RNPS No. 2142
ISSN 2072-6287

SERIE AZUL

CENATAV
Centro de Aplicaciones de Tecnologías de Avanzada
MINISTERIO DE LA INDUSTRIA BÁSICA

REPORTE TÉCNICO
Reconocimiento de Patrones

Métodos para acelerar la búsqueda de los k vecinos más cercanos utilizando distancias y disimilaridades

Yenny Villuendas Rey, Mairín Rodríguez Echemendía Pérez, Miguel Angel Medina Pérez, Milton García Borroto, José Ruiz Shulcloper

RT_007 **Diciembre 2008**

7ma. No. 21812 e/218 y 222,
Rpto. Siboney, Playa;
Ciudad de La Habana.
Cuba. C.P. 12200
www.cenatav.co.cu

Métodos para acelerar la búsqueda de los k vecinos más cercanos utilizando distancias y disimilaridades

Yenny Villuendas Rey, Mairín Rodríguez Echemendía Pérez, Miguel Angel Medina Pérez, Milton García Borroto, José Ruiz Shulcloper

Facultad de Informática y Centro de Bioplantas UNICA, Carretera a Morón km 9 ½, Ciego de Ávila, Centro de Aplicaciones de Tecnología de Avanzada, 7a #21812 e/ 218 y 222, Siboney, Playa, Habana, Cuba
yennyv@bioplantas.cu

RT 007 CENATAV
Fecha del camera ready: Diciembre 2008

Resumen: El presente Reporte Técnico aborda la problemática de la búsqueda de forma acelerada del vecino más cercano en espacios vectoriales y métricos y del vecino menos disimilar en espacios no métricos. Se realiza un estudio crítico de varios de los métodos más representativos reportados en la literatura y se presentan sus pseudocódigos. Además, se desarrolló una biblioteca de clases que permite la experimentación numérica con estos métodos. Los resultados obtenidos con bases de datos reales permitieron arribar a conclusiones acerca de su desempeño.

Palabras clave: Vecino más cercano, búsqueda del vecino más cercano, vecino más similar, búsqueda del vecino más similar.

Abstract: This Technical Report is about fast nearest neighbour finding in vector and metric spaces and fast less dissimilar neighbours finding in non metric spaces. Several of the most representative methods in the literature are studied, and its pseudo codes are given. Also, we developed a class library for numeric experimentation. The performance of these methods was evaluated in experimental results with real databases.

Keyword: Nearest neighbor, nearest neighbor search, most similar neighbor, most similar neighbor search.

Contenido

Notación

Se considerará que los conjuntos y las listas comenzarán con letra mayúscula, las variables con letra minúscula y las funciones se encontrarán en cursiva.

$\chi_i(o)$	Valor del rasgo χ_i en el objeto *o*
$V[i]$	Objeto de la lista *V* en la posición i.
$first(V)$	Devuelve el primer elemento de la lista *V*
$any(V)$	Devuelve cualquier elemento de la lista *V*
$random(V,n)$	Selecciona *n* elementos de *V* de forma aleatoria.
$count(V)$	Devuelve la cantidad de elementos de *V*.
$add(V,v)$	Adiciona el objeto v a la lista V, si no se encontraba en ésta.
$add(V,Z)$	Adiciona los objetos de la lista Z a la lista V, si no se encontraban en ésta.
$delete(V,v)$	Elimina el objeto v de la lista V.
$delete(V,Z)$	Elimina los objetos de la lista Z de la lista V.
$sortAsc(V,\prec)$	Ordena los elementos de la lista V de acuerdo a la relación de orden $\prec$.
$Element_x$	Elemento asociado al objeto x. Puede ser una lista, un valor numérico, etc.
$Var \leftarrow PDer$	Asignación. El valor de la expresión $PDer$ es asignado a la variable Var. El valor anterior de esta variable se pierde.
goto i	Salto al paso i.
*if Condición then Pasos*1	Si al evaluar la condición Booleana *Condición* el resultado es *verdadero*, se ejecuta $Pasos1$.
*if Condición then Pasos*1 *else Pasos*2	Si al evaluar la condición Booleana *Condición* el resultado es *verdadero* se ejecuta $Pasos1$, si no se ejecuta $Pasos2$.
While Condición *Pasos*	El contenido de *Pasos* es ejecutado si se cumple la *Condición*
foreach x in X *Pasos*	El contenido de *Pasos* es ejecutado para cada elemento x de la lista X.
End	Fin del algoritmo.
$root((V,\theta))$	Devuelve la raíz del árbol con nodos V y aristas θ
$getChildren((V,\theta),currentNode)$	Devuelve la lista de hijos de *currentNode* en el árbol (V,θ)
$clusterAlgorithm(V,i)$	Algoritmo de agrupamiento restringido, que devuelve los centros de los agrupamientos de los objetos en *V* al agruparlos en i agrupamientos

Introducción

En los últimos años, la existencia de sistemas de almacenamiento digital ha aumentado drásticamente, y la capacidad de crear y almacenar grandes bases de datos ha sido explotada en un amplio número de dominios y aplicaciones. Un ejemplo de esto es el desarrollo en Cuba de proyectos encaminados a interconectar los centros de asistencia médica, con el objetivo de acceder a datos de todo el país. Entre estos se encontrarán imágenes procedentes de radiografías, ultrasonidos, resonancias magnéticas, tomografías, entre otras. Esto permitirá al personal de salud mejorar el diagnóstico y pronóstico de enfermedades, por ejemplo: dado un nuevo caso, es beneficioso con frecuencia determinar el caso más parecido de los almacenados para realizar un mejor análisis de los síntomas e indicar el tratamiento adecuado, etc.

No sólo en la medicina se tienen bases de datos de imágenes. En otros dominios también existen grandes bases de datos de imágenes que son usadas para reconocer los objetos y la forma, en aplicaciones como por ejemplo, de estimación de las posturas del cuerpo y reconocimiento de caracteres ópticos, en el reconocimiento de rostros, reconocimientos de palabras, entre muchos otros problemas a los que no se le puede dar una solución trivial y se requiere hacer uso de las herramientas del Reconocimiento de Patrones (RP). Es de destacar que generalmente, las funciones de analogía entre imágenes tienen un costo computacional elevado [1].

Existen otros problemas como aplicaciones de optimización del uso de las redes de computadora donde también se necesita conocer el objeto más parecido a un objeto dado. Por ejemplo: cuando un ordenador realiza una petición para copiar un fichero de un contenido específico, el rendimiento de la red es optimizado si se puede identificar una localización de red cercana, usando alguna medida de proximidad de redes, que tenga el contenido deseado. También en el acceso a bases de datos temáticas, por ejemplo usuarios de una determinada base de datos (BD) necesitan con frecuencia, identificar y eficientemente obtener la BD en la que aparezca un contenido equivalente a un documento de entrada en particular.

Una de las familias de algoritmos de clasificación supervisada más popular es la familia del k-NN, clasificador de los k vecinos más cercanos (*Nearest Neighbors*, por sus siglas en inglés) Este clasificador, para asignarle la clase a un objeto nuevo, lo compara con todos los objetos de la matriz de entrenamiento, utilizando una función de distancia, y determina los k objetos de la matriz de entrenamiento más cercanos al objeto dado. Finalmente, le asigna la clase mayoritaria. Si hubiera un empate, se le asigna la clase de forma aleatoria [2]; pudiera asignársele las clases que provocaron el empate en caso de que el problema lo admita, de lo contrario, el clasificador podría abstenerse.

Este clasificador asume la existencia de una función de distancia para comparar los objetos por lo que estos tienen que estar representados en un espacio métrico. Una de sus ventajas es que no se usa ningún conocimiento explícito de la distribución subyacente de los datos [3], además, permite explicar por qué un objeto pertenece a determinada clase, retornando sus vecinos más cercanos. Este clasificador resulta además muy sencillo e intuitivo.

Pueden señalarse como principales desventajas de k-NN las siguientes:

1. Asume una función de distancia para comparar los objetos, y existen problemas reales donde la función de comparación entre objetos es una función determinada por la propia naturaleza del problema y no cumple las características de una distancia (definida positiva, simétrica y cumple la desigualdad triangular).

2. Asume que el espacio de representación de los objetos es un espacio métrico. En ciertos dominios, como en la clasificación de embriones, existen rasgos de diferente naturaleza. Algunos son numéricos, como el largo y el ancho del embrión, otros son categóricos, como el color, tipo de embrión y otros son Booleanos, como la presencia o no de agregados celulares en el embrión. Además, puede existir ausencia de información, es decir, ciertos valores de determinados rasgos no se conocen en un embrión dado. A este tipo de problemas, se les denomina problemas con datos mezclados e incompletos (MID, por sus siglas en inglés). En este tipo de problemas, hasta ahora no ha sido posible definir una función de distancia. Ejemplos de problemas de MID son: el pronóstico de zonas perspectivas de petróleo y el pronóstico de la rehabilitación de pacientes con afecciones de labio-paladar hendido [4, 5].
3. El clasificador de los k-NN tiene un costo computacional elevado. Para asignarle la clase a un nuevo objeto, debe primero compararlo con todos los objetos de la matriz de entrenamiento, lo que lo hace un clasificador lento. Retener todas las instancias en la memoria primaria también conlleva un costo de almacenamiento.

Una solución a los dos primeros problemas (distancias y espacios métricos) es extender la familia de los k vecinos más cercanos a los k vecinos más similares (k-MSN)[6-8]. La regla k-MSN, aunque no tiene las restricciones del k-NN y permite el trabajo con datos mezclados e incompletos y con funciones de similaridad que no son opuestas o inversas a una distancia, mantiene su elevado costo computacional.

También en la literatura se han reportado trabajos donde utilizan clasificadores k-NN con funciones de analogía que no cumplen las propiedades de las distancias, llamándole a estas funciones de diferente forma, usualmente como "distancias no métricas" [9-11]. Este asunto se tratará con mayor detalle en el acápite de Conceptos Básicos. Aunque estas funciones no son distancias, se les continúa llamando a estos clasificadores como k-NN, aunque éste en su definición utiliza funciones de distancia, como se mencionó anteriormente.

De manera general, la explosión en los volúmenes de datos ha hecho que resulte muy complejo hallar el (los) vecino(s) más cercano(s) a un objeto determinado, pues es necesario realizar un número muy grande de comparaciones, proporcional al tamaño del conjunto de datos, y con ello un alto consumo de recursos y tiempo, para alcanzar el resultado.

El problema de la explosión de los volúmenes de datos, en el contexto de los clasificadores k-NN y k-MSN ha sido abordado fundamentalmente de las siguientes formas:

- Seleccionando objetos de la matriz de entrenamiento [3, 6, 12-15].
- Seleccionando subconjuntos de rasgos [16-22].
- Seleccionando de forma conjunta o simultánea objetos y rasgos [23-27].
- Realizando búsquedas aceleradas (*fast*) de los k vecinos más cercanos [28-32]. Cabe señalar que hasta el momento no se han reportado métodos *fast* para clasificadores k-MSN.

De forma general no existe una correcta determinación del desempeño de los métodos reportados en la literatura. Como planteara Dasarathy en [33]: "Una evaluación experimental profunda de todas estas alternativas, utilizando diferentes conjuntos de datos, es necesaria para determinar realmente la eficiencia de estos procedimientos".

La eficiencia de estos métodos no sólo disminuye a medida que aumenta el tamaño de las bases de datos, sino que también se ve afectada, con el aumento de la dimensionalidad del espacio de representación de los datos. Además un aspecto muy importante es la complejidad de

la función de analogía empleada, pues mientras mayor sea su complejidad, mayor tiempo consume el proceso de búsqueda.

El trabajo está estructurado en las siguientes partes:

- **Introducción.**
- **Conceptos básicos.** Se presentan los conceptos básicos que permiten entender mejor el problema de la búsqueda rápida de los k vecinos de un objeto.
- **Evolución histórica de los algoritmos para el cálculo de los k vecinos de forma acelerada:** se presentan de manera general los principales algoritmos reportados en la literatura, en orden cronológico, lo que permite entender cómo ha sido el desarrollo de estos métodos. En aras de la comprensión de este reporte, se dividirá esta sección en dos partes: primero, se abordarán los métodos que utilizan funciones de distancias y posteriormente, aquellos que utilizan funciones que no cumplen con las características de las distancias.
- **Taxonomía de los métodos de búsqueda de los vecinos de un objeto de forma rápida.** Varias taxonomías son explicadas, las que ayudan a entender los métodos y seleccionar el más apropiado para un problema en particular.
- **Resultados experimentales:** en esta sección se presenta la comparación numérica efectuada entre varios de los métodos reportados, utilizando bases de datos del Repositorio de la Universidad de California en Irvin (UCI) [34].
- **Conclusiones.**
- **Referencias.**

El presente reporte dista mucho de ser exhaustivo, pues el tema de los métodos para acelerar la búsqueda del vecino más cercano está muy lejos de estar cerrado. Continúan apareciendo frecuentemente artículos con nuevas ideas, o mejoras de ideas anteriores. Además, cada método ha recibido a lo largo del tiempo una serie de mejoras, versiones y generalizaciones, de las cuales enunciamos aquí sólo algunas.

1 Conceptos básicos

1.1 Funciones de analogía

Los autores de este reporte no han podido encontrar en la literatura una definición formal de disimilaridad. Diversos autores utilizan este término, por ejemplo [35, 36] pero sin ofrecer una definición formal. De manera general, cuando se utiliza el término disimilaridad, se hace referencia a un grupo de funciones, que no tienen las restricciones de las distancias (definidas positivas, simétricas y cumplen la desigualdad triangular), y cuyo valor decrece en la medida en que las descripciones de los objetos que se comparan son más parecidas.

Como se mencionó anteriormente, existe una cierta tendencia a denominar a estas funciones como “distancias no métricas”, “distancias que no cumplen la desigualdad triangular”, “distancias no simétricas”, etc. [9-11, 28]. No consideramos correctas estas definiciones, pues a nuestro juicio, se altera con ellas la propia definición de distancia, y preferimos utilizar el término disimilaridad, más general, para hacer referencia a este tipo de funciones. Cabe señalar

que en la literatura [28, 32, 37] en métodos donde no se utilizan funciones de distancias para la comparación de los objetos, se utiliza el término "más cercano" para referirse al vecino menos disimilar. En este reporte, para hacer una distinción entre estos casos y aquellos en los que realmente se puede hablar de cercanía, se utilizará el término de vecino "menos disimilar" para evitar confusiones.

2 Evolución histórica de los algoritmos de búsqueda de los k vecinos de forma acelerada

2.1 Algoritmos que utilizan funciones de distancia

Para la reducción del costo computacional en el k-NN, una de las alternativas es utilizar propiedades de la función de distancia que se utiliza para comparar los objetos, para limitar la cantidad de comparaciones, es decir, no sería necesario para conocer cuáles son los k vecinos más cercanos de un objeto compararlo con cada uno de los objetos de la matriz de entrenamiento, sino sólo con algunos.

El primer intento de disminuir el costo computacional de la búsqueda de los k vecinos más cercanos, a través de la eliminación de cálculos de distancias fue en 1970 cuando Fisher y Patrick propusieron la realización de un preprocesamiento de la matriz de objetos, reordenando los objetos para eliminar el cálculo de algunas distancias (citado por [38]). En 1975 se publican en la IEEE dos trabajos relacionados con el tema: uno realizado por Fukunaga y Narendra [38] y otro por Friedman, Basket y Shustek (citado por [33]). El algoritmo propuesto en [38] ha sido extensamente citado en la literatura a lo largo de los años, y en comparaciones realizadas por diferentes autores ha mostrado un buen desempeño, a la altura de otros mucho más recientes. Es por esto que se detallará su funcionamiento a continuación.

El método consta de dos fases: primero la matriz de objetos es descompuesta jerárquicamente en subconjuntos disjuntos, el resultado de esta descomposición es representado por una estructura de árbol; y segundo, al árbol resultante se le aplica el algoritmo de Branch and Bound para buscar los k vecinos más cercanos a un objeto. Para ello, se usa la desigualdad triangular, para podar aquellas ramas que no conduzcan a encontrar algún vecino. Este método asume que los datos están representados en un espacio vectorial pues realiza operaciones de suma y multiplicación por un escalar entre los objetos.

2.1.1 Algoritmo B&B

Entradas:

T	Matriz de objetos
y	Objeto consulta
k	Número de vecinos
$d : E \times E \rightarrow \Re$	Función para calcular las distancias entre dos objetos definidos en E
h	Cantidad de niveles del árbol

$clusterAlgorithm$ Algoritmo utilizado para agrupar el conjunto de datos

c Número de agrupamientos

Pasos:

1	$K \leftarrow []$ $D \leftarrow []$	
2	$for\ \ i : 1..k$ $\quad add(K, null)$ $\quad add(D, \infty)$	Se inicializan las listas de los k vecinos más cercanos y sus distancias a y, con $null$ e ∞, respectivamente.
3	$(V, \theta) \leftarrow buildTree(T, h, clusterAlgorithm, c)$	Se construye el árbol.
4	$currentLevel \leftarrow 0$	
5	$d^* \leftarrow \infty$	Inicialmente se asume que la menor distancia es infinita, para luego ir buscando las menores.
6	$currentNode \leftarrow root((V, \theta))$	
7	$ActiveList \leftarrow []$	
8	$for\ \ i : 0..levelCount - 1$ $\quad add(ActiveList, [])$	Se inicializa una lista de nodos vacía por cada nivel.
9	$currentLevel \leftarrow currentLevel + 1$	
10	$Children \leftarrow getChildren((V, \theta), currentNode)$	Se obtiene la lista de hijos del nodo actual.
11	$foreach\ \ p\ in\ Children$ $\quad add(ActiveList[currentLevel], p)$	Se actualiza la lista con los nodos del nivel actual.
12	$foreach\ \ p\ in\ ActiveList[currentLevel]$ $\quad d_p \leftarrow d(p, y)$	Para cada uno de los nodos del nivel actual se calcula su distancia a y.
13	$foreach\ \ p\ in\ ActiveList[currentLevel]$ $\quad if\ \ d_p > d^* + r_p\ \ then$ $\quad\quad delete(ActiveList[currentLevel], p)$	Se aplica la regla de poda, basada en la desigualdad triangular.
14	$if\ \ count(ActiveList[currentLevel]) = 0\ \ then$ $\quad currentLevel \leftarrow currentLevel - 1$ $\quad if\ \ currentLevel = 0\ \ then$ $\quad\quad goto\ 15\ else\ goto\ 13$ $else$ $\quad currentNode \leftarrow \underset{\forall\ p \in ActiveList[currentLevel]}{\arg Min} \{d_p\}$ $\quad delete(ActiveList[currentLevel], currentNode)$ $\quad if\ \ currentLevel \neq h\ \ then$ $\quad\quad goto\ 9$	Si se terminan los nodos de la lista en el nivel actual, se sube de nivel, hasta llegar a la raíz (nivel 0). En caso contrario, se selecciona el nodo más cercano a y del nivel actual, y se elimina de la lista.

	else
	foreach x *in* $Associated_{currentNode}$
	if $d_{currentNode} \leq S_x + d^*$ *then*
	$s \leftarrow d(x,y)$
	$insertsorted(x,K,s,D)$
	$d^* \leftarrow D[k-1]$
	goto 13
		Si el nivel actual no es el más profundo del árbol, se retorna a 9, en caso contrario, para cada uno de los objetos asociados al nodo actual, se verifica si es un potencial vecino más cercano. Si se cumple la condición se calcula la distancia.
		Se actualizan la lista de los k vecinos más cercanos y d^*. Se utiliza el procedimiento $insertsorted(s,K,dsy,D)$ y se verificarán los nodos que queden para el nivel actual, pero con el nuevo valor de d^*.

15	*end*	

Procedimiento *buildTree*

Entradas:

T	Matriz de objetos
h	Cantidad de niveles del árbol
$clusterAlgorithm$	Algoritmo utilizado para agrupar el conjunto de datos
c	Número de agrupamientos

Pasos:

1	$V \leftarrow []$ $\theta \leftarrow []$	Conjunto de vértices y conjunto de aristas del árbol inicialmente vacíos.
2	$(C,\Omega) \leftarrow clusterAlgorithm(T,1)$	Se obtiene el centro del agrupamiento de la matriz de objetos, este pasa a ser la raíz del árbol.
3	$add(V,\Omega[0])$	Se comienza a construir el árbol por la raíz.
4	$Associated_{V[0]} \leftarrow C$	Se le asocia el conjunto de objetos a la raíz.
5	$highest \leftarrow 0$	
6	$S_{V[0]} \leftarrow []$	
7	*foreach* p *in* C $s \leftarrow d(p,V[0])$ $add(S_{V[0]},s)$ *if* $s > highest$ *then* $highest \leftarrow s$	A cada objeto se le asocia el conjunto de distancias de ellos a la raíz. Se busca la mayor distancia, que inicialmente era 0.
8	$r_{V[0]} \leftarrow highest$	Se asocia al centro la distancia al objeto más lejano de su agrupamiento.
9	$innerBuildTree((V,\theta),T,h,0,V[0],clusterAlgorithm,c)$	Se construye el interior del árbol.
10	*return* $((V,\theta))$	Se devuelve el árbol formado.

Procedimiento *innerBuildTree*

Entradas:

(V,θ)	Árbol
T	Conjunto a agrupar
h	Cantidad de niveles del árbol
currentLevel	Nivel por el que se está iterando
currentNode	Nodo padre
clusterAlgorithm	Algoritmo utilizado para agrupar el conjunto de datos
c	Número de agrupamientos

Pasos:

1	*if currentLevel* $\geq h$ *then goto* 6	Si se completaron todos los niveles no se continúa.
2	$(C,\Omega) \leftarrow clusterAlgorithm(T,c)$	Se obtienen c agrupamientos, en C, con sus centros, en Ω.
3	$currentlevel \leftarrow currentlevel + 1$	
4	$i \leftarrow 0$	
5	*foreach* x *in* Ω $add(V,x)$ $\theta \leftarrow \theta + \{(currentNode, x)\}$ $Associated_x \leftarrow C[i]$ $highest \leftarrow 0$ $S_x \leftarrow []$ *foreach* p *in* $C[i]$ $s \leftarrow d(p,x)$ $add(S_x, s)$ *if* $s > highest$ *then* $highest \leftarrow s$ $r_x \leftarrow highest$ $innerBuildTree((V,\theta), C[i], h, currentLevel, x, clusterAlgorithm, c)$ $i \leftarrow i + 1$.	Los centros de los agrupamiento se adicionan como los nuevos vértices a V. Se crean conexiones entre el nodo actual y cada uno de los nuevos vértices. A estos últimos se les asocian el conjunto de objetos de su agrupamiento, el conjunto de distancias del centro a cada objeto del agrupamiento y el radio. Se asocia al centro la distancia al objeto más lejano de su agrupamiento.
	Se continúa construyendo el árbol en profundidad	
6	*end*	

2.1.2 Valoración

Ventajas: Según los autores, este algoritmo requiere el cálculo de un número de distancias proporcionalmente menor a medida que el cardinal de la matriz de objetos aumenta. El preprocesamiento no es muy costoso en cuanto al tiempo de cálculo, es asequible incluso cuando

el número de objetos del árbol de búsqueda es menor que el número de objetos consulta, e incluso siendo iguales se pueden obtener resultados beneficiosos.

Desventajas: Según los autores, un factor dominante que afecta la eficiencia del algoritmo es el promedio de la cantidad de nodos por subclases finales, y el número total de nodos en el árbol: encontrar un equilibrio entre ambas medidas es una tarea difícil. Depende de la estructuración que tenga el árbol de búsqueda y de la selección del subconjunto de objetos asociados a cada nodo.

Comentarios: Para generar el árbol se descompone la matriz de objetos en conjuntos disjuntos utilizando el $c-means$, aunque enuncia que pueden ser usados otros métodos de agrupamiento. Presupone la existencia de un espacio vectorial.

En 1976, Yunck [39] propone un algoritmo donde utiliza hiperfiguras para buscar los vecinos dentro de ellas a través del cálculo de las distancias entre los objetos, pero sólo es aplicable en espacios con métricas de Minkowski (ver ecuación 1).

$$\rho_p(x,y)=\left(\sum_{k=1}^{n}\left|y_k-x_k\right|^p\right)^{\frac{1}{p}} \qquad \textbf{(1)}$$

En 1983 Miclet y Dabouz [40] proponen una taxonomía (la primera encontrada por nosotros) para los métodos de búsqueda rápida de los k-NN y presentan un algoritmo similar al de Fukunaga y Narendra [38], pero aplican otro algoritmo de agrupamiento.

Como parte del desarrollo de estos métodos en 1986 Vidal-Ruiz propone el AESA [31, 41], que solo es capaz de hallar un único vecino. Este método (y sus múltiples variantes) forman parte de los algoritmos de la categoría de "métodos que utilizan objetos ventajosos" [31]. Estas estrategias consisten en obtener un subconjunto de la matriz de entrenamiento (objetos ventajosos) y utilizarlos como base de las estrategias de poda, todas basadas en la desigualdad triangular. En 1994, se realiza una nueva formulación del AESA original [42]. Este método se basa en el uso combinado de las estrategias "Primero el mejor" y "Branch and Bound". Esta estrategia se apoya en una fuerte función de cota inferior, basada en la desigualdad triangular, con el fin de seleccionar sucesivamente el objeto candidato para el cálculo de la distancia y para la poda de aquellos objetos cuya evaluación sea mayor que la menor distancia encontrada hasta el momento. Este método es menos costoso computacionalmente que el AESA original [42]; permite una reducción significativa de los costos adicionales (no asociados al cálculo de la distancia), realiza una representación del algoritmo más clara y compacta y aunque la mejora realizada es ligera, resulta consistente pues requiere calcular un menor número de distancias. La complejidad de espacio en memoria para almacenar las distancias es de orden cuadrático y esto limita la aplicabilidad del algoritmo para grandes conjuntos de datos y con esto, el número de problemas en los que puede ser aplicado [43]. En los experimentos mostrados en el artículo original el rendimiento del algoritmo disminuye con el aumento de las dimensiones del espacio de representación de los datos.

Este algoritmo fue extendido en [44] para calcular los k vecinos más cercanos. Esta extensión consiste en el uso de una lista ordenada de los actuales vecinos más cercanos y su distancia al objeto consulta que es actualizada cada vez que se encuentre un objeto cuya distancia sea menor que la del k-ésimo vecino más cercano. Se le pudieran hacer ligeras modificaciones para obtener todos los vecinos más cercanos a un objeto (suponiendo que todos ellos tienen el mismo

valor de distancia con respecto a éste). Se detallará el funcionamiento de esta extensión, llamada KAESA, por su importancia dentro de estos métodos.

Esta estrategia, al igual que [42] se basa en una fuerte función de cota inferior, basada en la desigualdad triangular, ambas con el fin de seleccionar sucesivamente el objeto candidato para el cálculo de la distancia ("*Approximating*"). La función de cota inferior también es usada para la poda de aquellos objetos cuya evaluación sea mayor que la menor distancia encontrada hasta el momento ("*Eliminating*").

2.1.1 Algoritmo KAESA

Entradas:

T	Matriz de objetos
y	Objeto consulta
k	k vecinos más cercanos
$d : E \times E \to \Re$	Función para calcular las distancias entre dos objetos definidos en E

Pasos:

1	$A \leftarrow T$	
2	*foreach x in A* $G_x \leftarrow 0$	Se inicializa en cero el valor de la función de cota inferior asociado a cada objeto.
3	$s \leftarrow any(A)$	
4	$K \leftarrow []^k$	Lista de los k vecinos más cercanos.
5	$D \leftarrow []^k$	Lista de las distancias de los k vecinos más cercanos a y.
6	$nc \leftarrow 0$	
7	*While* $count(A) > 0$ $dsy \leftarrow d(s, y)$ $delete(A, s)$ $nc \leftarrow nc + 1$	Se calcula la distancia del objeto que se procesa al objeto consulta, y se elimina de la lista auxiliar *A*. Si aún no se ha llenado *K*, se inserta el objeto que se procesa y su distancia a y. Si no, si la distancia

	if $count(K) < k$ *then*
	  insertsorted(s, K, dsy, D)
	else
	  if $dsy < D[k]$ *then*
	  insertsorted(s, K, dsy, D)
	$s' \leftarrow s$
	$s \leftarrow A[0]$
	foreach p *in* A
	  $G_p \leftarrow \max\left(G_p, \left\|D_{p,s'} - dsy\right\|\right)$
	  if $\left(G_p \geq D[k] \wedge count(K) = k\right)$*then*
	  delete(A, p)
	  else
	  if $G_p < G_s$ *then*
	  $s \leftarrow p$
		calculada en la iteración actual es menor que la distancia del k-ésimo vecino (el de mayor distancia de los k más cercanos encontrado hasta el momento), se actualizan K y D. El procedimiento *insertsorted* es el definido para el algoritmo B&B
		Se le asigna a s el primero objeto de la lista *A* y para cada objeto en *A* se actualizan los valores de la función de cota inferior asociado a cada objeto.
		
		Para cada objeto en a, se halla la función de cota inferior (basada en la desigualdad triangular).
		
		Si el valor de la función de cota inferior de un objeto es mayor que la distancia del k-ésimo vecino de la iteración actual y ya está llena K, se puede eliminar este objeto. Esto garantiza después de iterar por todos los objetos de A que s sea el objeto de menor valor de la función de cota inferior. Esta heurística asume que este objeto posiblemente sea el más cercano al objeto y.

8	*end*	

2.1.2 Valoración

Ventajas: Determina el vecino más cercano en un tiempo constante promedio de forma asintótica. Es computacionalmente menos costoso que el AESA original [41]; permite una reducción significativa de los costos adicionales (no asociados al cálculo de la distancia), realiza una representación del algoritmo más clara y compacta y aunque la mejora realizada es ligera, resulta consistente pues requiere calcular un menor número de distancias.

Desventajas: La complejidad de espacio en memoria para almacenar las distancias es de orden cuadrático y esto limita la aplicabilidad del algoritmo para grandes conjuntos de datos y con esto, el número de problemas en los que puede ser aplicado [43, 45]. En los experimentos mostrados en el artículo el rendimiento del algoritmo disminuye con el aumento de las dimensiones del espacio de representación de los datos.

Comentarios: Presenta un costo computacional cuadrático para el cálculo de las distancias en la etapa de preprocesamiento. Presupone la existencia de un espacio métrico.

También en 1994 es reportado el algoritmo LAESA por Micó-Andrés, Oncina-Carratalá y Vidal-Ruiz [45], para la búsqueda rápida del 1-NN. En este método, a diferencia del AESA y el KAESA, el costo computacional para el cálculo de las distancias en la etapa de preprocesamiento y la complejidad de espacio en memoria para almacenarlas, son lineales con respecto al número de objetos. Este método tiene dos procedimientos fundamentales. Un procedimiento para seleccionar los "Prototipos base" (subconjunto de objetos de la matriz de objetos) y simultáneamente calcular las distancias entre estos y el resto de los objetos. Y el otro, para realizar la búsqueda del vecino más cercano. Este último es una extensión de la estrategias aplicadas en [42]. Esta extensión radica fundamentalmente en el uso de dos funciones auxiliares *choice* y *condition*. La primera ayuda a decidir si se debe seleccionar un prototipo base o uno

no base para continuar el proceso de búsqueda y la segunda controla la eliminación de los prototipos base. En [43] se formulan nuevos criterios para las funciones *choice* y *condition*, con el objetivo de disminuir el leve aumento de distancias que calcula el algoritmo si se utiliza la primera variante. Esta segunda variante permite la elección de objetos que no son base desde el principio, pero no frecuentemente. Con él los vecinos muy cercanos al objeto se encuentran rápidamente.

En el 2002 se le realiza una extensión al LAESA para encontrar los k vecinos más cercanos a un objeto [29]. Esta extensión, denominada KLAESA, calcula un número de distancias que crece junto con el valor de k y ligeramente con respecto al tamaño del conjunto de objetos de la matriz de objetos (n); como k es menor que n, siempre este número va a ser menor que la búsqueda exhaustiva, y se mantienen los requerimientos de espacio y tiempo lineales al igual que el LAESA. Sus autores proponen su uso cuando se tienen funciones de distancia que requieren mucho tiempo para el cálculo y grandes conjuntos de datos. Esta extensión se detallará a continuación. Otras variantes y modificaciones a estos algoritmos pueden ser encontradas en [30, 46, 47].

2.1.3 Algoritmo KLAESA

Notación y Definiciones

G_x Es el valor de la función de cota inferior asociado al objeto x

$D_{x,y}$ Es la distancia entre los objetos x y y

Entradas:

T Matriz de objetos

B Prototipos Base

y Objeto consulta

k k vecinos más cercanos

$d : E \times E \to \Re$ Función para calcular las distancias entre dos objetos definidos en E

Pasos:

1	$A \leftarrow T$ $B \leftarrow BP - Selection(T)$	Se hallan los prototipos base (*B*).
2	*foreach* x *in* A $G_x \leftarrow 0$	Se inicializa en cero el valor de la función de cota inferior asociado a cada objeto.
3	$s \leftarrow any(B)$	
4	$K \leftarrow []^k$	Lista de los k vecinos más cercanos.
5	$D \leftarrow []^k$	Distancias de los k vecinos más cercanos a y.
6	$nc \leftarrow 0$	

7

```
While count(A) > 0
   dsy ← d(s, y)
   delete(A, s)
   nc ← nc + 1
   if count(K) < k then
         insertsorted(s, K, dsy, D)
      else
         if dsy < D[k] then
            insertsorted(s, K, dsy, D)
   q ← null
   b ← null
   gq ← ∞
   gb ← ∞
   foreach p in A
         if s in B then
            G_p ← max(G_p, |D_{p,s} − dsy|)
         gp ← G_p
```

Mientras que A no esté vacía.

Si aún no se ha llenado K pues se inserta s y su distancia a y. Si no, se verifica si la distancia calculada en la iteración actual es menor que la distancia del k-ésimo vecino (el de mayor distancia de los k más cercanos) encontrado hasta el momento, se actualizan K y D.

q es el supuesto vecino más cercano en A.
b es el supuesto vecino más cercano en B.
gb y *gq* son los valores de la función de cota inferior asociados a *b* y *q*, respectivamente.

Para cada objeto, se chequea si es un prototipo base, y de serlo se actualizan los valores de la función de cota inferior asociada.

```
         if p in B then
            if (g_p ≥ D[k] ∧ count(K) = k ∧ condition) then
               delete(A, p)
```

Si es un prototipo base, y si el valor de la función de cota inferior de p es mayor que la distancia del k-ésimo vecino encontrado hasta la iteración actual, ya está llena K y además se cumple $condition$, se puede eliminar este objeto de A.

```
            else
               if gp < gb then
                     b ← p
                     gb ← gp
                  else
                     if (gp ≥ D[k] ∧ count(K) = k) then
                        delete(A, p)
                           else
                              if gp < gq then
                                    q ← p
                                    gq ← gp
```

De lo contrario, si el valor de la función de cota inferior de *p* (que no es base) es mayor que la distancia del k-ésimo vecino encontrado hasta la iteración actual y ya está llena *K* se puede eliminar este objeto de *A*

	$s \leftarrow choice(b, q)$	Procedimiento para seleccionar b o q en dependencia de la heurística determinada.
8	*end*	

Algoritmo $BP - Selection$
Entradas:

P Matriz de objetos
m Número de prototipos base
$d : E \times E \rightarrow \Re$ Función para calcular las distancias entre dos objetos definidos en E

Pasos:

1	$b' \leftarrow any(P)$	
2	$add(B, b')$	Primer prototipo base.
3	*foreach* x *in* P $A_x \leftarrow 0$	Se inicializa en cero el valor de las distancias acumuladas asociadas a cada objeto.
4	*While* $count(B) < m$ $highest \leftarrow 0$ $b \leftarrow b'$ *foreach* p *in* P $D_{p,b} \leftarrow d(p, b)$ $A_p \leftarrow A_p + D_{p,b}$ *if* $A_p > highest$ *then* $b' \leftarrow p$ $highest \leftarrow A_p$ $add(B, b')$	Mientras no se obtengan los m (número especificado por el usuario) prototipos base. Determina la distancia de todos los objetos al prototipo base. Se actualizan los valores de las distancias acumuladas Esto garantiza que al iterar por todos los objetos, b' sea el de mayor valor de distancia acumulada. Siguiendo la heurística, este objeto será el que mayor valor de las distancias acumuladas tenga (el más alejado)
5	*end*	

Procedimiento $choice$ (Variante 1)
Entradas:

b Prototipo base
q Prototipo no base

Pasos:

1	*if* $b \neq null$ *then* b *else* q	Si después de iterar por todos los objetos de A que sean prototipos base, el objeto b toma valor entonces se continua la búsqueda por él, en caso contrario, se prosigue con el objeto no base que posiblemente sea el más cercano a y . Este procedimiento garantiza que la búsqueda se realice primero por los prototipos base. Esta primera variante fue propuesta en [45]

Procedimiento $choice$ (Variante 2)
Entradas:

b Prototipo base
q Prototipo no base

Pasos:

1	*if* en las *rc* iteraciones anteriores *q* tiene el mismo valor, *then q else b.*
	Esta variante se corresponde con un nuevo criterio propuesto en [43] para eliminar el leve aumento del promedio de distancias calculadas que realiza el LAESA con respecto al AESA. Este método requiere de una lista que almacene el estado de q en las rc iteraciones anteriores, pero para evitar que el proceso se vuelva engorroso y no hacerle grandes modificaciones al algoritmo para su explicación se va a prescindir de esta lista. Si en las rc iteraciones anteriores el valor de q no cambió entonces se continua la búsqueda por él, en caso contrario se prosigue con b .

Procedimiento *condition* (Variante 1)
Pasos:

1	*if* $nc > \frac{m}{k}$ *then* *return true* *else return false*	La función *condition* garantiza que primero se eliminen los prototipos que no sean base. Este procedimiento corresponde a la familia de las funciones Booleanas y fue propuesta en [45].

Procedimiento *condition* (Variante 2)
Pasos:

1	*if no se descartó ningún objeto en el paso anterior then*
	return true
	else
	return false
		Esta nueva variante fue propuesta en [43] para junto con la Variante 2 de la función *choice* disminuir las distancias calculadas que realiza el LAESA. Esta función permite la eliminación de los prototipos base si en el paso anterior no se eliminó ningún objeto.

2.1.4 Valoración

Ventajas: La complejidad de espacio en memoria para almacenar las distancias y el costo computacional para el cálculo de las mismas en la etapa de preprocesamiento son lineales con respecto al número de objetos; porque sólo requiere de las distancias entre el subconjunto de prototipos base (que es más pequeño que el conjunto de entrenamiento original) y todos los objetos. Logra una eficiencia en la búsqueda muy cercana a la del AESA original. Ayuda a reducir los costos adicionales (no asociados al cálculo de la distancia). Análogamente a las versiones anteriores del AESA [44] determina el vecino más cercano en un tiempo constante promedio asintóticamente. Este algoritmo es una buena alternativa cuando el cálculo de la distancia requiere mucho tiempo y el conjunto de objetos es grande. Se proponen segundas variantes para las funciones *choice* y *condition* , con el objetivo de disminuir el leve aumento de distancias que calcula el algoritmo si se utiliza la primera variante [45].

Desventajas: Para grandes conjuntos de datos y cálculos de distancia no muy complejas, los costos adicionales pueden ser mayores que el costo de realizar una búsqueda exhaustiva. Este cuello de botella de todas las estrategias relacionadas con el AESA limita su uso en un gran número de problemas [43, 45]. Se alcanza una reducción de los costos en memoria, pero esto

produce un pequeño incremento del número de distancias calculadas con respecto a las realizadas por el AESA.

Comentarios: Este método encuentra los k vecinos más cercanos a un objeto, calculando un número de distancias que crece junto con el valor de k y ligeramente con respecto al tamaño del conjunto de objetos (n), como k es menor que n, siempre este número va a ser menor que la búsqueda exhaustiva. Para el caso que se use la primera variante de las funciones *choice* y *condition* se aprecia que el rango óptimo de los prototipos base se amplía y la libertad del algoritmo para eliminarlos crece con el aumento de los valores de k (esta variable es la usada en *condition*, no el número de vecinos más cercanos). Estos experimentos muestran que para grandes conjuntos de datos y para $k = 1$, no hay variaciones significativas en el número óptimo de prototipos bases. Según los autores en [45], el nuevo método calcula ligeramente más distancias que el AESA; específicamente en los experimentos se muestra que este número es menor que 1,5 veces el número de distancias calculadas por el AESA. La eficiencia de este algoritmo depende del número de prototipos base y de su localización con respecto a los demás. Aunque en el método, los costos adicionales (no asociados al cálculo de la distancia) son lineales y no parecen tener mucha importancia, para la aplicación real donde estén involucradas distancias complejas y grandes conjuntos de datos, se origina un serio cuello de botella, si el número de objetos está por encima de los millones, según sus autores. Para solucionar este problema, la poda o eliminación de los objetos debe organizarse de forma tal que se pueda eliminar un gran conjunto de objetos en un solo paso y algunas posibles soluciones son planteadas en [43]. Experimentos simulados que son mostrados en [45] revelan que para grandes conjuntos de datos y un número fijo de prototipos base, el costo del método tiende a ser una constante pequeña e independiente del total de prototipos.

Con el nuevo criterio para *choice* y *condition* no crece el número de distancias calculadas con el aumento de la dimensionalidad sino que incluso hasta se pueden disminuir si los datos están agrupados (intervalos de tolerancia menores de 15%), según se aprecia en experimentos mostrados en [43]. Los intervalos de tolerancia muestran la cercanía del objeto consulta a sus objetos más cercanos, mientras menor sea el valor significa que más agrupados están. Esta segunda variante permite la elección de objetos que no son base desde el principio, pero no frecuentemente. Con él los vecinos que están muy cercanos al objeto se encuentran rápidamente. Presupone la existencia de un espacio métrico.

2.2 Algoritmos que utilizan funciones de disimilaridad

Cabe recordar que en estos métodos, aunque no se utilizan funciones de distancias para la comparación de los objetos, se utiliza el término "más cercano" para referirse al vecino menos disimilar. Para hacer una distinción entre este caso y aquellos en los que realmente se puede hablar de cercanía, se utilizará el término de búsqueda del vecino "menos disimilar" para evitar confusiones y mantener en lo posible el concepto al que se quiere hacer referencia.

La primera propuesta de un método de búsqueda rápida para el cálculo del vecino menos disimilar de un objeto en espacios no métricos es la de Faragó, Linder y Lugosi en 1994 [48]. Los autores definen "espacio de disimilaridad" como un conjunto D no vacío con una función $\rho : D \times D \to \Re$ en el que $\forall x, y \in D$ se cumple que:

1. $\rho(x,y) \geq 0$
2. $\rho(x,y) = 0 \Leftrightarrow x = y$
3. $\rho(x,y) = \rho(y,x)$

A este espacio de disimilaridad se le impone la restricción de tener una estructura geométrica y una dimensionalidad. Para ello se define un subconjunto acotado H del espacio de disimilaridad que debe cumplir que $\sup\{\rho(o_1,o_2) : o_1,o_2 \in H\} < \infty$. Además, sea D un espacio de disimilaridad y $\alpha \geq \beta > 0$, se dice que los puntos $z_1, z_2, \cdots, z_k \in D$ forman una base en el nivel (α,β) para un conjunto $H \subset D$ si se cumple $\forall x, y \in H$ que:

1. $\alpha\rho(x,y) \geq |\rho(x,z_i) - \rho(y,z_i)| \quad i = 1,\cdots,k$
2. $\max_{1\leq j\leq k} |\rho(x,z_j) - \rho(y,z_j)| \geq \beta\rho(x,y)$

El algoritmo trata de eliminar aquellos elementos que se encuentren alejados del objeto al que se le quiere hallar el vecino, utilizando para ello los elementos que forman una base. Según sus autores, cuando se cumplen las condiciones requeridas, el método logra reducciones similares a las de los métodos como el AESA que funcionan en espacios métricos y siempre encuentra el vecino deseado, es decir, es equivalente a la búsqueda exhaustiva, sin embargo, necesita de los elementos que forman la base, lo cual puede resultar muy difícil de obtener cuando se trabaja con datos limitados, donde no se conoce su distribución ni las características de su espacio de representación; y además, en algunas ocasiones el método puede no llegar a realizar reducción alguna en el número de comparaciones.

No es hasta casi 10 años más tarde que aparece otro método que no utiliza las propiedades de las funciones de distancia para el cálculo de los vecinos de un objeto. En el 2002 es propuesto el DynDex en [28], con el objetivo de obtener los k vecinos menos disimilares utilizando la función de disimilaridad DPF. Los autores se refieren a esta función como "una distancia no métrica, dinámica y parcial". Esta función (ver ecuación 2) no cumple con la desigualdad triangular y no es definida positiva y compara pares de objetos utilizando diferentes sub-espacios de rasgos. Es esta última característica a la que se le denomina dinámica, ya el conjunto de rasgos que se utiliza para compara los objetos varía para cada par de objetos.

$$DPF(x,y) = \sum_{\delta_i \in \Delta m} \delta_i \qquad \textbf{(2)}$$

donde $\delta_i = |\chi_i(x) - \chi_i(y)|$ y Δm es el conjunto formado por los menores m δ_i.

Este método consiste en la aplicación del CLARANS [49] y busca los representantes de cada agrupamiento. Para la búsqueda de los representantes se pueden utilizar varios esquemas. Se proponen cuatro de ellos: *Medoide* (el representante es el objeto más central del agrupamiento, es decir, el de menor valor de la función DPF con el resto de los objetos de su agrupamiento), *Selección aleatoria*, *Objetos atípicos* (para cada objeto del agrupamiento, se halla el valor de DPF del objeto a los representantes del agrupamiento y si es mayor que la máxima, se intercambian y se considera como el objeto más atípico), y *Selección correlacionada* (para cada objeto del agrupamiento, se halla la correlación del objeto a los representantes del agrupamiento y si el valor de DPF encontrado es menor que el mínimo, se intercambian y se

considera como el objeto más correlacionado). Luego de tener los agrupamientos y sus representantes, éstos se ordenan de acuerdo al valor de la función DPF de los representantes con respecto al objeto al que se le quieren buscar los vecinos y éstos se buscan en el agrupamiento del representante ganador (el primero en el orden realizado). El algoritmo ofrece la posibilidad de continuar buscando en otros agrupamientos, hasta que se obtenga el *recall* (ver ecuación 8) deseado.

Algoritmo *Dyndex*

Entradas:

$T = \{t_1, t_2, \cdots t_n\}$ Matriz de objetos

q Elemento al que se le quiere hallar los vecinos más cercanos

c Cantidad de agrupamientos que se desea obtener

Pasos:

1	$Clusters \leftarrow CLARANS(T, c)$	Se aplica al algoritmo de agrupamiento.
2	$foreach\ C \in Clusters$ $R_C \leftarrow Representant(C)$	Para cada uno de los agrupamientos C, se hallan sus representantes, siguiendo uno de los esquemas propuestos.
3	$foreach\ r \in R_c$ $d_r \leftarrow d(q, r)$	Se halla el valor de la función DPF del objeto q a cada uno de los representantes de los agrupamientos.
4	$Sort(Clusters, \prec)$	Se ordenan los agrupamientos ascendentemente. $X \prec Y \Leftrightarrow \min_{x \in R_X} d_x < \min_{y \in R_y} d_y$
5	$C \leftarrow first(Clusters)$	
6	$NN \leftarrow \arg\min_{u \in C} d(q, u)$	Se buscan los vecinos de forma exhaustiva en el primer agrupamiento. Si se desea una mayor precisión, pueden buscarse los vecinos en los siguientes agrupamientos.
7	*end*	

Procedimiento *CLARANS*

Entradas:

$d : E \times E \rightarrow \Re$ Función para calcular las disimilaridades entre dos objetos definidos en E

numlocal Número máximo de mínimos locales (iteraciones) que se desea obtener

maxNeighbor Número máximo de vecinos que serán visitados

c Cantidad de agrupamientos que se desea obtener

T Matriz de objetos

Pasos:

1	$i \leftarrow 1$	
2	$minCost \leftarrow \infty$	
3	$n \leftarrow 0$	
4	$while\ \ n < c$ $m \leftarrow any(T)$ $add(Current, m)$	Se busca un conjunto de medoides (vértice del grafo) de forma aleatoria y se añaden a la lista actual.
5	$foreach\ m \in Current$	Para cada medoide m , se buscan los

	$Cluster_m \leftarrow \{x \in T : d(x,m) = \min_{y \in Current} d(x,y)\}$	elementos de la matriz de objetos que pertenecen al agrupamiento representado por él.
6	$j \leftarrow 1$	
7	$index \leftarrow any(0..2)$ $m \leftarrow any(T)$ $S \leftarrow Current$ $S[index] \leftarrow m$	Se busca un vecino de *current* de forma aleatoria.
8	$m, x \in Current \cap S : m \in Current,\ x \in S$	Se buscan los medoides que diferencian a S de *current.*
9	$Cost_{m,x} \leftarrow 0$	

10	*foreach* $y \in T : y \neq x$
	$newM \leftarrow \arg \min_{q \in Current \setminus m} d(x,q)$
	if $(y \in Cluster_m)$ *then*
	if $d(y,x) \geq d(y,newM)$ *then*
	$Cost_{m,x} \leftarrow Cost_{m,x} + [d(y,newM) - d(y,x)]$
	else
	$Cost_{m,x} \leftarrow Cost_{m,x} + [d(y,x) - d(y,m)]$
	else
	if $d(y,x) < d(y,newM)$ *then*
	$Cost_{m,x} \leftarrow Cost_{m,x} + [d(y,x) - d(y,newM)]$
		Para cada elemento de la matriz de objetos:
		Se busca el medoide más cercano, excepto m. Si el elemento pertenece al agrupamiento de m y si al realizar el cambio de medoide no permanecería, se calcula el costo como la diferencia de disimilaridades.
		Si el elemento no pertenece al agrupamiento de m se calcula el costo como la diferencia de disimilaridades, suponiendo que cambia de agrupamiento.

11	*if* $(Cost_{m,x} < 0)$ *then*
	$Current \leftarrow S$
	goto 5
	else
	$j \leftarrow j + 1$
		Si el costo del intercambio es menor que cero, se intercambian los conjuntos y se regresa a 5. En otro caso, se incrementa el valor de j.

12	*if* $(j \leq maxNeignbor)$ *then*
	goto 6
	else
	if $Cost_{m,x} < minCost$ *then*
	$minCost \leftarrow Cost_{m,x}$
	$Output \leftarrow Current$
		Si no se han visitado todos los vecinos previstos, se regresa a 6.
		En otro caso, se analiza el costo. En caso de ser menor que el mínimo, se actualizan el costo mínimo y el mejor conjunto de medoides, *output*, que será la salida del algoritmo.

13	$i \leftarrow i + 1$	
14	*if* $(i < numlocal)$ *then* *goto* 2	Si el número de iteraciones no se ha alcanzado, se regresa a 2.
15	*end*	

2.2.1 Valoración

Ventajas: Es un método diseñado especialmente para el manejo de funciones de comparación no métricas y que pueden evaluar diferentes objetos utilizando diferentes subconjuntos de rasgos.

Desventajas: No siempre encuentra los vecinos de forma correcta, es decir, no es equivalente con la búsqueda exhaustiva.

Comentarios: El método está desarrollado para utilizar la función DPF, pero puede ser extendido (cambiando los criterios de comparación de los rasgos) a otras funciones de disimilaridad que al igual que la DPF, no cumplan con la desigualdad triangular y no sean definidas positivas. En los experimentos realizados por sus autores se muestra que entre los esquemas para seleccionar los representantes: medoide, aleatorio, atípico y correlacionado, los esquemas medoide y aleatorio mostraron los mejores resultados. Mientras menor sea el número de agrupamientos, entonces mayor será la cantidad de objetos de cada uno y por lo tanto más exacto el método. En los resultados experimentales los autores muestran que se obtiene un balance entre el costo de entrada-salida y el *recall*, para 100 conjuntos de agrupamientos, y que el funcionamiento de la función DPF es positivo para los valores de $m = 114$ y $r = 2$.

El algoritmo ClusterTree fue propuesto por Zhang y Srihari en el 2002 [50]. Este método, según sus autores, aunque está propuesto para una determinada función de disimilaridad, es aplicable al cálculo de los vecinos con cualquier función de disimilaridad. El algoritmo funciona en dos etapas: entrenamiento y clasificación. En la primera etapa, se construye un árbol y en la segunda, se realiza la búsqueda de los vecinos menos disimilares en los distintos niveles del árbol. Para construir el árbol, se calcula, para cada objeto *z*, su disimilaridad al vecino menos disimilar de otra clase γ_z, se hallan los vecinos menos disimilares de su clase que tienen un valor menor de disimilaridad que γ_z, estos vecinos formarán el conjunto ψ_z. Luego, se calcula el valor de ι_z como la cantidad de objetos en ψ_z. Los objetos se ordenan descendentemente de acuerdo a los valores de ι_z, y de acuerdo a este orden se van seleccionando para formar parte de los niveles del árbol. Inicialmente se construyen dos niveles, el *bottom level* y el *hyper level*. En estos dos niveles, los objetos son agrupados de acuerdo a sus clases. Posteriormente, en dependencia del umbral definido, se generarán nuevos niveles en el árbol, hasta obtener un nodo raíz. A los objetos que no se encuentran en el *bottom level*, se les denomina *hyper nodos* Para el cálculo de los vecinos, se recorre el árbol por las ramas en dependencia del valor de disimilaridad del objeto consulta con respecto a los nodos del árbol, hasta llegar al *bottom level*.

2.2.2 Algoritmo ClusterTree

Algoritmo $Find\ k-NN$

Entradas:

T	Matriz de entrenamiento
y	Objeto consulta.
k	Cantidad de vecinos que se desea

$d : E \times E \to \Re$ Función para calcular las disimilaridad entre dos objetos definidos en E.

Pasos:

1	$(V,\theta) \leftarrow TreeGeneration(T,d,\alpha)$	
2	$level = (V,\theta).count$	
3	$M \leftarrow (V,\theta)[level]$	Nodos en el nivel superior
4	$\zeta \leftarrow k$	
5	$nc = 0$	
6	$K \leftarrow []$ $D \leftarrow []$	Listas de los k vecinos menos disimilares y sus disimilaridades.
7	*foreach* x *in* M $d \leftarrow d(z,y)$ $insertSorted(d,D,z,L)$ $nc = nc + 1$ $level \leftarrow level - 1$	Se seleccionan los ζ nodos menos disimilares a y en el nivel superior. Se almacenan las disimilaridades y los nodos en listas paralelas.

8	*if* $level > 0$ *then*
	foreach x *in* L
	if $count(\psi_x) > 1$ *then*
	foreach z *in* ψ_x
	if $z \neq x$ *then*
	$d \leftarrow d(z,y)$
	$insertSorted(d,D,z,L)$
	$nc = nc + 1$
	goto 10
		Se calcula la disimilaridad entre y y los sub-nodos enlazados a los nodos de L. Se actualiza L con los ζ nodos más cercanos.

9	$H \leftarrow []$ *foreach* x *in* L *if* $d(x,y) < \gamma_x$ *then* $add(H,x)$	Se seleccionan los *hyper nodos* con disimilaridad a y menor que la menor disimilaridad hasta el momento.

	$cl \leftarrow class(H[0])$
	$difcl \leftarrow false$
	$i = 0$
	while $(!(difcl) \wedge (i < count(H)))$
	  if $class(H[i]) \neq cl$ *then*
	  $difcl \leftarrow true$
	  $i \leftarrow i + 1$
	  if $!(difcl)$ *then*
	  foreach x *in* H
	  foreach z *in* ψ_x
		Se verifica si todos los *hyper nodos* seleccionados son de la misma clase.

10	$d \leftarrow d(z, y)$ $insertSorted(d, D, z, K)$ $nc = nc + 1$ *return* K *else* *foreach* x *in* L *foreach* z *in* ψ_x $d \leftarrow d(z, y)$ $insertSorted(d, D, z, K)$ $nc = nc + 1$ *return* K	Si todos los *hyper nodos* seleccionados son de la misma clase, entonces los k vecinos menos disimilares se seleccionan de los sub-nodos enlazados a ellos. Si no, se seleccionan de los sub-nodos enlazados a los *hyper nodos* de L.
11	*end*	

Procedimiento *TreeGeneration*

Entradas:

T	Matriz de entrenamiento
$d : E \times E \rightarrow \Re$	Función para calcular las disimilaridad entre dos objetos definidos en E.
η	Constante

Pasos:

1	$A \leftarrow T$	
2	$V \leftarrow []$ $\theta \leftarrow []$	Conjunto de vértices y aristas del árbol, inicialmente vacíos.
3	$B \leftarrow []$	*bottom level* sin nodos
4	$H \leftarrow []$	*hyper level* sin nodos

5	*while* $count(A) > 0$
	  foreach z *in* A
	  $\gamma_z \leftarrow \infty$
	  foreach x *in* A
	  if $class(z) \neq class(x)$ *then*
	  $d = d(z, x)$
	  if $d < \gamma_z$ *then*
	  $\gamma_z \leftarrow d$
	  foreach x *in* A
	  if $class(z) = class(x)$ *then*
	  $d = d(z, x)$
	  if $d < \gamma_z$ *then*
	  $add(\psi_z, x)$
	  $\ell_z \leftarrow count(\psi_z)$
	  $Sort(A, \prec)$
	  $hn \leftarrow A[0]$
	  $add(B, \psi_{hn})$
	  $add(H, hn)$
	  foreach z *in* ψ_{hn}
	  $\theta \leftarrow \theta + \{(hn, z)\}$
	  $delete(A, hn)$
	  $delete(A, \psi_{hn})$
		Disimilaridad entre z y su vecino más cercano con diferente clase.
		Lista de vecinos de la misma clase de z pero con disimilaridad menor que la menor disimilaridad de los vecinos con clase contraria (hijos).
		Se ordena la matriz de entrenamiento descendentemente
		$X \prec Y \Leftrightarrow \underset{x \in A}{\ell_x} > \underset{y \in A}{\ell_y}$
		Hypernodo, objeto de mayor ℓ de A
		Se crean conexiones entre el hypernodo hn y cada uno de los objetos de ψ_{hn}

6	$add(V, B)$ $add(V, H)$	Se adicionan el *bottom level* y el *hyper level*, al conjunto de vértices.
7	$currentLevel \leftarrow H$ $stop \leftarrow$ *false*	

8	*while* $not(stop)$
	  $P \leftarrow []$
	  $T \leftarrow currentLevel$
	  $y \leftarrow currentLevel[0]$
	  if $count(currentLevel) = 1$ *then*
		Si el nivel actual tiene un solo nodo, se adiciona, a la lista del nivel superior.

	$add(P, y)$	

	else
	while $count(currentLevel) > 1$
	foreach x, p *in currentLevel*
	$d \leftarrow d(p, x)$
	if $d < \eta$ *then*
	$add(C, x)$
	$add(C, p)$
	$delete(currentLevel, x)$
	$delete(currentLevel, p)$
	$\min \leftarrow \infty$
	if $count(C) > 1$
	foreach x *in* C
	foreach z *in* T
	$d \leftarrow d(z, x)$
	if $d < \min$ *then*
	$cent \leftarrow z$
	$\min \leftarrow d$
	$\psi_{cent} \leftarrow C$
	$add(P, cent)$
	else
	$cent \leftarrow C[0]$
	$\psi_{cent} \leftarrow C$
	$add(P, cent)$
	if $count(P) \neq count(T)$ *then*
	$add(V, P)$
	foreach z *in* P
	foreach x *in* ψ_z
	$\theta \leftarrow \theta + \{(z, x)\}$
	if $count(P) \neq 1$ *then*
	$\eta \leftarrow \eta + 1$
	else
	$stop \leftarrow true$
		En caso contrario, se agrupan los objetos del nivel actual, cuya disimilaridad sea menor que el umbral.
		
		Si el grupo formado en el paso anterior tiene más de un objeto, se determina el objeto que minimiza las disimilaridades con respecto al resto y se adicionan a la lista del nivel superior.
		
		En caso contrario, se adiciona como centro el único elemento del grupo.
		
		Si ha habido cambios en la lista del nivel superior, se adiciona a las lista de vértices y se crean conexiones con los nodos del nivel anterior
		
		Si existe más de un objeto en el *huper level* se incrementa el valor de η hasta obtener un solo nodo.

9	*end*	

2.2.3 Valoración

Comentarios: El algoritmo reduce el costo computacional de la búsqueda de los vecinos, aunque con una pérdida de exactitud. Si se utilizan sus resultados para clasificar, este proceso se realiza mucho más rápido que utilizando métodos de condensación basados en árboles. Este algoritmo tiene costo computacional $m^2 \log m$, donde m es el número de nodos del árbol de agrupaciones [32].

3 Taxonomías de los métodos de búsqueda rápida de los k vecinos de un objeto.

3.1 Métodos para espacios vectoriales [51]

3.1.1 Esquemas no jerárquicos

Dividen el espacio de búsqueda en regiones que tienen la propiedad que la región a la que pertenece un objeto consulta puede ser identificada en un número constante de operaciones. Trabajan eficientemente en el indexado de espacios de pocas dimensiones (≤10), pero su eficiencia decae exponencialmente cuando la dimensionalidad es (>20). En el rango definido entre estos 2 valores, la eficiencia del método estará de acuerdo a las características de la base de datos [39].

3.1.2 Métodos particionados recursivamente

Dividen el espacio de búsqueda en pequeñas regiones progresivamente, las cuales dependen de que la matriz de objetos sea indexada. El resultado jerárquico puede ser bien representado por un árbol. Sufren "la maldición de la dimensionalidad", ya que generalmente se convierten en ineficientes para dimensionalidades mayores de 20, excepto en raros casos donde las bases de datos tienen una estructura particular [52].

3.1.3 Métodos basados en proyecciones

Varios métodos proyectan la base de datos sobre los ejes coordenados, manteniendo una lista para cada colección de proyecciones, y usan la lista para identificar rápidamente una región que contenga una hiperesfera de radio r centrada en el objeto consulta. Otros métodos proyectan la base de datos sobre un hiperplano apropiado de $d+1$ dimensiones, y buscan los vecinos más cercanos trazando una "línea apropiada" (detalles acerca de qué constituye una "línea apropiada" pueden ser encontrados en [51]) del objeto consulta y hallando su intersección con los hiperespacios.

3.2 Métodos para espacios métricos

Estos métodos indexan las distancias entre los elementos de la matriz de objetos. Se pueden distinguir 2 tipos fundamentales: aquellos que indexan la estructura métrica del espacio de búsqueda y los que hacen uso de las propiedades de los objetos.

3.2.1 Métodos que indexan la estructura métrica

Existen dos formas de indexar la estructura métrica. La primera es aplicable cuando se conoce la función de distancia y consiste en indexar la región de Voronoi de cada objeto de la base de datos y la segunda es factible cuando se tienen todos los pares de distancias entre los objetos de las bases de datos, es decir, se tiene una matriz de distancias, y se le asocia a cada objeto de la base de datos una lista ordenada ascendentemente por las distancias al resto de los elementos. La búsqueda de los vecinos más cercanos se hace en estas listas, pero siempre hay que hacerle variaciones a este esquema básico para reducir la complejidad del método. Organiza el espacio de una manera jerárquica, cada nodo corresponde con un hiperplano particular. Esta estructura permite una eficiente poda de las ramas basados en la desigualdad triangular. En [31] se realiza una descripción de este tipo de métodos.

3.2.2 Métodos con objetos ventajosos

Estos métodos explotan la desigualdad triangular para su funcionamiento. La matriz de objetos es decompuesta en conjuntos disjuntos, y la representan en una estructura arbórea para luego realizar la búsqueda [29, 43]. Cada nodo corresponde a una esfera, cuyo radio es la mayor de las distancias de los elementos contenidos. Las colecciones de los datos deben ser estáticas, es decir, una vez construida la estructura arbórea no se pueden agregar o quitar objetos. En [31] se realiza una descripción de este tipo de métodos.

3.3 Métodos para espacios no métricos

3.3.1 Métodos que realizan estructuraciones jerárquicas para acelerar las búsquedas

Estos métodos organizan los datos en una estructura jerárquica utilizando procedimientos de agrupamiento que no usen distancias y siguen heurísticas para realizar la búsqueda de los vecinos, porque no hacen uso de la desigualdad triangular. Sus resultados no son exactos, es decir, no son los mismos que los de la búsqueda exhaustiva [48, 53].

3.3.2 Empotrados

Es una nueva familia de métodos propuesta recientemente en [1]. Estos métodos tratan de transformar el espacio de representación de los datos, que tiene definida una medida de disimilaridad, en otro espacio (generalmente $\Re$), con una distancia. Luego, el problema consiste

en buscar los vecinos en el nuevo espacio, y finalmente transformar el resultado al espacio original. Detalles sobre esta familia pueden ser encontrados en [1].

Resultados experimentales

Se seleccionaron para la realización de la evaluación experimental los siguientes métodos:

- KAESA [44].
- KLAESA [45].
- B&B [38].
- DynDex [28].
- ClusterTree [32].

Se escogen estos métodos por considerarse representativos de sus taxonomías, además, se encuentran entre los más citados en la literatura, en el caso de los dos primeros constituyen mejoras y extensiones de otros métodos y han reportado resultados satisfactorios. En el tercer caso, el B&B original de Fukunaga y Naredra, a pesar de su edad, mantiene desempeños comparables con las propuestas más actualizadas. Se escogen además, los métodos analizados para espacios no métricos (excepto el propuesto en [48]).

A continuación se muestra la clasificación taxonómica de los métodos seleccionados para la evaluación experimental.

Tabla 1. Clasificación taxonómica de los métodos seleccionados para la evaluación experimental.

Método	***Espacio***	***Clasificación***
B&B	Vectorial	Particionado heterogéneamente
KAESA	Métrico	Indexan la estructura métrica
KLAESA	Métrico	Uso de objetos ventajosos
DynDex, ClusterTree	No métrico	Estructuraciones jerárquicas

Los resultados experimentales se llevaron a cabo con 10 bases de datos de la Universidad de California en Irvine (UCI)[34], con datos numéricos y mezclados e incompletos, pues se desea evaluar el funcionamiento de estos métodos en este tipo de datos, ya que este Reporte forma parte del proyecto de “Mejoramiento de clasificadores para datos mezclados e incompletos”. La descripción de estas bases de datos se muestra a continuación. En los experimentos, las bases de datos que no se encontraban divididas se dividieron en Entrenamiento (80%), y Prueba (20%), de forma aleatoria.

Tabla 2. Descripción de las bases de datos utilizadas en los experimentos.

Base de Datos	**# de objetos**	**# de rasgos cuantitativos**	**# de rasgos cualitativos**	**# de ausencias de valores**	**# de clases**
allbp	3772	6	23	4355	3
allhyper	3772	6	23	4355	5
allrep	3772	6	23	4355	4
ann	3772	6	15	0	3
annealing	7200	9	29	19005	6
credit	662	6	9	67	2
satimage	6435	36	0	0	6

segment	2310	19	0	0	7
shuttle	14500	9	0	0	5
sick	3772	6	23	4355	2

En los experimentos se utilizaron dos funciones de disimilaridad. La primera función de disimilaridad utilizada en los experimentos fue la HVDM. Esta función, propuesta por [54] como una distancia para datos mezclados e incompletos, tiene la característica de que a pesar de lo planteado por su autor, en presencia de datos incompletos no es definida positiva, y por tanto no es una distancia (Si un objeto *o* tiene un valor de un rasgo desconocido, HVDM(*o, o*) ≠ 0, por lo que no es definida positiva). Sin embargo, en presencia de datos completos sí es una distancia, por lo que no se afectaría el funcionamiento de los métodos propuestos para espacios vectoriales y métricos con su utilización en estas bases de datos. Además, cumple la desigualdad triangular, que es la propiedad que utilizan estos métodos para la poda.

$$HVDM(o_1, o_2) = \sqrt{\sum_{i=1}^{m} d_i^2(x, y)} \quad \textbf{(3)}$$

, donde

$d_i(x, y) = 1$ Si $x = ?$ o $y = ?$

$d_i(x, y) = \dfrac{|x - y|}{\max_i - \min_i}$ S i i es numérico

$d_i(x, y) = \sum_{c=1}^{C} \left| \dfrac{N_{i,x,c}}{N_{i,x}} - \dfrac{N_{i,y,c}}{N_{i,y}} \right|^q$ Si i es nominal

Por motivos de simplificación se considera $x = \chi_i(o_1)$ y $y = \chi_i(o_2)$.

Donde

- $N_{i,x,c}$ es el número de objetos de la matriz de objetos T que tienen valor x para el rasgo i y clase c.
- $N_{i,x}$ es el número de objetos de la matriz de objetos T que tienen valor x para el rasgo i
- C es el número de clases en el dominio del problema
- q es una constante, que usualmente toma valor 1 o 2. En nuestros experimentos, se utilizó $q = 2$.

La otra función utilizada fue una extensión introducida en este reporte de la DPF propuesta en [28], aunque utilizando diferentes criterios de comparación de los rasgos para adaptarla al trabajo con datos mezclados e incompletos.

$$eDPF(o_1, o_2) = \left(\sum_{C_a \in \Delta f} C_{\chi_i}(\chi_i(o_1), \chi_i(o_2)^r \right)^{\frac{1}{r}} \quad \textbf{(4)}$$

El criterio de comparación para los rasgos numéricos fue:

$$C_{\chi_i}(\chi_i(o_1),\chi_i(o_2)) = \begin{cases} 0 & \text{if } (\chi_i(o_1)=?) \vee (\chi_i(o_2)=?) \vee (|\chi_i(o_1)-\chi_i(o_2)| \geq \sigma) \\ 1 & \text{en otro caso} \end{cases} \quad \textbf{(5)}$$

donde "?" denota la ausencia de valores, y σ es la desviación estándar de los valores del rasgo.

Para los rasgos no numéricos se utilizó el criterio:

$$C_{\chi_i}(\chi_i(o_1),\chi_i(o_2)) = \begin{cases} 0 & \text{if } (\chi_i(o_1)=?) \vee (\chi_i(o_2)=?) \vee (\chi_i(o_1) \neq \chi_i(o_2)) \\ 1 & \text{en otro caso} \end{cases} \quad (\textbf{6})$$

Uno de los parámetros más importantes para el correcto funcionamiento de esta función es el denotado por la variable f, la cual determina el número de rasgos a evaluar para hallar la disimilaridad entre dos objetos cualesquiera. En estos experimentos se tuvo en cuenta el 60% de los rasgos, ya que esta variable no puede tener un valor fijo, pues no todas las bases de datos tienen la misma dimensionalidad. Los métodos evaluados fueron implementados en la plataforma desarrollada para ello. En todos los casos, se buscaron los 50 vecinos más cercanos (o menos disimilares de cada objeto)

Los resultados obtenidos para cada base de datos se muestran a continuación. Para evaluar el desempeño de los diferentes algoritmos se utilizaron tres medidas de calidad: la precisión, el *recall*y la reducción en el número de comparaciones.

La precisión (P) obtiene el porcentaje de los resultados retornados por el método que son correctos, entendiéndose por correcto la obtención de todos los vecinos más similares o cercanos (búsqueda exhaustiva), aunque se sobrepase el número solicitado de vecinos por existir varios vecinos con el mismo valor de similaridad. Sea *D* en conjunto de todos los vecinos correctos, y *A* el conjunto de vecinos retornados por el método:

$$P = \frac{|A \cap D|}{|A|} * 100 \quad (\textbf{7})$$

El *recall* (R) obtiene el porcentaje de los resultados retornados por el método que forman parte del resultado obtenido por la búsqueda exhaustiva.

$$R = \frac{|A \cap D|}{|D|} * 100 \quad \textbf{(8)}$$

El porcentaje de reducción (N) en el número de comparaciones (*Comp*) se realiza tomando como total a las realizadas por la búsqueda exhaustiva (cantidad de elementos en la matriz de objetos T).

$$N = \left(1 - \frac{Comp}{|T|}\right) * 100 \quad \textbf{(9)}$$

A continuación se detallan los valores de los parámetros utilizados en nuestras experimentaciones para los diferentes algoritmos.

KLAESA

Número de prototipos base…….. 40 (siguiendo a [55])
CHOICE ……Variante1 (siguiendo a [45])
CONDITION …… Variante1 (siguiendo a [45])

B&B

Cantidad de niveles……..3
Número de grupos por nodo…… 3

DynDex

Número de grupos……30
Cantidad de veces que se va a aplicar el algoritmo de agrupamiento……...3
Cantidad de grupos para aplicar el algoritmo…...20
Esquema para seleccionar los representantes…... Medoide

ClusterTree

α ……2

Para realizar los experimentos se utilizaron computadoras con las siguientes características: Intel Pentium4, con 2.6GHz de velocidad del microprocesador, 1 GB de memoria RAM, y 80 GB de capacidad en disco duro.

3.4 Resultados con la función HVDM

En cuanto a la precisión alcanzada, nótese que tanto el KAESA como el KLAESA alcanzan un 100%, pues aunque la función HVDM no es una distancia (porque con datos incompletos no es definida positiva), sí satisface la desigualdad triangular, que es la propiedad que utilizan estos métodos para encontrar los k vecinos más cercanos. El ClusterTree mostró una pobre precisión con respecto a la alcanzada por el resto de los métodos, aunque superior al 70%. El DynDex presenta de forma general resultados satisfactorios, comportándose por encima del 90% en todas las bases de datos.

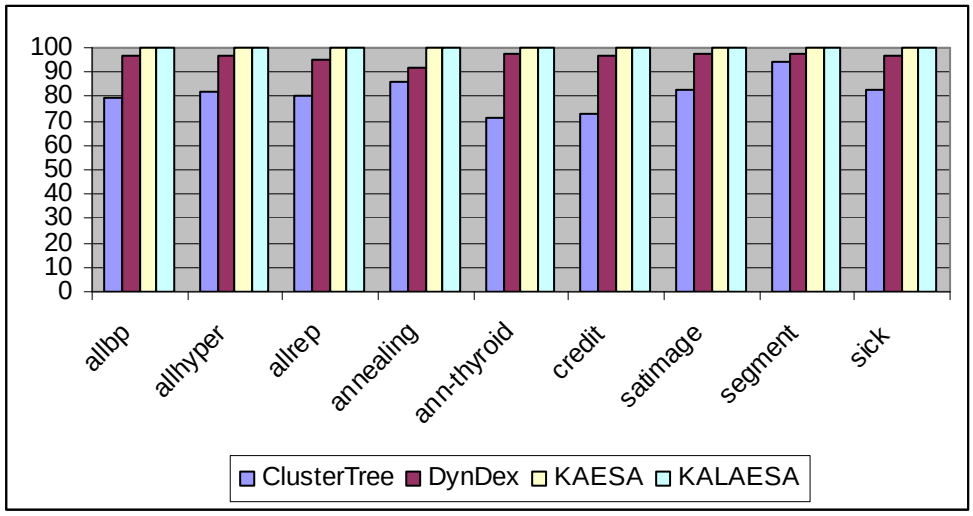

Fig. 1. Precisión (en porcentaje) alcanzada por los diferentes métodos en las bases de datos utilizadas.

En cuanto al *recall* tanto el KAESA como el KLAESA alcanzan resultados muy cercanos al 100%, pues estos métodos se restringen a un número específico de vecinos y no devuelve todos los correspondientes a la búsqueda exhaustiva. El ClusterTree mostró un *recall* menor que el del resto de los métodos, aunque superior al 70%. El DynDex presenta de forma general resultados satisfactorios, comportándose por encima del 90% en todas las bases de datos.

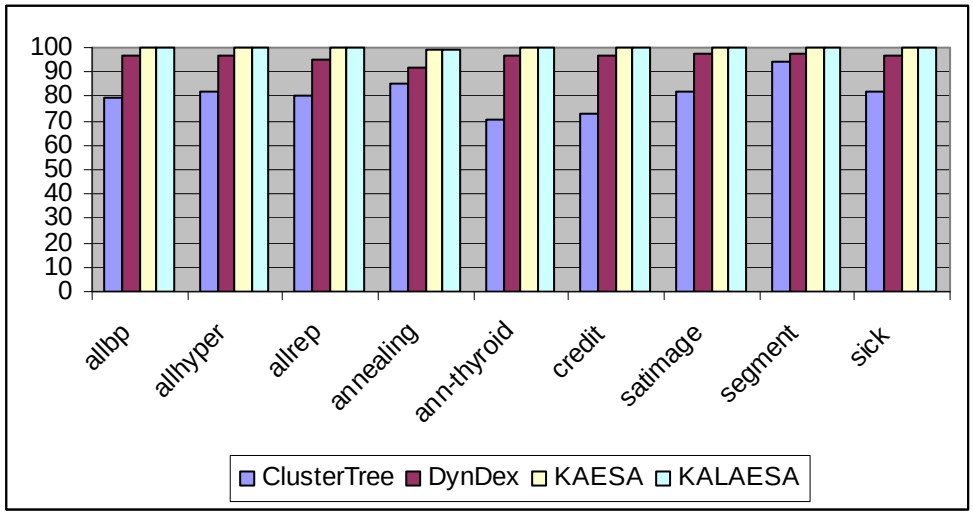

Fig. 2. *Recall* (en porcentaje) alcanzada por los diferentes métodos en las bases de datos utilizadas.

En cuanto a la reducción en el número de comparaciones, los métodos más reductores son el KAESA, KLAESA y el ClusterTree. En cuanto al DynDex, sus reducciones son pobres, pues no llegan al 20%.

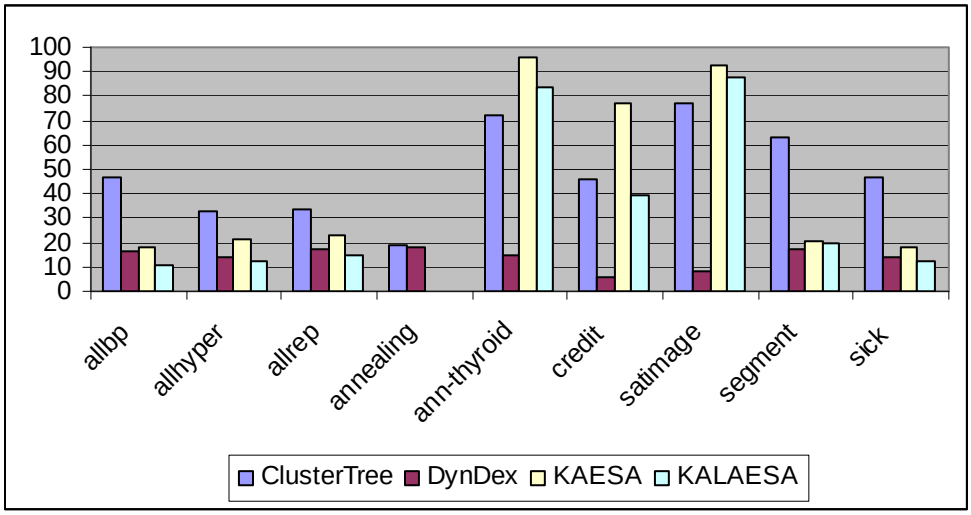

Fig. 3. Reducción en el número de comparaciones (en porcentaje) alcanzada por los diferentes métodos en las bases de datos utilizadas.

El método Branch and Bound (B&B) solo pudo ejecutarse en dos bases de datos, la "*satimage*" y la "*segment*", pues no es posible realizar promedios con datos mezclados e incompletos. Su comportamiento en estas bases de datos, para la función HVDM fue positivo, teniendo una precisión y un *recall* del 100% y con una elevada reducción en el número de comparaciones en la primera de las bases de datos, no así en la segunda, donde no sobrepasó el 10%.

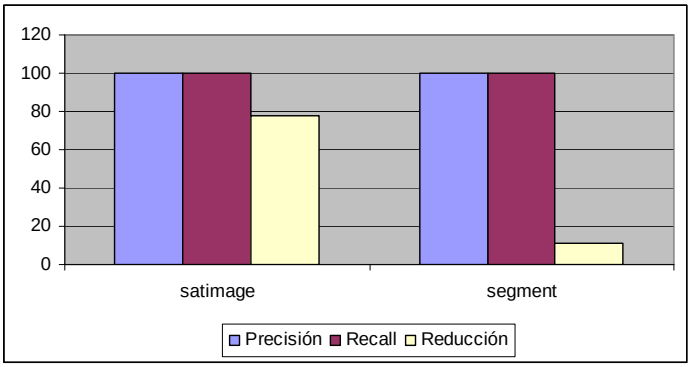

Fig. 4. Aplicación del método Branch and Bound.

3.5 Resultados con la función eDPF

En cuanto a la precisión alcanzada, nótese que tanto el KAESA como el KLAESA alcanzan una precisión muy pobre (solo superior al 30% en tres bases de datos), pues la función DPF no es

una distancia y no satisface la desigualdad triangular, que es la propiedad que utilizan estos métodos para encontrar los k vecinos más cercanos, de ahí la degradación en su desempeño. El ClusterTree mostró una precisión pobre es dos bases de datos (menor del 50%) y en el resto algo mayor, rebasando el 90% en solo una base de datos. El DynDex presenta los mejores resultados, comportándose por encima del 90% en todas las bases de datos.

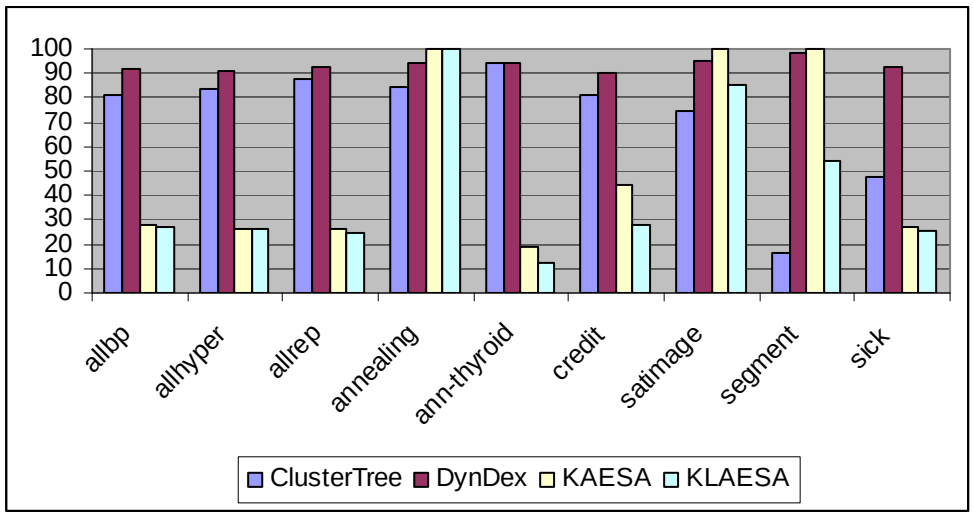

Fig. 5. Precisión (en porcentaje) alcanzada por los diferentes métodos en las bases de datos utilizadas.

En cuanto al *recall*, tanto el KAESA como el KLAESA alcanzan resultados muy bajos (solo superiores al 30% en tres de las bases de datos), pues estos métodos se restringen a un número específico de vecinos y no devuelve todos los correspondientes a la búsqueda exhaustiva. El ClusterTree mostró un *recall* por encima del 60%, excepto en una base de datos. El DynDex posee las mejores prestaciones, estando por debajo del 80% en solo una base de datos.

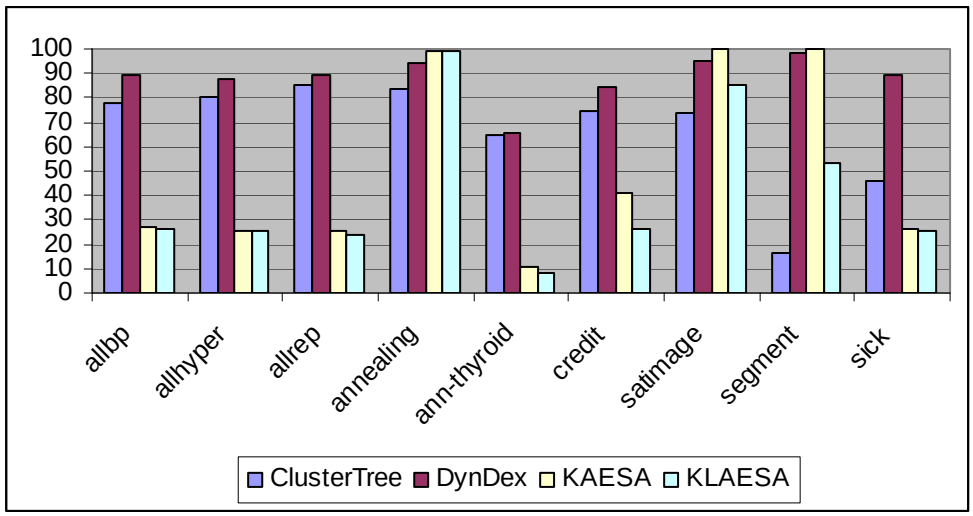

Fig. 6. *Recall* (en porcentaje) alcanzada por los diferentes métodos en las bases de datos utilizadas.

En cuanto a la reducción en el número de comparaciones, los métodos más reductores son el KAESA y el KLAESA, con reducciones superiores al 90% en todos los casos. El ClusterTree presenta reducciones altas en unas bases de datos y pobres en otras, y el DynDex muestra reducciones pobres, pues no llegan al 20%.

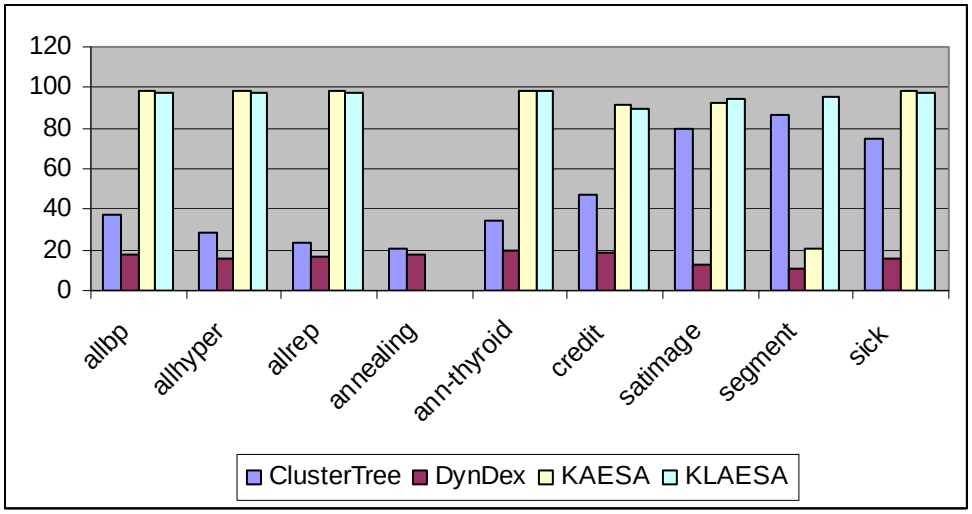

Fig. 7. Reducción en el número de comparaciones (en porcentaje) alcanzada por los diferentes métodos en las bases de datos utilizadas.

El método Branch and Bound (B&B) utilizando la función DPF tuvo una precisión y un *recall* muy cercanos al 100% y una elevada reducción en el número de comparaciones en la primera de las bases de datos, no así en la segunda, donde no sobrepasó el 10%.

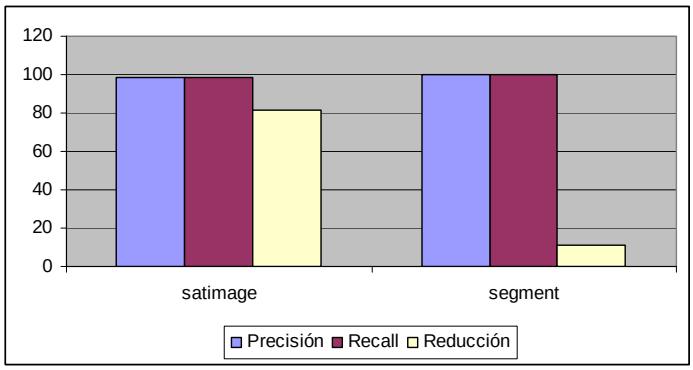

Fig. 8. Aplicación del método Branch and Bound.

De forma general, se puede concluir que cuando la función de analogía entre objetos es una distancia, y por tanto el espacio de representación de los objetos es métrico, los mejores métodos (entre los comparados) de búsqueda de los k vecinos son el KAESA y el KLAESA, el primero realiza menos comparaciones, mientras que el segundo tiene menores costos de almacenamiento. Por otra parte, en el caso de la búsqueda de los k vecinos más similares, cuando se utilizan funciones de analogía que no son distancias, y el espacio de representación de los objetos es no métrico, el DynDex es el método de mejores resultados, pues a pesar de reducir menos que el ClusterTree, tiene en la mayoría de las bases de datos buenos resultados en cuanto a la precisión y al *recall*.

4 Conclusiones

Se realizaron experimentos con 10 de las Bases de Datos del repositorio de la Universidad de California IRVIN (UCI), y se corrieron los métodos seleccionados en cada una de ellas, variando la función de analogía, pero en igualdad de condiciones para evaluar su desempeño.

Los experimentos realizados permiten afirmar que para los espacios métricos tanto el KAESA como el KLAESA tienen un buen desempeño. Hay que tener presente que los requerimientos de memoria del KAESA son superiores a los del KLAESA, aunque presenta mejores reducciones en el número de comparaciones realizadas. Para espacios no métricos el DynDex mostró el mejor desempeño, pues aunque reduce menos que el ClusterTree es mucho más preciso y retorna objetos que forman mayormente parte de la búsqueda exhaustiva.

La aplicación de los métodos diseñados para el trabajo con espacios métricos, utilizando funciones de disimilaridad que no son distancias, tiene como consecuencia una degradación importante en el desempeño de estos métodos, lo que se evidenció al utilizar la función eDPF, no así en el caso de la HVDM, que aunque no es una distancia, sí cumple la desigualdad triangular, que es la propiedad que utilizan para la poda.

Referencias

[1] V. Athitsos, "Learning embeddings for indexing, retrieval, and classification, with applications to object and shape recognition in image databases." vol. Dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy: Boston University, 2006, p. 156.

[2] L. I. Kuncheva, *Combining pattern classifiers: methods and algorithms*. Hoboken, N.J.: Wiley-Interscience, 2004.

[3] G. Toussaint, "Proximity graphs for Nearest Neighbor decision rules: recent progress," in *34 Symposium on Computing and Statistics INTERFACE-2002*, Montreal, Canada, 2002.

[4] J. Gómez-Herrera, O. Rodríguez-Morán, S. Valladares-Amaro, J. Ruiz-Shulcloper, and R. Pico-Peña, "Prognostic of Gas-oil deposits in the Cuban ophiological association, applying mathematical modeling," *Geophysics International,* vol. 33, pp. 447-467, 1995.

[5] M. R. Ortiz-Posadas, "Prognosis and evaluation of cleft palate patients' rehabilitation using pattern recognition techniques," *World Congress on Medical Physics and Biomedical Engineering,* vol. 35, p. 500, 1997.

[6] M. García-Borroto and J. Ruiz-Shulcloper, "Selecting prototypes in mixed incomplete data," *Lecture Notes in Computer Science,* vol. 3773, pp. 450-459, 2005.

[7] J. A. Olvera-López, J. A. Carrasco-Ochoa, and J. F. Martínez-Trinidad, "Sequential search for decremental edition," *Lecture Notes in Computer Science,* vol. 3578, pp. 280-285, 2005.

[8] J. A. Olvera-López, J. F. Martínez-Trinidad, and J. A. Carrasco-Ochoa, "Edition schemes based on BSE," *Lecture Notes in Computer Science,* vol. 3773, pp. 360-367, November 2005.

[9] D. Jacobs, D. Weinshall, and Y. Gdalyahu, "Class representation and image retrieval with non metric distances," in *international Conference on Computer Vision*, 1998.

[10] M. Vlachos, D. Gunopulos, and G. Kollios, "Robust similarity measures for mobile object trajectories," in *13th International Workshop on Database and Expert Systems Applications*, 2002, pp. 721- 726.

[11] L. Chen and R. T. Ng, "On the marriage of Lp norms and edit distances," in *30th International Conference on Very Large Data Bases (VLDB)*, 2004, pp. 792-803.

[12] J. S. Aguilar, J. C. Riquelme, and M. Toro, "Data set editing by ordered projection," *Intelligent Data Analysis,* vol. 5, pp. 1-13, 2001.

[13] B. V. Dasarathy, J. S. Sanchez, and S. Townsend, "Nearest Neighbour editing and condensing tools - Synergy exploitation," *Pattern Analysis & Applications,* vol. 3, pp. 19-30, 2000.

[14] S.-W. Kim and J. B. Oommen, "A brief taxonomy and ranking of creative prototype reduction schemes," *Pattern Analalisys & Applications,* vol. 6, pp. 232-244, 2003.

[15] R. Paredes and E. Vidal, "Weighting prototypes. A new editing approach," in *XV International Conference on Pattern Recognition*, 2000, pp. 25-28.

[16] A. L. Blum and P. Langley, "Selection of relevant features and examples in Machine Learning," *Artificial Intelligence,* pp. 245--271, 1997 1997.

[17] M. Dash and H. Liu, "Feature selection for classification," *Intelligent Data Analysis,* vol. I, pp. 131-156, 1997 1997.

[18] P. Domingos, "Context-sensitive feature selection for lazy learners," *Artifficial Intelligence Review,* vol. 11, pp. 227-253, 1997.

[19] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research,* vol. 3, pp. 1157 - 1182, March 2003 2003.

[20] H. Liu, H. Lu, and L. Yu, "Active Sampling: An effective approach to feature selection," in *SIAM International Conference on Data Mining*, San Francisco, 2003, pp. 244-248.

[21] M. Morita, F. Bortolozzi, R. Sabourin, and C. Y. Suen, "Unsupervised Feature Selection Using Multi-Objective Genetic Algorithms for Handwritten Word Recognition," in *ICDAR03*, 2003, pp. 666-670.

[22] Y. Santiesteban and A. Pons-Porrata, "LEX: A new algorithm to calculate typical testors," *Revista de Ciencias Matméticas,* vol. 21, 2003.

[23] B. V. Dasarathy, "Concurrent feature and prototype selection in the Nearest Neighbor decision process," in *4th World Multiconference on ystemics, Cybernetics and Informatics*, Orlando, USA, 2000, pp. 628-633.

[24] H. Ishibushi and T. Nakashima, "Evolution of reference sets in nearest neighbor classification," *LNCS,* vol. 1585, pp. 82-89, 1999.

[25] L. I. Kuncheva and L. C. Jain, "Nearest neighbor classifier: Simultaneous editing and feature selection," *Pattern Recognition Letters,* vol. 20, pp. 1149--1156, 1999.

[26] D. B. Skalak, "Prototype and feature selection by sampling and random mutation hill climbing algorithms," in *Eleventh International Conference on Machine Learning*, 1994.

[27] Y. Villuendas-Rey, M. García-Borroto, M. A. Medina-Pérez, and J. Ruiz-Shulcloper, "Simultaneuos features and objects selection for mixed and incomplete data," *Lecture Notes in Computer Science,* vol. 4225, pp. 597-605, 2006.

[28] K.-S. Goh, B. Li, and E. Chang, "DynDex: a dynamic and non-metric space indexer," in *Proceedings of the tenth ACM international conference on Multimedia* Juan-les-Pins, France: ACM Press, 2002.

[29] F. Moreno-Seco, M. L. Micó-Andrés, and J. Oncina-Carratalá, "Extending LAESA fast nearest neighbour algorithm to find the k nearest neighbours," in *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, Windsor, Ontario, Canada, 2002, pp. 718-724.

[30] F. Moreno-Seco, M. L. Micó-Andrés, and J. Oncina-Carratalá, "A modification of the LAESA algorithm for approximated k-NN classification," *Pattern Recognition Letters,* vol. 24, pp. 47-53, January 2003.

[31] J. Vermorel, "Near neighbor search in metric and nonmetric space. Tech. Rep," Laboratoire de bioinformatique de l'École des Mines 2005.

[32] B. Zhang and S. N. Srihari, "Fast k-Nearest Neighbor classification using cluster-based trees," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 26, pp. 525-528, April 2004.

[33] B. D. Dasarathy, *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* Los Alamitos, California: IEEE Computer Society Press, 1991.

[34] C. J. Merz and P. M. Murphy, "UCI Repository of Machine Learning Databases," University of California at Irvine, Department of Information and Computer Science, Irvine, Technical report 1998.

[35] E. Pękalska, R. P. W. Duin, and P. Paclík, "Prototype selection for dissimilarity-based classifiers," *Pattern Recognition,* vol. 39, pp. 189-208, 2006.

[36] K. Sang-Woon and R. P. W. Duin, "On Combining Dissimilarity-Based Classifiers to Solve the Small Sample Size Problem for Appearance-Based Face Recognition," *Lecture Notes in Computer Science,* vol. 4509, pp. 110-121, 2007.

[37] R. Paredes and E. Vidal Ruiz, "A class-dependent weighted dissimilarity measure for nearest neighbor classification problems " *Pattern Recognition,* vol. 21, pp. 1027-1036 2000.

[38] K. Fukunaga and P. M. Narendra, "A branch and bound algorithm for computing k-Nearest Neighbors," *IEEE Transaction on Computers,* vol. 24, pp. 750-753, July 1975.

[39] T. P. Yunck, "A technique to identify nearest neighbors," *IEEE Transaction Systems, Man and Cybernetics,* vol. 6, pp. 678-683, October 1976 1976.

[40] L. Miclet and M. Dabouz, "Approximative fast nearest-neighbour recognition," *Pattern Recognition Letters,* vol. 1, pp. 277-285, July 1983 1983.

[41] E. Vidal Ruiz, "An algorithm for finding nearest neighbours in (approximately) constant average time," *Pattern Recognition Letters,* vol. 4, pp. 145-157, July 1986 1986.

[42] E. Vidal Ruiz, "New formulation and improvements of the Nearest-Neighbour aproximating and elimination search algorithm (AESA)," *Pattern Recognition Letters,* vol. 15, pp. 1-7, January 1994 1994.

[43] M. L. Micó-Andrés and J. Oncina-Carratalá, "A new criterion for approximating in a recent version with linear preprocessing of the AESA algorithm," in *Selected papers from the 5th Spanish Symposium on Pattern recognition and images analysis : advances in pattern recognition and applications*, Valencia, Spain, 1994, pp. 3-11

[44] P. Aibar, A. Juan, and E. Vidal Ruiz, "Extensions to the approximating and eliminating search algorithm (AESA) for finding k-nearest-neighbours," *New Advances and Trends in Speech Recognition and Coding,* pp. 23-28, 1994.

[45] M. L. Micó-Andrés, J. Oncina-Carratalá, and 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,* vol. 15, pp. 9-17, January 1994.

[46] M. L. Micó-Andrés, J. Oncina-Carratalá, and R. C. Carrasco-Jiménez, "A fast branch & bound nearest neighbour classifier in metric spaces," *Pattern Recognition Letters,* vol. 17, pp. 731-739, June 10 1996.

[47] F. Moreno-Seco, M. L. Micó-Andrés, and J. Oncina-Carratalá, "A fast approximately k-nearest-neighbour search algorithm for classification tasks," in *Proceedings of the Joint IAPR International Workshops on Advances in Pattern Recognition*, Alicante, Spain, 2000, pp. 823-831.

[48] A. Faragó, T. Linder, and G. Lugosi, "Fast nearest neighbor search in dissimilarity spaces," *IEEE Transaction on Pattern Analysis and Machine Intelligence,* vol. 15, pp. 957-962, Semptember, 1993 1993.

[49] T. N. Raymond and J. Han, "Efficient and effective clustering methods for spatial data mining," in *20th conference on Very Large Databases (VLDB)*, Santiago de Chile, 1994, pp. 144-155.

[50] B. Zhang and S. N. Srihari, "A Fast Algorithm for Finding k-Nearest Neighbors with Non-metric Dissimilarity," in *Eighth International Workshop on Frontiers in Handwriting Recognition*, 2002, pp. 13-18.

[51] V. Castelli, "Multidimensional Indexing structure for Content-based Retrieval," Thomas J. Vawtson Research Center, New York, USA October 2001.

[52] J. L. Bently and R. A. Finkel, "Quad trees, a data structure for retreival on composite keys," *Acta Informática,* vol. 4, pp. 1-9, 1974.

[53] B. Zhang and S. N. Srihari, "Fast k-Nearest Neighbor Classification Using Cluster-Based Trees.," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 26, pp. 525-528, April, 2004 2004.

[54] R. D. Wilson and T. R. Martinez, "Improved Heterogeneous Distance Functions," *Journal of Artificial Intelligence Research,* vol. 6, pp. 1-34, 1997.

[55] L. Micó and J. Oncina, "Comparison of fast nearest neighbour classifiers for handwritten character recognition," *Pattern Recognition Letters,* vol. 19, pp. 351-356, 1998.

RT _007, Diciembre 2008
Aprobado por el Consejo Científico CENATAV

Editor: Lic. Miriela Santos Toledo
Diseño de Portada: DCG Matilde Galindo Sánchez
RNPS No. 2142
ISSN 2072-6287
Indicaciones para los Autores:
Seguir la plantilla que aparece en www.cenatav.co.cu
C E N A T A V
7ma. No. 21812 e/218 y 222, Rpto. Siboney, Playa;
Ciudad de La Habana. Cuba. C.P. 12200
Impreso en Cuba