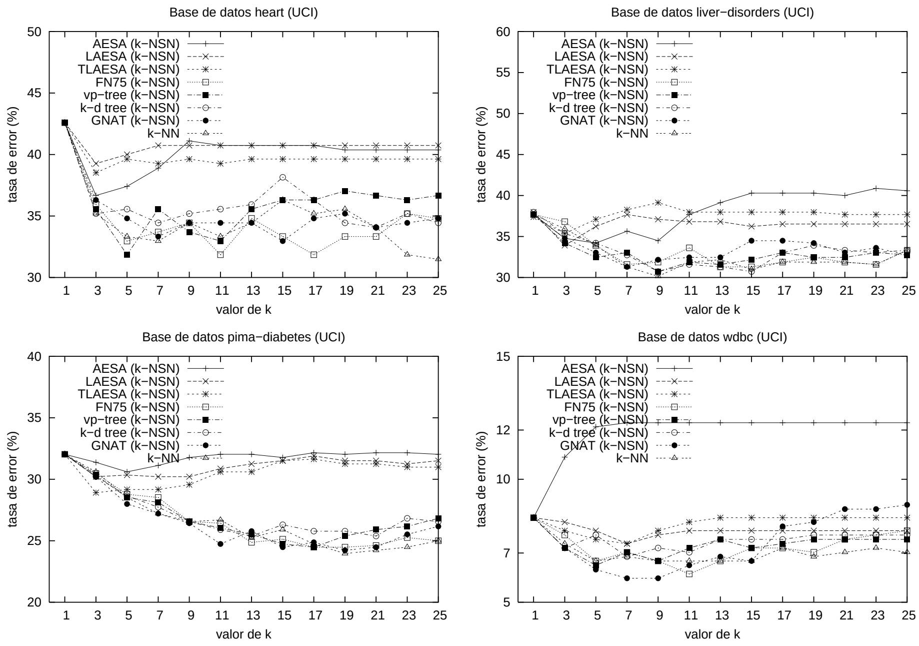


Figura 4.12: Resultados de la regla $k$-NSN con la base de datos UCI.

algoritmos que menos distancias calculan (el AESA y el LAESA) obtienen peores tasas de acierto que los demás con la regla $k$-NSN, pero en otros algoritmos como el de Fukunaga y Narendra los resultados de la regla $k$-NSN son similares y en algún caso mejores que los de la regla $k$-NN.

## 4.4. Estudio de coincidencias entre los $k$-NSN y los $k$-NN

La escasa diferencia entre las tasas de error de las reglas $k$-NSN y $k$-NN nos ha llevado a conjeturar que probablemente muchos de los $k$-NSN se encontraban entre los $k$-NN. Para comprobar esta hipótesis se han realizado varios experimentos para calcular el porcentaje de los $k$-NSN que se encuentran entre los $k$-NN, y cómo evoluciona dicho porcentaje en función de la dimensión y el tamaño del conjunto de entrenamiento. Además, se ha analizado la relación entre la distancia al $k$-ésimo vecino seleccionado más cercano y la distancia al $k$-ésimo vecino más cercano.

### 4.4.1. Estudio según aumenta la dimensión

El primer conjunto de experimentos trata de comprobar el grado de coincidencia entre los $k$-NSN y los $k$-NN según aumenta la dimensión de

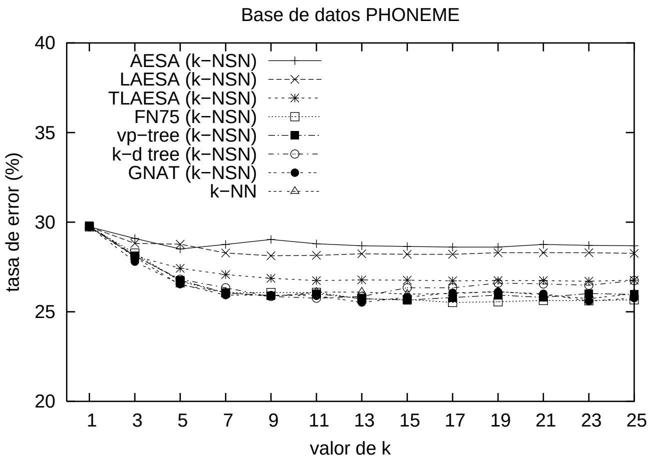


Figura 4.13: Tasa de error de la regla $k$-NSN con respecto a la regla $k$-NN con la base de datos PHONEME.

los datos, para un valor de $k$ y un tamaño del conjunto de entrenamiento fijos, concretamente para $k = 15$ y 4096 prototipos de entrenamiento. Los conjuntos de datos son los de la base de datos SINTE (con dimensiones 10 y 20, procedentes de 8 y 32 clases), ampliados con datos de dimensiones 5, 30, 40 y 50, generados con el mismo algoritmo y los mismos parámetros. Para cada dimensión se han realizado experimentos con 10 pares de conjuntos de entrenamiento/test de 4096 y 1024 prototipos respectivamente, calculando el porcentaje de coincidencia, y la relación entre las distancias del último $k$-NSN y el último $k$-NN.

La figura 4.14 muestra los resultados, en los que se puede observar que conforme aumenta la dimensión el porcentaje de coincidencia entre los $k$-NSN y los $k$-NN aumenta hasta llegar al 100 % en todos los algoritmos excepto en el LAESA y TLAESA[1]; en cualquier caso, el porcentaje en estos dos algoritmos tiende al 100 %, y en caso de seguir aumentando la dimensión muy probablemente se alcanzaría dicho porcentaje.

Aunque el porcentaje de coincidencia es un buen indicador, la relación entre la distancia del último de los $k$-NSN y la del $k$-ésimo vecino más cercano es posiblemente un mejor indicador, ya que puede suceder que el $k$-ésimo vecino más cercano y el $(k + 1)$-ésimo se encuentren a la misma distancia de la muestra. La figura 4.15 muestra la evolución de esta relación ($d(k - \mathrm{NSN})/d(k - \mathrm{NN})$) según aumenta la dimensión; como se puede observar, la relación tiende a 1 (están a la misma distancia) en cuanto au-

[1]En el caso de estos algoritmos se ha fijado arbitrariamente el número de prototipos base en 260. La búsqueda del valor más adecuado para cada dimensión tendría que haberse hecho de forma exhaustiva y hubiera disparado el tiempo necesario para realizar el experimento.

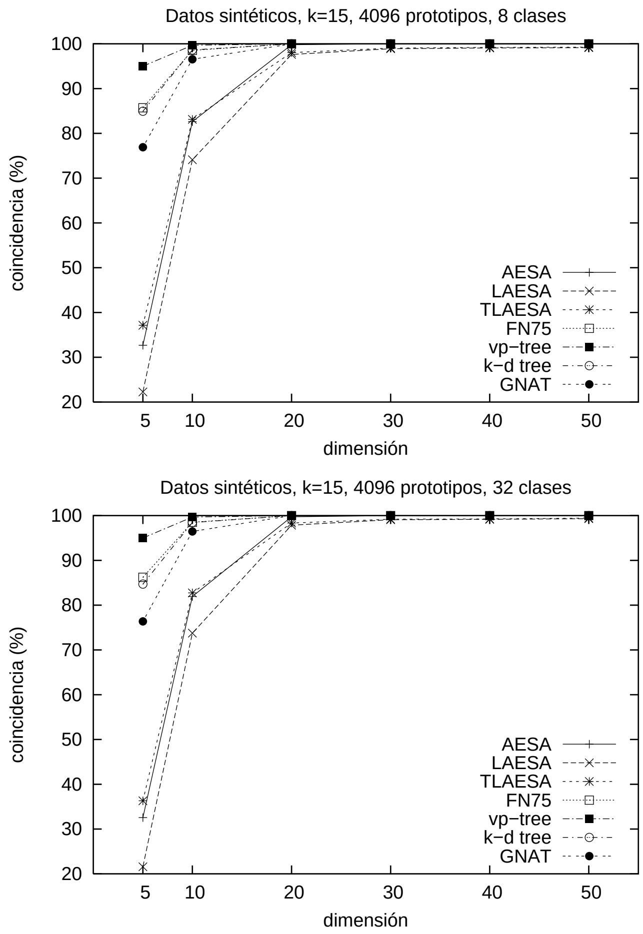


Figura 4.14: Porcentaje de coincidencia entre los $k$-NSN y los $k$-NN según aumenta la dimensión.

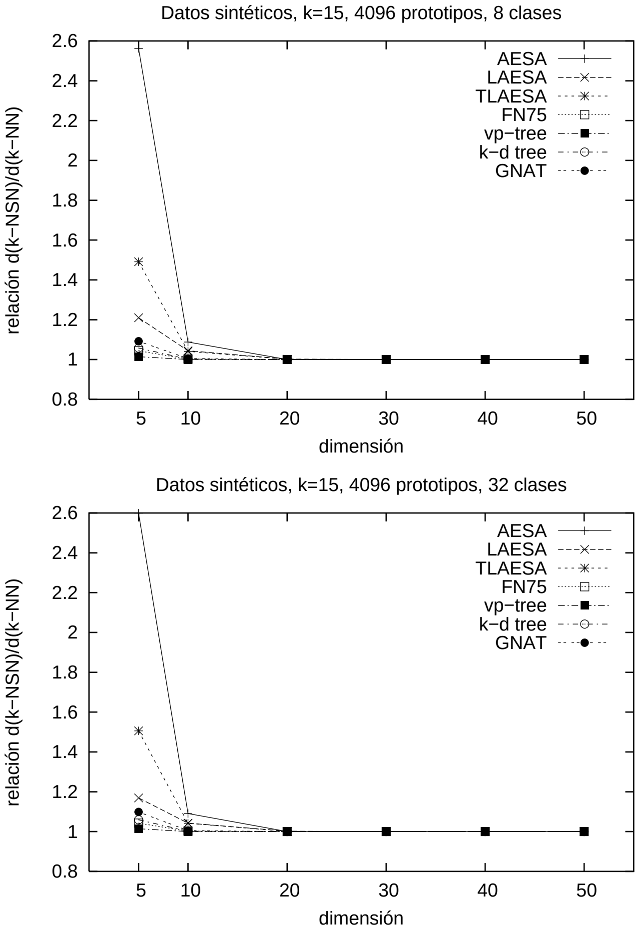


Figura 4.15: Relación entre la distancia del último $k$-NSN y el último $k$-NN, según aumenta la dimensión.

menta la dimensión. Según el algoritmo, tiende más o menos rápidamente; nuevamente, los de la familia AESA son los que más lentamente tienden a 1.

### 4.4.2. Estudio según aumenta el tamaño del conjunto de entrenamiento

La hipótesis en la que se basa este estudio es que al aumentar considerablemente el tamaño del conjunto de entrenamiento, los $k$-NSN coinciden exactamente con los $k$-NN. En caso de comprobarse esta condición, se podría haber afirmado que la regla $k$-NSN tiene las mismas propiedades asintóticas (su tasa de error está acotada por el doble del error de Bayes) que la regla $k$-NN. Las gráficas de las figuras 4.16 y 4.17 muestran los resultados de un

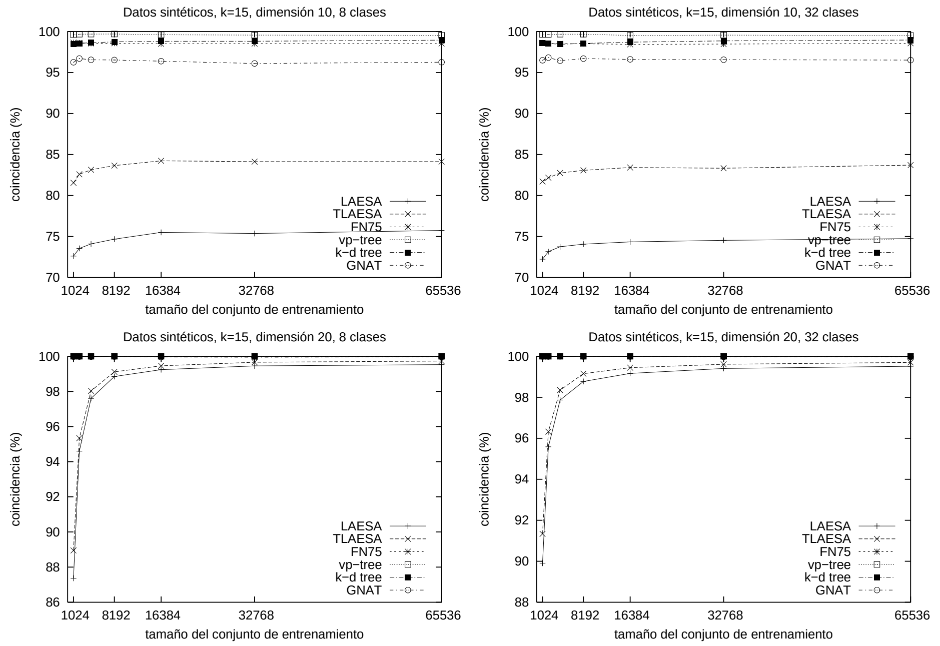


Figura 4.16: Porcentaje de coincidencia entre los $k$-NSN y los $k$-NN, según aumenta el tamaño del conjunto de entrenamiento.

experimento realizado con la base de datos SINTE, de la cual se han extraído conjuntos de entrenamiento de 1024, 2048, 4096, 8192, 16384, 32768 y 65536 prototipos.[2] Estos resultados desmienten la hipótesis inicial: al aumentar el tamaño del conjunto de entrenamiento los porcentajes de coincidencia aumentan, pero a partir de cierto valor se estabilizan y no aumentan más. Lo mismo sucede con la relación entre las distancias del $k$-ésimo vecino seleccionado más cercano y el $k$-ésimo vecino más cercano.

[2]Esta prueba no ha podido realizarse con el algoritmo AESA debido a que no es posible almacenar en memoria la matriz cuadrada de distancias entre todos los prototipos de entrenamiento, pero se podría haber simulado haciendo que cada vez que se desea consultar la matriz se calculase la distancia entre los prototipos. El tiempo que hubiera requerido hacer el experimento de esta forma seguramente habría resultado excesivo, y los resultados de los otros algoritmos son suficientes para extraer conclusiones.

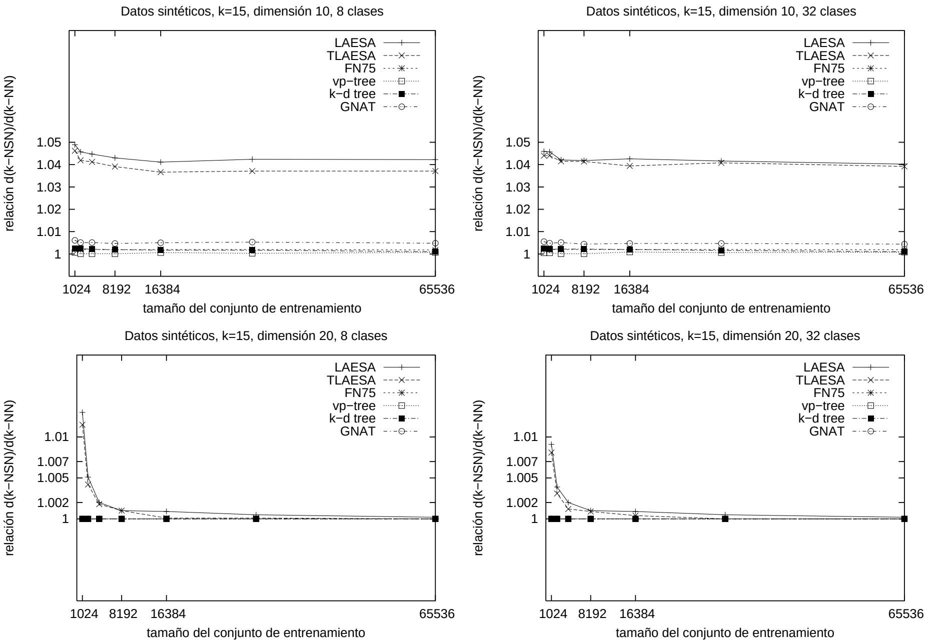


Figura 4.17: Relación entre la distancia del último $k$-NSN y el último $k$-NN, según aumenta el tamaño del conjunto de entrenamiento.

# Capítulo 5

# Búsqueda aproximada del vecino más cercano

En muchas tareas reales en las que la distancia es muy costosa o bien el conjunto de entrenamiento es muy grande, un clasificador que utilice exactamente el vecino más cercano (aunque la búsqueda sea muy eficiente) no es lo suficientemente rápido. Por este motivo, en los últimos años se han desarrollado algoritmos que obtienen el vecino *aproximadamente* más cercano (*approximate nearest neighbour* en inglés), con el fin de acelerar la clasificación, aún a costa de una pequeña pérdida en lo referente a las tasas de acierto en la clasificación. El trabajo más citado en este sentido es el de Arya y Mount (Arya et al., 1998), pero existen otros muchos (Indyk y Motwani, 1998; Clarkson, 1999; Kleinberg, 1997).

La gran mayoría de las propuestas actuales para la búsqueda aproximada del vecino más cercano están basadas en espacios de vectores, en los que el problema más habitual suele ser el tamaño del conjunto de entrenamiento o la alta dimensionalidad de los datos. Sin embargo, en el caso de un espacio métrico general con una distancia muy cara (como por ejemplo la distancia de edición de árboles), las técnicas comentadas anteriormente no son aplicables. Sin embargo, una de las ideas de Arya y Mount (almacenar los nodos de un k-d tree pendientes de visitar en una cola de prioridad, (Mount y Arya, 1997)) se puede aplicar a otros algoritmos válidos para espacios métricos generales como el de Fukunaga y Narendra (Moreno-Seco et al., 2003a) y el TLAESA.

En este capítulo se exponen las modificaciones necesarias en algunos algoritmos para realizar búsquedas aproximadas del vecino más cercano en espacios métricos generales, y los resultados obtenidos con tareas reales.

## 5.1. Búsqueda aproximada usando una cola de prioridad

En los algoritmos basados en árboles como el de Fukunaga y Narendra o el TLAESA, cuando se visita un nodo se elige el hijo por el que debe continuar la búsqueda (normalmente implementada de forma recursiva); la exploración del otro (u otros) hijo queda aplazada hasta que termine la búsqueda en el hijo elegido en primer lugar. En muchas ocasiones, cuando se va a explorar el hijo pendiente de visitar ya se ha encontrado el vecino más cercano y dicha exploración ya no es necesaria.

Otra posible implementación de la búsqueda consiste en almacenar el hijo pendiente de visitar en una cola de prioridad ordenada según una clave que en el caso de estos algoritmos debe ser una cota inferior de la distancia del subarbol a la muestra desconocida, aunque otra posibilidad es almacenar ambos hijos, que es la que se ha utilizado en esta tesis (véase el esquema de la figura 5.1). Por tanto, en cada paso del algoritmo se extrae el nodo de la cola de prioridad con menor cota inferior, se examina el nodo (calculando o no la distancia del representante del nodo a la muestra), se podan aquellos hijos que sea posible podar y se almacenan los hijos no podados en la cola de prioridad; cuando la clave del nodo extraído de la cola (que es una cota inferior de la distancia del nodo a la muestra) es mayor que la distancia al vecino más cercano hasta el momento, la búsqueda ha terminado puesto que en la cola no puede existir un nodo que contenga un vecino más cercano que el actual.

```
dnn := ∞
InsertarCola(raíz,0)                        // inserción de la raíz del árbol
finbusqueda := falso
mientras no finbusqueda y no ColaVacia() hacer
    (nodo,clave) := ExtrMinCola()
    si clave > dnn entonces                 // condición de finalización
        finbusqueda := cierto
    si no
        para todo h = Hijo(nodo) hacer
            si NoEliminable(h) entonces
                nClave := CotaInf(h,muestra)    // cálculo de la nueva clave
                InsertarCola(h,nClave)
            fsi
        fpara
    fsi
fmientras
```

Figura 5.1: Esquema de búsqueda con cola de prioridad.

Este esquema permite la búsqueda aproximada de forma sencilla introduciendo un potenciómetro (denominado habitualmente $\epsilon$), que se debe ajustar en cada tarea real, en la condición de finalización:

**si** $(1+\epsilon) \times \text{clave} > d_{nn}$ **entonces**

Con esta nueva condición, para $\epsilon > 0$ se consigue terminar la búsqueda antes, con la consiguiente ganancia en tiempo de clasificación. Sin embargo, el hecho de no utilizar exactamente el vecino más cercano produce normalmente peores tasas de acierto en la clasificación, por lo que la elección del valor de $\epsilon$ debe hacerse teniendo en cuenta ambos factores.

## 5.2. Búsqueda aproximada con el algoritmo de Fukunaga y Narendra

El algoritmo de Fukunaga y Narendra (Fukunaga y Narendra, 1975) se describe con detalle en la sección 2.3.2, y en la figura 2.6 se muestra una formulación recursiva del mismo. Como se explica en la sección anterior, para permitir la búsqueda aproximada es necesario replantear el algoritmo de manera que utilice una cola de prioridad (véase la figura 5.2), en la que la clave que sirve para ordenar los elementos en la cola es una cota inferior (muy pesimista en este caso) de la distancia de los prototipos contenidos en el nodo a la muestra, cuya expresión se obtiene a partir de la condición de eliminación de un nodo $p$ con representante $M_p$ y radio $R_p$:

**si** $d(x, M_p) > d_{nn} + R_p$ **entonces** *podar nodo*

La expresión $d(x, M_p) - R_p$ se puede considerar una cota inferior de la distancia de cualquier prototipo contenido en el nodo $p$ a la muestra $x$, ya que el radio $R_p$ es la distancia entre los dos prototipos de $p$ más alejados entre sí.

La única modificación necesaria para la búsqueda aproximada en la figura 5.2 sería cambiar la condición marcada con $(a)$ por la siguiente:

**si** $(1+\epsilon) \times m > d_{nn}$ **entonces**

donde $m$ es la clave del nodo que se acaba de extraer (la mínima de entre todas las claves contenidas en la cola). Al introducir el factor $(1+\epsilon)$ es posible que los nodos que todavía no se han explorado contengan al vecino más cercano, por lo que el vecino que se obtiene no es necesariamente el más cercano.

## 5.3. Búsqueda aproximada con el TLAESA

Este algoritmo es el único de la familia AESA que utiliza un árbol como estructura de datos para almacenar el conjunto de entrenamiento. En la

```
funcion FN-aprox
    entrada  t (árbol)
             x (muestra desconocida)
    salida   nn ∈ P (vecino más cercano de x en P)
comienzo
     InsertarCola(t,0)                    //inserción de la raíz del árbol en la cola
     finbusqueda := falso ; d_nn := ∞
     mientras no finbusqueda y no ColaVacia() hacer
         (t, m) := ExtrMinCola()  //extracción del nodo t con la clave mínima m
(a)      si m > d_nn entonces
             finbusqueda := cierto
         si no
             para todo p = Hijo(t) hacer
                 sea M_p el representante de p, y R_p el radio de p
                 d_p := d(x, M_p)
                 si d_p < d_nn entonces                    // actualización de nn
                     d_nn := d_p ; nn := p
                 fsi
                 si d_p ≤ d_nn + R_p entonces               // hijo no eliminable
                     si Hoja(p) then
                         para todo prototipo x_i ∈ S_p hacer
                             si d_p ≤ d(x_i, M_p) + d_nn entonces
                                 d_{x_i} := d(x, x_i)
                                 si d_{x_i} < d_nn entonces    // actualización de nn
                                     d_nn := d_{x_i} ; nn := x_i
                                 fsi
                             fsi
                         fpara
                     si no                                  // no es una hoja
                         InsertarCola(p,d_p − R_p)
                     fsi
                 fsi
             fpara
         fsi
     fmientras
fin FN-aprox
```

Figura 5.2: Algoritmo de Fukunaga y Narendra implementado con una cola de prioridad.

sección 2.4.3 y las figuras 2.17 y 2.18 se describe con detalle este algoritmo; en la fase de entrenamiento se construye un árbol binario en el que cada nodo representa a un subconjunto del conjunto de entrenamiento, contiene un prototipo representante $r$ y dos hijos: el de la derecha tiene el mismo representante que el padre, y el de la izquierda contiene como representante el prototipo más alejado de $r$.

La fase de búsqueda tiene dos etapas: en la primera (figura 2.17), se calculan las distancias de los *prototipos base* a la muestra, lo cual permite obtener (utilizando la desigualdad triangular) una cota inferior de la distancia de cada prototipo a la muestra. La segunda etapa (figura 2.18) consiste en recorrer utilizando un esquema de ramificación y poda el árbol, usando únicamente las cotas inferiores (y la distancia al vecino más cercano hasta ese momento) para podar ramas del árbol. Solamente al llegar a una hoja (que contiene únicamente un prototipo) se calcula la distancia a la muestra[1]; de esta manera se obtiene un número muy pequeño de distancias calculadas.

Como en el caso del algoritmo de Fukunaga y Narendra, el primer paso para poder realizar búsquedas aproximadas en el TLAESA es plantear el recorrido del árbol utilizando una cola de prioridad (véase la figura 5.3); la primera etapa de la fase de búsqueda no es necesario modificarla. En el caso del TLAESA, la distancia del representante del nodo a la muestra no se calcula como en el algoritmo de Fukunaga y Narendra, sino que en su lugar se utiliza una cota inferior de dicha distancia (obtenida a partir de los prototipos base) para podar y dirigir la búsqueda hacia un subárbol u otro. Por tanto, la clave para la cola de prioridad sería $g_i - r_i$ para el hijo izquierdo y $g_d - r_d$ para el hijo derecho. Estas expresiones se deducen directamente de la condición de eliminación de un subárbol (figura 2.18):

...
**si** $d_{nn} + r_i > g_i$ **entonces**
  *// examinar nodo izquierdo, no se puede eliminar*
**fsi**
**si** $d_{nn} + r_d > g_d$ **entonces**
  *// examinar nodo derecho, no se puede eliminar*
**fsi**

Al utilizar una cota para la clave en lugar de la distancia, los valores de $\epsilon$ que son buenos para el algoritmo de Fukunaga y Narendra no resultan ser adecuados para el TLAESA, por lo que es necesario ajustarlos a cada algoritmo en particular.

Como en el caso del algoritmo de Fukunaga y Narendra, lo único que es necesario cambiar para introducir la búsqueda aproximada es la condición marcada con $(a)$ en la figura 5.3, quedando de la siguiente forma:

**si** $(1 + \epsilon) \times m > d_{nn}$ **entonces**

[1]Esta restricción hace que se empiece a podar solamente después de haber alcanzado una hoja.

```
funcion buscarTLAESA-aprox
    entrada  x (muestra a clasificar)
             t ∈ T (nodo del árbol)
             g_t (cota de t)
    salida   nn ∈ P (vecino más cercano de x en P)
comienzo
     InsertarCola(t,g_t)                   //inserción de la raíz del árbol en la cola
     finbusqueda := falso ; d_nn := ∞
     mientras no finbusqueda y no ColaVacia() hacer
         (t, m) := ExtrMinCola()  //extracción del nodo t con la clave mínima m
(a)      si m > d_nn entonces
             finbusqueda := cierto
         si no
             si EsHoja(t) entonces
                 si g_t < d_nn entonces
                     d_t := d(x, t)
                     si d_t < d_nn entonces                 // actualización de nn
                         d_nn := d_t
                         nn := t
                     fsi
                 fsi
             si no       // inserción de los dos hijos en la cola (si no se eliminan)
                 t_i := HijoIzquierda(t) ; t_d := HijoDerecha(t)
                 g_i := G[t_i] ; gd := G[t_d]
                 r_i := Radio(t_i) ; r_d := Radio(t_d)
                 si d_nn + r_i > g_i entonces
                     InsertarCola(t_i,g_i − r_i)
                 fsi
                 si d_nn + r_d > g_d entonces
                     InsertarCola(t_d,g_d − r_d)
                 fsi
             fsi
         fsi
     fmientras
fin buscarTLAESA-aprox
```

Figura 5.3: Algoritmo TLAESA implementado con una cola de prioridad.

Dependiendo del número de prototipos base y de la forma de elegirlos, y de la situación de la muestra en el espacio de representación, la cota inferior que obtiene el TLAESA estará más o menos ajustada a la distancia real de cada prototipo a la muestra; el valor adecuado para $\epsilon$ debe ajustarse de forma empírica en cada tarea.

## 5.4. Búsqueda aproximada con otros algoritmos

El trabajo de Arya y Mount (Arya et al., 1998) se basa en k-d trees y en otro tipo de árboles llamados bbd-trees, y es en una de las implementaciones de estos autores (Mount y Arya, 1997) en la que se utiliza la cola de prioridad para buscar en el k-d tree; esta idea es la que hemos extendido a otros algoritmos en esta tesis. En el caso de los algoritmos basados en árboles que hemos estudiado en esta tesis (véase el capítulo 2) la búsqueda aproximada basada en una cola de prioridad solamente hemos podido aplicarla al algoritmo de Fukunaga y Narendra y al TLAESA; en los casos del vp-tree y GNAT no existe en el algoritmo la noción de cota inferior, si no que más bien se utilizan rangos de distancias para guiar la búsqueda, por lo que no hemos podido implementar de una forma coherente una cola de prioridad.

Los algoritmos AESA y LAESA no utilizan árboles para representar el conjunto de entrenamiento, pero sí utilizan una cota inferior de la distancia de cada prototipo a la muestra. En el caso del AESA el valor de la cota se puede actualizar en cada paso del algoritmo, mientras que en el LAESA las cotas se obtienen en una primera etapa a partir de los prototipos base. Dado que ambos algoritmos disponen de una cota inferior de la distancia y siempre eligen como siguiente candidato a vecino más cercano aquel cuya cota sea la menor, sería posible terminar la búsqueda (encontrando exactamente el vecino más cercano) cuando dicha cota sea mayor que la distancia al vecino más cercano hasta el momento, al menos en el caso del LAESA (en el AESA las cotas pueden cambiar y podría no encontrarse el vecino más cercano).

En la versión del LAESA que aparece en la figura 3.20 la condición para continuar buscando el vecino más cercano es la siguiente:

**mientras** $G[i] < d_{nn}$ **hacer**

donde $G[i]$ es la cota menor de entre los prototipos no seleccionados (el vector de cotas inferiores $G$ se ordena previamente). Si cambiamos dicha condición por

**mientras** $(1 + \epsilon) \times G[i] < d_{nn}$ **hacer**

sería posible realizar una búsqueda aproximada regulada por el valor de $\epsilon$. En el AESA es posible realizar una modificación similar para obtener también una búsqueda aproximada. En ambos algoritmos existe otra manera de realizar una búsqueda más rápida (y aproximada) que consiste en utilizar una cierta *holgura* (Vidal et al., 1985) en las condiciones de eliminación de los algoritmos; ambas formas de búsqueda aproximada son similares, pero no exactamente iguales.

## 5.5. Resultados con datos reales

La búsqueda aproximada con los algoritmos AESA, LAESA, TLAESA y de Fukunaga y Narendra se ha experimentado con dos bases de datos reales: la de los CROMOSOMAS (véase el apéndice A) y una base de datos construida a partir de interpretaciones de melodías musicales.

### 5.5.1. Resultados con la base de datos de melodías musicales

En este caso (véanse los detalles en (Rizo et al., 2003)), de cada interpretación de una melodía se extraen los 8 primeros compases y se codifican en un árbol en el que cada nivel representa la duración (redonda, negra, corchea, etc), y cada hoja está etiquetada con la *altura* en el pentagrama de la nota (do, re, mi, etc). Los nodos intermedios no tienen etiqueta, y dado que la distancia de edición de árboles utilizada (la de Shasha y Zhang (1997)) necesita que todos los nodos se encuentren etiquetados, se realiza un proceso de propagación de etiquetas desde las hojas a la raíz, según una serie de reglas. Además, dado que los árboles pueden tener una profundidad que hace que la distancia de edición resulte muy lenta, se han podado los árboles para que tengan como máximo nivel 5.

En resumen, la base de datos de melodías consta de 641 interpretaciones codificadas como árboles y clasificadas en 149 clases (melodías). Las pruebas se han realizado utilizando el método *leaving-one-out* y, como sucede en otros problemas reales, los mejores resultados se han obtenido con el vecino más cercano; al utilizar $k$ vecinos la tasa de error aumenta. En este caso, al tener cada clase entre 4 y 6 muestras, tampoco parece muy razonable utilizar valores altos de $k$.

Se han realizado experimentos con diferentes valores para $\epsilon$, diferentes para el algoritmo de Fukunaga y Narendra; la evolución según el valor de $\epsilon$ de las tasas de error y los tiempos medios de clasificación de una muestra se puede observar en la figura 5.4. Las líneas con puntos representan la tasa de error, y las líneas sin puntos el tiempo medio de clasificación. Las diferencias entre los resultados de LAESA y TLAESA no son apreciables en la gráfica, y se debe a que utilizan exactamente la misma cota inferior de la distancia; además, al tratarse de una distancia tan cara la menor complejidad temporal del TLAESA (compensada en parte por la gestión de la cola de prioridad) no llega a apreciarse.

En esta tarea, el mejor algoritmo (como siempre que se utilicen distancias caras) es el AESA, pero tiene el gran inconveniente de su complejidad

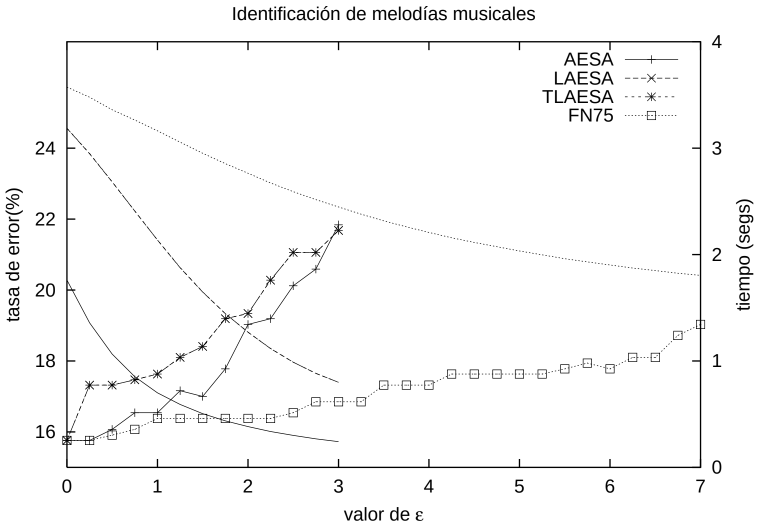


Figura 5.4: Tasa de error (líneas con puntos) y tiempo medio de clasificación (líneas simples) en función del valor de $\epsilon$ en la tarea de identificación de melodías musicales.

espacial cuadrática, lo cual limita su aplicación en la práctica (es de suponer que la base de datos crecerá considerablemente). En el algoritmo de Fukunaga y Narendra se observa que tanto la tasa de error como el tiempo de clasificación van cambiando lentamente con el valor de $\epsilon$, y eso se debe a la cota inferior es más pesimista que la que utiliza la familia AESA y por tanto se aleja mucho de la distancia real. Los resultados de este tipo de búsqueda aproximada son muy interesantes, ya que si permitimos como máximo una pérdida del 2 % en la tasa de error, el tiempo de clasificación se reduce en aproximadamente un tercio en el caso de LAESA, TLAESA y en el algoritmo de Fukunaga y Narendra, y en tres cuartos en el caso del AESA. En general, descartando el AESA por su complejidad espacial, los mejores algoritmos serían el LAESA y TLAESA.

### 5.5.2. Resultados con la base de datos CROMOSOMAS

Los resultados con la base de datos de CROMOSOMAS (algunos de los cuales se presentan en (Moreno-Seco et al., 2003a)) son similares, pero en este caso el algoritmo que mejor se comporta es el de Fukunaga y Narendra (exceptuando el AESA, por supuesto). Los detalles de la base de datos de CROMOSOMAS se pueden consultar en el apéndice A, y en el cuadro 3.2; en resumen, se trata de dos conjuntos de 2200 cromosomas,[2] codificados como

[2]Se realizan dos experimentos en el que uno se utiliza para entrenamiento y el otro para test.

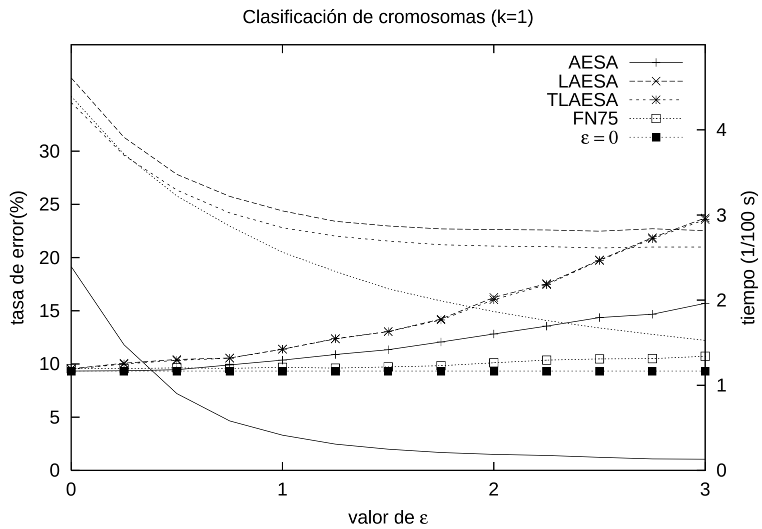


Figura 5.5: Tasa de error (líneas con puntos) y tiempo medio de clasificación (líneas simples) en función del valor de $\epsilon$ en la tarea de clasificación de cromosomas. Como referencia aparece la tasa de error para $\epsilon = 0$.

cadenas de caracteres y clasificados en 22 clases. La figura 5.5 muestra los resultados de la búsqueda aproximada del vecino más cercano para distintos valores de $\epsilon$, con un formato similar al de la figura 5.4 (tasa de error y tiempo de clasificación en función del valor de $\epsilon$).

En el caso de esta base de datos, la tasa de error mejora al utilizar $k$ vecinos; si se utilizan exactamente los $k$ vecinos más cercanos ($k$-NN), el tiempo de clasificación aumenta, aunque no de forma significativa en este caso (la distancia no es excesivamente costosa). También es posible utilizar los $k$ vecinos seleccionados más cercanos ($k$-NSN), como se explica en el capítulo 4. En este caso, y dado que la tasa de error óptima parece ser la de $k = 11$ (sin utilizar búsqueda aproximada), se han realizado experimentos con distintos valores de $\epsilon$ utilizando 11 vecinos (los más cercanos o los seleccionados más cercanos) para la clasificación. La figura 5.6 muestra los resultados del algoritmo de Fukunaga y Narendra; los resultados de los demás algoritmos (AESA, LAESA y TLAESA) son similares. Debemos destacar que el tiempo de clasificación de los 11-NSN es exactamente el mismo que para $k = 1$, y el tiempo de clasificación de los 11-NN es siempre mayor aún para valores altos de $\epsilon$. Si el objetivo de la búsqueda aproximada es mejorar los tiempos de clasificación, la regla de los vecinos seleccionados más cercanos es especialmente interesante en estas tareas ya que permite mejorar las tasas de acierto en la clasificación (cuando es posible) sin ningún incremento apreciable en el tiempo de clasificación.

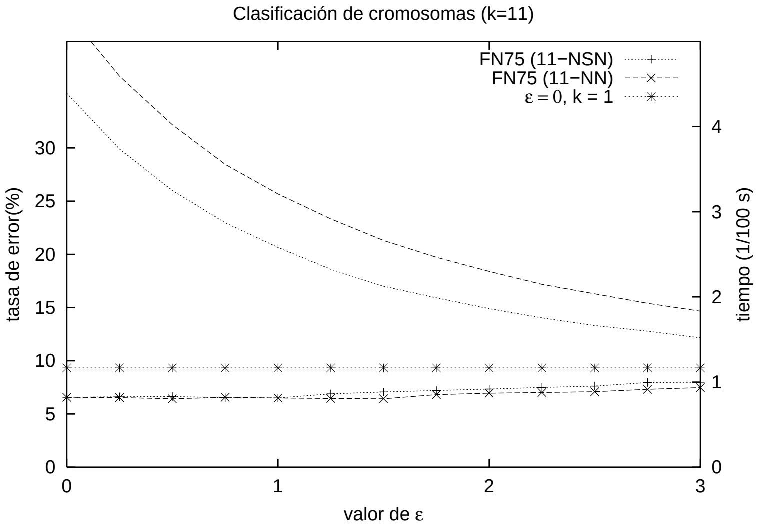


Figura 5.6: Tasa de error (líneas con puntos) y tiempo medio de clasificación (líneas simples) del algoritmo de Fukunaga y Narendra utilizando 11 vecinos y las reglas de clasificación $k$-NSN y $k$-NN, en función del valor de $\epsilon$. Como referencia aparece la tasa de error para $k = 1$ y $\epsilon = 0$.

## 5.6. Resultados con datos sintéticos

Con el propósito de obtener una referencia de la eficacia de la búsqueda aproximada con los algoritmos de la familia AESA y el de Fukunaga y Narendra, se han comparado sus resultados con los que obtiene la implementación presente en el paquete ANN (Mount y Arya, 1997) utilizando una base de datos sintética (la base de datos SINTE, véase el apéndice A). De todas las implementaciones de la búsqueda aproximada presentes en el paquete ANN se ha utilizado la que emplea k-d trees y una cola de prioridad. La comparación se ha realizado para distintos valores de $\epsilon$, con 4096 prototipos en el conjunto de entrenamiento y 1024 de test, utilizando la regla de los $k$ vecinos más cercanos con $k = 25$.

Por un lado se ha comparado la evolución de la tasa de error según el valor de $\epsilon$, y por otro lado se ha comparado el número de distancias calculadas por los diferentes algoritmos. Los tiempos de clasificación no se han comparado puesto que las implementaciones son muy diferentes entre sí, y al tratarse de un espacio de vectores en el que el cálculo de la distancia es poco costoso, el principal factor que influye en el tiempo de clasificación es el coste adicional de cada algoritmo (en todas las pruebas realizadas el algoritmo más rápido ha sido el del paquete ANN). En los resultados de la comparación de distancias debe tenerse en cuenta que la implementación de Arya y Mount utiliza un k-d tree, que realiza cálculos parciales de distancias

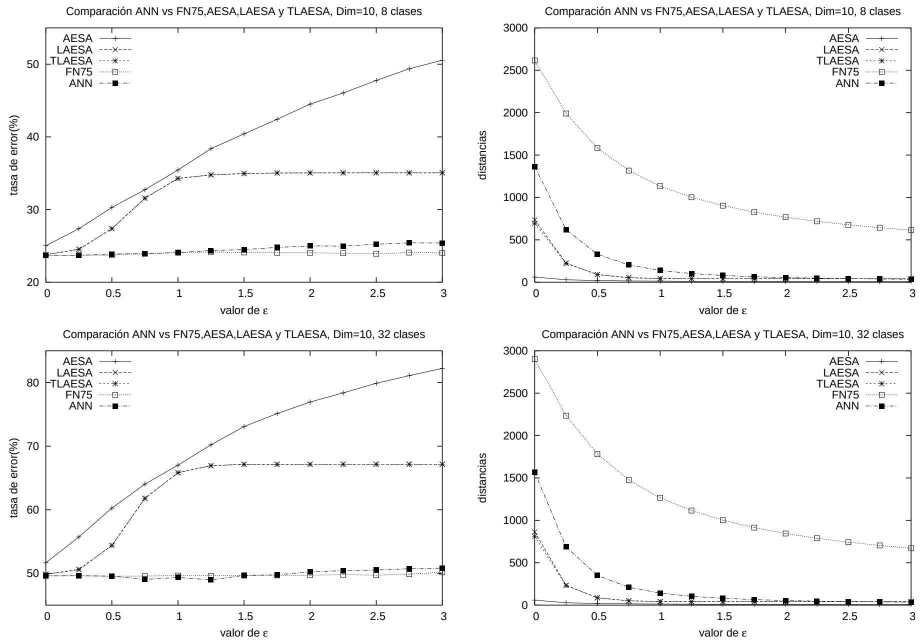


Figura 5.7: Comparación de resultados entre la búsqueda aproximada de ANN (Mount y Arya, 1997) y la implementada en otros algoritmos, con datos de dimensión 10. A la izquierda se muestra la tasa de error y a la derecha el número medio de distancias calculadas. Se han utilizado 4096 prototipos de entrenamiento y 1024 de test, y se ha empleado la regla de los $k$ vecinos más cercanos, con $k = 25$.

para guiar la búsqueda en el árbol, y solamente calcula distancias completas en las hojas, que son las que se han contabilizado (en esta implementación, cada hoja contiene únicamente un prototipo).

La figura 5.7 muestra los resultados obtenidos con datos de dimensión 10 procedentes de 8 y 32 clases (gráficas de arriba y abajo, respectivamente). Las gráficas de la izquierda muestran tasa de error, y las de la derecha muestran número medio de distancias calculadas. La figura 5.8 muestra los mismos resultados para datos de dimensión 20.

En las figuras 5.7 y 5.8 se puede observar que los algoritmos de la familia AESA funcionan bien cuando se utilizan valores pequeños de $\epsilon$, pero el algoritmo del paquete ANN alcanza el mismo nivel de distancias (completas) calculadas con un aumento en la tasa de error apenas apreciable. Por otro lado, el algoritmo de Fukunaga y Narendra parece requerir valores más grandes de $\epsilon$, pero no tiende a obtener resultados tan buenos en lo referente a número de distancias como los obtenidos por el algoritmo del paquete ANN y los algoritmos de la familia AESA. Aunque ninguno de los

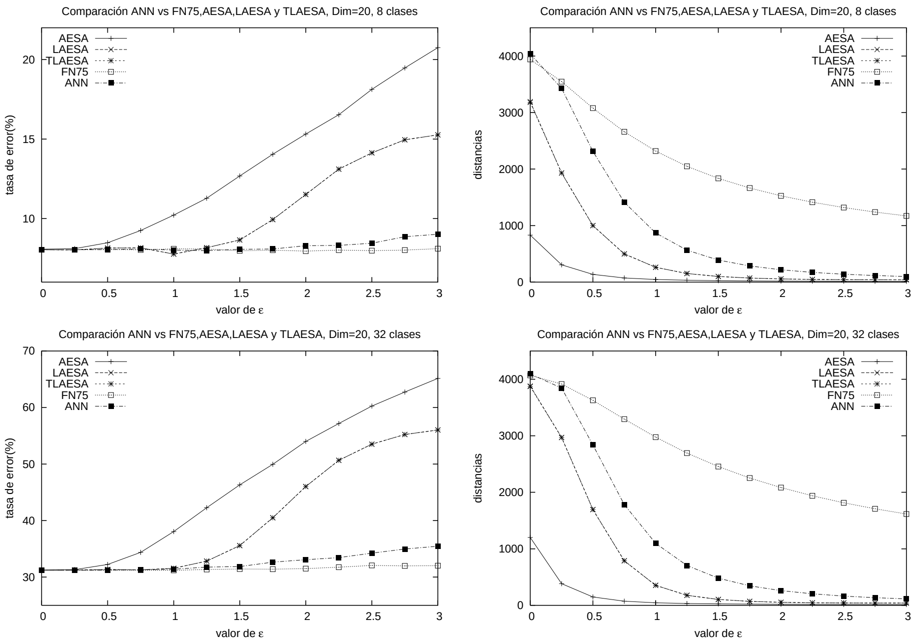


Figura 5.8: Comparación de resultados entre la búsqueda aproximada de ANN (Mount y Arya, 1997) y la implementada en otros algoritmos, con datos de dimensión 20 y $k = 25$.

algoritmos extendidos para la búsqueda aproximada en este capítulo mejora los resultados del algoritmo del paquete ANN (algo habitual cuando se trata de espacios de vectores de dimensionalidad moderada), los resultados son razonablemente buenos e indican que son una alternativa interesante en espacios que no permitan una representación con vectores (espacios métricos generales).

### 5.6.1. Resultados con la regla $k$-NSN

En los experimentos descritos en el epígrafe anterior también se ha empleado la regla de los $k$ vecinos seleccionados más cercanos, obteniéndose (como con la base de datos de CROMOSOMAS) resultados similares a los de la regla de los $k$ vecinos más cercanos. Los parámetros del experimento son los que se describen en la sección anterior: 4096 prototipos de entrenamiento y 1024 de test, y $k = 25$. Las figuras 5.9 y 5.10 muestran los resultados de la aplicación de las dos reglas de clasificación con el algoritmo de Fukunaga y Narendra;[3] como referencia se muestran tambien los resultados del algoritmo del paquete ANN. Como se puede observar, la regla $k$-NSN obtiene resultados similares a los de la regla $k$-NN cuando se utiliza búsqueda aproximada.

[3]Los resultados del LAESA, TLAESA y AESA no se muestran porque en esta tarea la tasa de error aumenta rápidamente con $\epsilon$ (véanse las figuras 5.7 y 5.8).

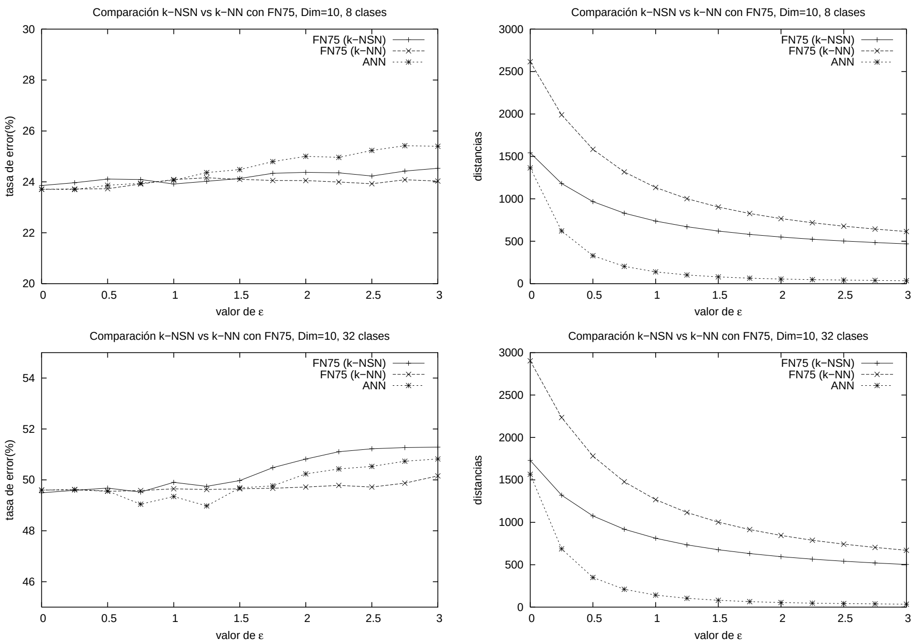


Figura 5.9: Comparación de resultados entre la regla $k$-NSN y la $k$-NN aplicadas al algoritmo de Fukunaga y Narendra (FN75) con búsqueda aproximada, con datos de dimensión 10. A la izquierda se muestra la tasa de error y a la derecha el número medio de distancias calculadas. Se han utilizado 4096 prototipos de entrenamiento y 1024 de test, con $k = 25$.

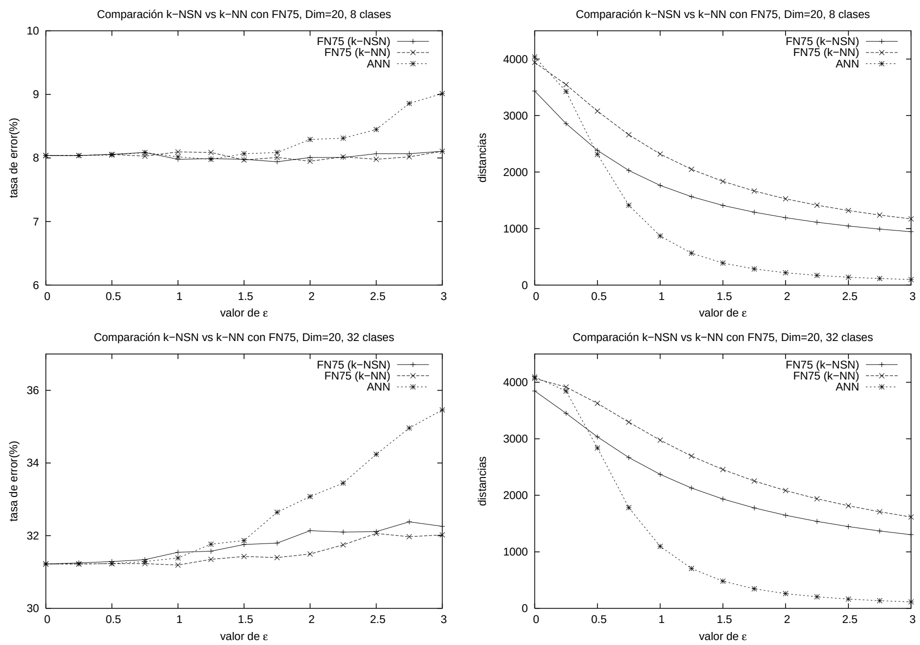


Figura 5.10: Comparación de resultados entre la regla $k$-NSN y la $k$-NN aplicadas al algoritmo de Fukunaga y Narendra con búsqueda aproximada, con datos de dimensión 20 y $k = 25$.

# Capítulo 6

# Conclusiones y desarrollos futuros

## 6.1. Conclusiones

Los clasificadores basados en la regla del vecino más cercano se utilizan en muchas tareas reales por su simplicidad (una vez definida una distancia entre prototipos) y por sus buenos resultados. En general, si la distancia es costosa, el tamaño del conjunto de entrenamiento es grande o la dimensionalidad de los datos es alta se puede utilizar uno de los muchos algoritmos rápidos para encontrar el vecino más cercano, como alternativa a la búsqueda exhaustiva. Por otro lado, si se utilizan en la clasificación los $k$ vecinos más cercanos, la tasa de error normalmente disminuye. Para encontrar exactamente los $k$ vecinos más cercanos se puede extender un algoritmo rápido de búsqueda del vecino más cercano; en este caso, el número de distancias y el tiempo de clasificación aumentan con el valor de $k$ (véase la sección 2.5).

La primera de las aportaciones de esta tesis (capítulo 3) es un conjunto de mejoras sobre un algoritmo rápido de búsqueda del vecino más cercano, el LAESA, con el objetivo de reducir su tasa de error en la clasificación, utilizando para ello $k$ vecinos (sean o no los más cercanos) y manteniendo el mismo tiempo de clasificación (e incluso reduciéndolo en algún caso). Los algoritmos resultantes de las distintas mejoras se han denominado A$k$-LAESA, y obtienen tasas de error similares a las de un clasificador que utilice exactamente los $k$ vecinos más cercanos, pero sin incrementar el tiempo de clasificación. Estos resultados convierten al A$k$-LAESA en un clasificador mucho más potente que el LAESA con exactamente el mismo número de distancias calculadas y prácticamente el mismo tiempo de clasificación.

La última de las versiones del A$k$-LAESA, la versión 4, utiliza para la votación los $k$ vecinos más cercanos de entre aquellos que hayan sido seleccionados por el algoritmo de búsqueda (el del LAESA). La idea de utilizar los $k$ vecinos seleccionados más cercanos se puede extender a otros algoritmos basados como el LAESA en el esquema de aproximación y eliminación (capítulo 4), definiendo de esta manera una nueva regla de clasificación, la regla de los $k$ vecinos seleccionados más cercanos ($k$-NSN), que se situa en un punto intermedio entre la regla del vecino más cercano y la de los $k$ vecinos más cercanos, con las ventajas de ambas reglas: es tan rápida como la regla del vecino más cercano, y consigue tasas de error similares a las de la regla de los $k$ vecinos más cercanos. Los experimentos realizados permiten afirmar que el porcentaje de los prototipos utilizados en la votación de la regla $k$-NSN que se encuentran entre los $k$ vecinos más cercanos es muy alto, y que dicho porcentaje aumenta con la dimensionalidad de los datos. De forma simplificada, se podría decir que los algoritmos rápidos de búsqueda del vecino más cercano encuentran en su camino todos o casi todos los $k$ vecinos más cercanos (para valores razonables de $k$ como los utilizados en clasificación), pero los desechan; la regla $k$-NSN los aprovecha para mejorar la clasificación sin incrementar por ello el tiempo de clasificación.

Los resultados de la regla $k$-NSN aplicada a diferentes algoritmos indican que existe una relación entre el número de distancias calculadas (el número de prototipos seleccionados) y la tasa de error: cuantos más prototipos selecciona un algoritmo, más se acerca su tasa de error a la de los $k$ vecinos más cercanos (y más aumenta el porcentaje de éstos entre los que utiliza la regla $k$-NSN para votar). Los casos extremos en casi todos los experimentos son el AESA, que suele ser el que menos distancias calcula y el que menos se acerca a la tasa de error de los $k$ vecinos más cercanos al utilizar la regla $k$-NSN, y el vp-tree, que suele calcular un número muy elevado de distancias (en algunos casos cerca del máximo posible), y a la vez obtiene prácticamente siempre las mismas tasas de error que la regla de los $k$ vecinos más cercanos. Además, cuanto más aumenta la dimensión de los datos, más se acercan las tasas de error de la regla $k$-NSN a las de la regla $k$-NN.

En muchas tareas reales en las que los prototipos se representan como cadenas de caracteres o árboles, por ejemplo, el tiempo empleado por un clasificador basado en el vecino más cercano es excesivo. La tercera aportación de este trabajo es la extensión de las ideas de Arya et al. (1998) sobre búsqueda aproximada del vecino más cercano (pensadas para prototipos representados como vectores) a otros algoritmos válidos para cualquier clase de prototipo (siempre que exista una distancia que cumpla las propiedades de una métrica), como son el de Fukunaga y Narendra, el TLAESA, el AESA y el LAESA. De esta forma, se consiguen tiempos de clasificación mucho menores (del orden de una tercera parte en el caso de la identificación de melodías codificadas como árboles) con un incremento aceptable en el error de clasificación. Además, la búsqueda aproximada puede combinarse con la regla $k$-NSN, obteniéndose resultados similares a los de la búsqueda no aproximada: la tasa de error de la regla $k$-NSN es similar a la de la regla $k$-NN, pero el número de distancias no aumenta con el valor de $k$.

## 6.2. Trabajos futuros

Aunque la continuación lógica del trabajo relacionado con el A$k$-LAESA (cap. 3) es la regla de los $k$ vecinos seleccionados más cercanos (cap. 4), y se da la circunstancia que el LAESA extendido con esta regla (el A$k$-LAESA en su versión 4) es de los algoritmos que peores resultados obtiene, es posible explorar otras mejoras en el A$k$-LAESA:

1. Terminar la búsqueda del vecino más cercano cuando se hayan calculado un número constante de distancias (fijado de antemano en función del valor de $k$), y clasificar la muestra utilizando los $k$ vecinos seleccionados más cercanos; habría que estudiar la evolución de las tasas de error en diferentes circunstancias: dimensionalidad elevada, datos reales, etc. Un algoritmo que calcule un número constante de distancias (que no crece con el tamaño del conjunto de entrenamiento) asegura un tiempo casi constante de clasificación, al menos cuando la distancia es el principal factor que influye en el tiempo de clasificación.

2. Estudiar la influencia del número de prototipos base y del algoritmo de selección en la tasa de error del A$k$-LAESA. En todos los experimentos de esta tesis se ha utilizado el número de prototipos base óptimo para el LAESA (en aquellos casos en los que se conocía (Micó, 1996)), y el mismo algoritmo de selección. En el caso del LAESA, el objetivo es reducir al máximo el número de distancias calculadas, pero en el caso del A$k$-LAESA el objetivo es además conseguir una tasa de error similar a la de los $k$ vecinos más cercanos, y dado que el número de prototipos base y el algoritmo de selección influyen en el cálculo de la cota inferior de la distancia, y la cota determina el orden en que se seleccionan los prototipos, una cota más lejana (o a lo mejor más cercana) a la distancia real podría hacer que las tasas de error del A$k$-LAESA disminuyeran, aunque probablemente se incrementaría el número de distancias calculadas.

3. La última versión del A$k$-LAESA utiliza los $k$ vecinos seleccionados durante la búsqueda del vecino más cercano, pero es posible que los resultados mejoren si se buscan los $m$ vecinos más cercanos, con $k > m$ y valores pequeños de $m$, con el fin de no aumentar mucho el número de distancias calculadas.

Aunque los resultados empíricos avalan la solidez de la regla $k$-NSN, sería deseable encontrar una justificación más formal de su comportamiento, aunque probablemente implicaría un análisis algoritmo por algoritmo de los mecanismos de selección de prototipos como candidatos a vecino más cercano. Además, sería interesante extender con esta regla todos aquellos algoritmos rápidos de búsqueda del vecino más cercano que lo permitan, con el fin de confirmar los resultados de esta tesis.

Por otro lado, se pretende estudiar (Sánchez, 2003) la aplicación de la idea subyacente en la regla $k$-NSN en conjunción con la regla $k$-NCN (Sánchez et al., 2000; Sánchez, 1998), ya que esta última requiere un cálculo exhaustivo de los centroides sucesivos. En experimentos preliminares hemos comprobado que el tiempo de clasificación de dicha regla se puede reducir utilizando para obtener los centroides únicamente aquellos prototipos seleccionados por un algoritmo rápido de búsqueda del vecino más cercano, pero las tasas de error aumentan. El objetivo de este trabajo sería por tanto reducir el tiempo de clasificación sin aumentar excesivamente las tasas de error.

Los resultados presentados en el capítulo 5 abren una interesante línea de trabajo, que podría comenzar por extender la búsqueda aproximada con colas de prioridad a otros algoritmos como el vp-tree y el GNAT. En unos experimentos preliminares con distintas claves para la cola de prioridad no se han conseguido buenos resultados, pero habría que intentar encontrar una expresión adecuada para dicha clave en ambos algoritmos. Además, la clave empleada en los algoritmos de Fukunaga y Narendra y TLAESA es una cota inferior de la distancia de cualquier prototipo del nodo a la muestra, y probablemente podría ajustarse para reducir más el tiempo de búsqueda sin aumentar demasiado la tasa de error.

# Apéndice A

# Bases de datos utilizadas en los experimentos

En este apéndice se describen las bases de datos que se han utilizado en los experimentos de los capítulos 3, 4 y 5, con el fin de evitar repetir en cada capítulo las características de cada base de datos. En cada uno de ellos se hace referencia a este apéndice y para facilitar la identificación de cada base de datos se les ha asignado una abreviatura o nombre. En dichos capítulos se utilizan además otras bases de datos, pero para simplificar las comparaciones entre los distintos algoritmos se han realizado experimentos con todos ellos con las bases de datos de este apéndice.

## A.1. Base de datos SINTE

Esta base de datos se ha generado de forma sintética utilizando el algoritmo que aparece en un apéndice del libro de Jain y Dubes (1988). Dicho algoritmo genera puntos en un espacio $\mathbb{R}^n$ siguiendo una distribución normal y tiene como entrada los siguientes parámetros:

1. La dimensionalidad del espacio.
2. Número de puntos que se desea generar.
3. Número de clases.
4. Número mínimo de puntos por clase.
5. Varianza (común para todas las clases).
6. Solapamiento máximo entre dos clases.

El algoritmo funciona de forma incremental: primero genera al azar la media de una clase y los puntos correspondientes; a continuación, genera al azar la media de la siguiente clase y sus puntos, y mide el solapamiento. Si el solapamiento es menor que el máximo, sigue con la siguiente clase; de lo contrario, cambia la media y vuelve a generar los puntos. Este proceso se repite hasta que los puntos de todas las clases se hayan generado.

Puede ocurrir que el algoritmo esté continuamente generando una clase (la media y los puntos correspondientes) y no consiga que el solapamiento sea menor que el máximo. Cuando esta situación se produce 50 veces, el propio algoritmo incrementa el solapamiento máximo y continua.

Para los experimentos de esta tesis se han generado en total cuatro conjuntos de puntos con dimensiones 10 y 20, y con 8 y 32 clases. El número de puntos total es de 69932, con la idea de construir conjuntos de entrenamiento de 65536 puntos (y 4096 para test); sin embargo, tras unos experimentos preliminares se observó que los resultados con conjuntos de datos tan grandes eran similares a los obtenidos con 16384 puntos, y el tiempo de entrenamiento de los algoritmos era muy superior (un par de dias en algún caso). Por este motivo, los conjuntos de entrenamiento utilizados tienen como máximo 16384 puntos en total. La varianza se ha fijado en 0.1 y el solapamiento máximo en 0.12, aunque para el caso de 32 clases y dimensión 10 el algoritmo de Jain y Dubes (1988) aumentó este valor a 0.120175.[1] El número de puntos por clase es el mismo para todas las clases (el número total de puntos dividido por el número de clases) para que todas sean equiprobables.

A partir de cada uno de los cuatro conjuntos de datos generados se han realizado particiones con distintos tamaños para el conjunto de entrenamiento, manteniendo siempre la proporción de puntos por clase en todos los conjuntos de entrenamiento y de test. Los tamaños para los conjuntos de entrenamiento son 1024, 4096 y 16384, y el tamaño del conjunto de test es de 1024 puntos. Se han realizado 10 particiones para cada tamaño del conjunto de entrenamiento, utilizando siempre los primeros puntos de los conjuntos generados con el algoritmo de Jain y Dubes (1988). La distancia utilizada con estos experimentos ha sido la distancia euclídea ($L_2$ de Minkowski). Debido al tamaño del conjunto de entrenamiento más grande (16384), el algoritmo AESA se ha modificado ligeramente, ya que no podía contener la matriz cuadrada de 16384 filas por 16384 (lo cual requeriría como mínimo 2 GB de memoria); en el caso de que el conjunto de entrenamiento tenga 16384 muestras, el AESA no construye la matriz de distancias y en su lugar calcula las distancias según se necesiten. Esta solución supone mayor tiempo de clasificación, pero en esta base de datos los tiempos no se han medido al ser muy pequeños por tratarse de la distancia euclídea.

Las tasas de error obtenidas van desde aproximadamente el 17 % con 8 clases y dimensión 20, al 65 % con 32 clases y dimensión 10. Estas tasas tan altas permiten apreciar bien las mejoras obtenidas cuando se usan $k$ vecinos

[1] Si el número total de puntos hubiera sido el finalmente utilizado, 16384+1024, es probable que el solapamiento se hubiera quedado en 0.12.

para clasificar, bien utilizando los $k$ vecinos más cercanos o bien utilizando los $k$ vecinos seleccionados más cercanos (véase el capítulo 4).

## A.2. Base de datos CROMOSOMAS

Esta es una base de datos reales conocida como *Copenhagen* (Lundsteen et al., 1980; Granum et al., 1989; Granum y Thomason, 1990) que nos proporcionó Alfons Juan, por lo cual le estamos muy agradecidos. En su tesis doctoral (Juan, 1999) describe con todo detalle esta base de datos y las características de la distancia de edición que utiliza. La base de datos se encuentra también disponible en la página web del IAPR TC5 (Lucas) en formato XML.

Se trata de dos conjuntos de 2200 cromosomas humanos cada uno (4400 en total) codificados como cadenas y clasificados en 22 clases. Es posible utilizar un corpus extendido (denominado *corpus X* en la tesis de Juan (1999)) que contiene además información acerca del centrómero (codificado con una X en la cadena, de ahí el nombre del corpus), pero para los experimentos de esta tesis se ha utilizado el corpus en bruto. Un ejemplo de cadena que describe un cromosoma es la siguiente:

```
/ 514 69 10 47 17 / A=B=aA==a=Aa==D==c==C====b==A===a=A==c====A=a=b
```

El primer número (514 en el ejemplo) es el identificador del cromosoma, el segundo (69) es la metafase, el tercero (10) es la clase del cromosoma y el cuarto es la longitud de la cadena. El último número indica la posición del centrómero en la cadena. En los experimentos realizados para esta tesis solamente se ha utilizado la clase y la longitud de la cadena. El alfabeto está compuesto por las letras `a`,`b`,`c`,`d`,`e`, `A`,`B`,`C`,`D`,`E` y el símbolo "`=`" (aunque en una cadena de las 4400 aparece una "`f`").

Con estos conjuntos se han realizado dos experimentos, utilizando cada vez uno de ellos para entrenamiento y el otro para test. Se ha empleado la distancia de edición normalizada, con los pesos de inserción y borrado fijados a 1 y el peso de sustitución a 1 si los caracteres son distintos y a 0 si son iguales. [2]

Es destacable el hecho de que al utilizar la distancia de edición los tiempos de clasificación de cada algoritmo van paralelos al número de distancias que calcula, algo que no sucede cuando se utilizan distancias euclídeas debido a lo poco costosas que resultan estas distancias en comparación con el coste del resto del algoritmo de búsqueda. Por este motivo, hemos intentado utilizar otras bases de datos basadas en distancia de edición entre cadenas o entre árboles, pero en todos los casos al aumentar el valor de $k$ para utilizar $k$ vecinos en la clasificación la tasa de error disminuía muy ligeramente para

[2] En la tesis de Alfons Juan (Juan, 1999) se utiliza una matriz de pesos de sustitución que pondera las sustituciones según lo lejano que esté un carácter de otro, pero en esta tesis hemos optado por un peso de sustitución simple.

comenzar a crecer con el valor de $k$, haciendo que en muchos casos el valor óptimo para $k$ fuese 1. Este comportamiento era poco dependiente de si se utilizaban o no distancias normalizadas o distintas matrices de pesos de sustitución, y, aunque es un problema por explorar, se escapa de la temática de esta tesis.

## A.3. Base de datos UCI

De todas las bases de datos presentes en el conocido repositorio de la UCI (Blake y Merz, 1998), solamente algunas de ellas se han utilizado para esta tesis, en concreto aquellas en las que la utilización de $k$ vecinos para clasificar mejora apreciablemente las tasas de acierto. Las principales características de estas bases de datos se describen en la siguiente tabla:

| Nombre | Clases | Dimensión | Prototipos |
|---|---:|---:|---:|
| heart | 2 | 13 | 270 |
| liver-disorders | 2 | 6 | 345 |
| pima-diabetes | 2 | 8 | 768 |
| wdbc | 2 | 30 | 569 |

En todos los casos la distancia utilizada ha sido la euclídea, y también en todos ellos se ha utilizado (a diferencia de las otras bases de datos de esta tesis) el método *leave-one-out* en los experimentos, dado el reducido número de prototipos que contienen.

## A.4. Base de datos PHONEME

Esta base de datos procede del proyecto ESPRIT ROARS (Alinat, 1993), y contiene 5404 prototipos de dimensión 5 que representan fonemas de vocales nasales y orales (dos clases). El conjunto completo nos los proporcionó J. Salvador Sánchez, junto con 5 pares de conjuntos de entrenamiento y de test obtenidos a partir del conjunto inicial. Cada conjunto de entrenamiento contiene aproximadamente 4300 prototipos, y cada conjunto de test contiene unas 1080 muestras.

Aunque en la documentación original dice que los mejores resultados se obtienen con el método *leave-one-out* y $k = 1$, también se dice que en el conjunto hay tres muestras por vocal, por lo que se sugiere utilizar $k = 20$. En nuestro caso, hemos optado por realizar los experimentos con los 5 pares entrenamiento/test y diferentes valores de $k$, dado que el número de prototipos es más elevado que en las bases de datos UCI.

# Bibliografía

P. Aibar, A. Juan, y E. Vidal. Extensions to the approximating and eliminating search algorithm (AESA) for finding k-nearest-neighbours. En *New Advances and Trends in Speech Recognition and Coding*, pp. 23–28, 1993.

P. Alinat. Periodic progress report 4, ROARS project ESPRIT II - number 5516. Thomson technical report ts asm 93/s/egs/nc/079, 1993.

S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, y A. Wu. An optimal algorithm for approximate nearest neighbor searching. *Journal of the ACM*, 45:891–923, 1998.

J.L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18:509–517, 1975.

C.L. Blake y C.J. Merz. UCI repository of machine learning databases, 1998. URL `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

T. Bozjaya y M. Ozsoyoglu. Distance-based indexing for high dimensional metric spaces. En *Proc. ACM SIGMOD International Conference on Management of Data*, volume 26, pp. 357–368, 1997.

L. Breiman, J. Friedman, R. Olshen, y C. Stone. *Classification and regression trees*. Wadsworth International Group., 1984.

S. Brin. Near neighbor search in large metric spaces. En *Proceedings of the $21^{st}$ VLDB Conference*, pp. 574–584, 1995.

E. Chavez, G.Ñavarro, R.A. Baeza-Yates, y J.L. Marroquin. Searching in metric spaces. *ACM Computing Surveys*, 33(3):273–321, 2001.

K.L. Clarkson. Nearest neighbor queries in metric spaces. *Discrete Computational Geometry*, 22(1):63–93, 1999.

F.J. Cortijo. Apuntes de reconocimiento de formas, 2001. URL `http://www-etsi2.ugr.es/depar/ccia/rf/material.html`.

B.V. Dasarathy. *Nearest neighbor norms: NN pattern classification techniques*. IEEE Computer Society Press, 1991.

P.A. Devijver y J. Kittler. *Pattern Recognition: A Statistical Approach*. Prentice-Hall, 1982.

R. Duda y P. Hart. *Pattern Recognition and Scene Analysis*. Wiley, 1973.

R. Duda, P. Hart, y D. Stork. *Pattern Classification*. Wiley, 2001.

R.P.W. Duin, F. Roli, y D. de Ridder. A note on core research issues for statistical pattern recognition. *Pattern Recognition Letters*, 23:493–499, 2002.

J.H. Friedman, F. Baskett, y L.J. Shustek. An algorithm for finding nearest neighbors. *IEEE Transactions on Computers*, 24(10):1000–1006, 1975.

J.H. Friedman, J.L. Bentley, y R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3:209–226, 1977.

K. Fukunaga. *Introduction to Statistical Pattern Recognition*. Academic Press, 1990.

K. Fukunaga y M. Narendra. A branch and bound algorithm for computing $k$–nearest neighbors. *IEEE Transactions on Computing*, 24:750–753, 1975.

E. Gómez, L. Micó, y J. Oncina. Testing the linear approximating and eliminating search algorithm in handwritten character recognition tasks. En *Proceedings of the VI Spanish Symposium on Pattern Recognition and Image Analysis*, pp. 212–217, Córdoba (Spain), 1995.

E. Granum y M.G. Thomason. Automatically inferred Markov network models for classification of chromosomal band pattern structures. *Cytometry*, 11:26–39, 1990.

E. Granum, M.G. Thomason, y J. Gregor. On the use of automatically inferred Markov networks for chromosome analysis. En C. Lundsteen y J. Piper, editores, *Automation of Cytogenetics*, pp. 233–251. Springer-Verlag, Berlin, 1989.

W.H. Highleyman. The design and analysis of patter recognition experiments. *Bell Systems Technical Journal*, 41:723–744, 1962.

P. Indyk y R. Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. En *Proceedings of the 30th ACM Symposium on Theory of Computing*, pp. 604–613, 1998.

A.K. Jain y R.C. Dubes. *Algorithms for clustering data*. Prentice-Hall, 1988.

A. Juan. *Optimización de prestaciones en técnicas de aprendizaje no supervisado y su aplicación al reconocimento de formas.* Tesis doctoral. Universidad Politécnica de Valencia, 1999.

J. Kleinberg. Two algorithms for nearest-neighbor search in high dimensions. En *Proceedings of 29th ACM Symposium on Theory of Computing*, pp. 599–608, 1997.

P.A. Lachenbruch y R.M. Mickey. Estimation of error rates in discriminant analysis. *Technometrics*, 10:1–11, 1968.

S. Lucas. Iapr tc5 benchmarking and software, datasets. `http://algoval.essex.ac.uk/tc5/Datasets.html`, (enlace comprobado el 28 de julio de 2003).

C. Lundsteen, J. Phillip, y E. Granum. Quantitative analysis of 6985 digitized trypsin G-banded human metaphase chromosomes. *Clinical Genetics*, 18:355–370, 1980.

L. Micó. *Algoritmos para la búsqueda del vecino más cercano en espacios métricos.* Tesis doctoral. Universidad Politécnica de Valencia, 1996.

L. Micó y J. Oncina. Comparison of fast nearest neighbour classifiers for handwritten character recognition. *Pattern Recognition Letters*, 19:351–356, 1998.

L. Micó, J. Oncina, y R. C. Carrasco. A fast branch and bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17:731–739, 1996.

L. Micó, J. Oncina, y E. Vidal. A new version of the nearest neighbour approximating and eliminating search algorithm (AESA) with linear preprocessing-time and memory requirements. *Pattern Recognition Letters*, 15:9–17, 1994.

F. Moreno-Seco, J.M. Iñesta, L. Micó, y J. Oncina. Fast $k$–neighbour classification of human vertebrae levels. En *Proceedings of the IX Spanish Symposium on Pattern Recognition and Image Analysis*, pp. 343–348, Benicassim (Spain), 2001.

F. Moreno-Seco, L. Micó, y J. Oncina. A fast approximately k-nearest-neighbour search algorithm for classification tasks. En *Advances in Pattern Recognition, Proceedings of the Joint IAPR International Workshops SSPR'2000 and SPR'2000 (Alicante, Spain). Lecture Notes in Computer Science, F.J. Ferri et al (Eds.) vol. 1876, Springer-Verlag.*, pp. 823–831, Berlin, 2000.

F. Moreno-Seco, L. Micó, y J. Oncina. Extending LAESA fast nearest neighbour algorithm to find the $k$ nearest neighbours. En *Proceedings of the Joint IAPR International Workshops SSPR'2002 and SPR'2002 (Windsor, Canada). Lecture Notes in Computer Science, T. Caelly et al (Eds.) vol. 2396, Springer-Verlag.*, pp. 691–699, Berlin, 2002.

F. Moreno-Seco, L. Micó, y J. Oncina. Approximate nearest neighbour search with the fukunaga and narendra algorithm and its application to chromosome classification. En *Proceedings of the 8th Iberoamerican Congress on Pattern Recognition CIARP'2003 (La Habana, Cuba). Lecture Notes in Computer Science, A. Sanfeliu et al (Eds.) vol. 2905, Springer-Verlag.*, pp. 322–328, Berlin, 2003a.

F. Moreno-Seco, L. Micó, y J. Oncina. Extending fast nearest neighbour search algorithms for approximate $k$-nn classification. En *Proceedings of the 1st Iberian Conference on Pattern Recognition and Image Analysis (Port d'Andratx, España). Lecture Notes in Computer Science, F.J. Perales et al (Eds.) vol. 2652, Springer-Verlag.*, pp. 589–597, Berlin, 2003b.

F. Moreno-Seco, L. Micó, y J. Oncina. A modification of the LAESA algorithm for approximated $k$-nn classification. *Pattern Recognition Letters*, 24(1–3):47–53, 2003c.

F. Moreno-Seco, J. Oncina, y L. Micó. Improving the LAESA algorithm error rates. En *Proceedings of the VIII Symposium Nacional de Reconocimiento de Formas y Análisis de Imágenes*, pp. 413–419, Bilbao (Spain), 1999.

D.M. Mount y S. Arya. Ann: A library for approximate nearest neighbor searching, 1997. url: `http://www.cs.umd.edu/ mount/ANN`.

M. Murphy y S. Skiena. Ranger - nearest neighbor search in high dimensions, 1996.
`http://www.cs.sunysb.edu/ algorith/implement/ranger/implement.shtml`, (enlace comprobado el 2 de abril de 2003).

V. Ramasubramanian y K.K. Paliwal. Fast nearest-neighbor search algorithms based on approximation-elimination search. *Pattern Recognition*, 33: 1497–1510, 2000.

D. Rizo, J.M. Iñesta, y F. Moreno-Seco. Tree-structured representation of musical information. En *Proceedings of the 1st Iberian Conference on Pattern Recognition and Image Analysis (Port d'Andratx, Spain). Lecture Notes in Computer Science, F.J. Perales et al (Eds.) vol. 2652, Springer-Verlag.*, pp. 838–846, Berlin, 2003.

J.S. Sánchez. *Aprendizaje y clasificación basados en criterios de vecindad. Métodos alternativos y análisis comparativo.* Tesis doctoral. Universitat Jaume I, Castellón., 1998.

J.S. Sánchez. Comunicación personal., 2003.

J.S. Sánchez, F. Pla, y F.J. Ferri. Surrounding neighbourhood techniques for nearest–neighbour based classification and prototype selection. En *Recent Research Developments in Pattern Recognition*, volume 1, pp. 63–76. Transworld Research Network, 2000.

S. Shasha y K. Zhang. *Approximate Tree Pattern Matching. Pattern Matching Algorithms*, pp. 341–371. Oxford Press, 1997.

S. Theodoridis y K. Koutroumbas. *Pattern Recognition.* Academic Press, 1999.

E. Vidal. An algorithm for finding the nearest neighbour in (approximately) constant time. *Pattern Recognition Letters*, 4:145–157, 1986.

E. Vidal. New formulation and improvements of the nearest-neighbour approximating and eliminating search algorithm (AESA). *Pattern Recognition Letters*, 15:1–7, 1994.

E. Vidal, F. Casacuberta, y H. Rulot. Is the dtw "distance" really a metric? an algorithm reducing the number of dtw comparisons in isolated word recognition. *Speech Communication*, (4):333–344, 1985.

P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. En *Proceedings of the ACM-SIAM Symposium on Discrete Algorithms*, pp. 311–321, 1993.

P.N. Yianilos. Excluded middle vantage point forestsfor nearest neighbour search. En *DIMACS Implementation Challenge, ALENEX'99*, Baltimore, 1999.