Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Doctorado en Ciencias en Ciencias de la Computación

Índice aproximado probabilístico para la búsqueda por proximidad en memoria secundaria.

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Doctor en Ciencias

Presenta:

Fernando Luque Suárez

Ensenada, Baja California, México

2019

Tesis defendida por

Fernando Luque Suárez

y aprobada por el siguiente Comité

Dr. Edgar Leonel Chávez González

Director del Comité

Dr. Armando Castañeda Rojano

Dr. José Antonio Camarena Ibarrola

Dr. Pedro Gilberto López Mariscal

Dr. Ubaldo Ruiz López

Dr. Ubaldo Ruiz López

Coordinador del Programa de Posgrado en Ciencias de la Computación

---

Dra. Rufina Hernández Martínez

Directora de Estudios de Posgrado

Resumen de la tesis que presenta **Fernando Luque Suárez** como requisito parcial para la obtención del grado de Doctor en Ciencias en Ciencias de la Computación con orientación en Instrumentación y Control.

Índice aproximado probabilístico para la búsqueda por proximidad en memoria secundaria.

Resumen aprobado por:

Dr. Edgar Leonel Chávez González
Director de Tesis

La búsqueda en una colección por objetos similares a un objeto consulta, es un problema que se presenta en muchas aplicaciones en diversas áreas de las ciencias de la computación, como reconocimiento de patrones, recuperación multimedia, identificación con biométricos, etc. En este trabajo nos enfocamos en el modelo más general para la solución del problema que se basa en la noción de espacio métrico, donde la única información disponible es la distancia entre cualquier pareja de objetos en la colección. Con el de objetivo de ser eficiente en una búsqueda, se ha propuesta la organización de los objetos en la colección en una estructura llamada índice, para solo revisar una parte de la colección en una consulta. Debido al efecto de la maldición de la dimensión, los índices más rápidos sacrifican algunas respuestas correctas para poder ser eficientes aún cuando la representación de los objetos en la colección se encuentra en altas dimensiones, estos son los llamados índices aproximados para la búsqueda de similitud. Los índices aproximados más eficientes en la literatura se basan en la navegación sobre un grafo que aproxima una triangulación Delaunay, pero son eficientes únicamente en memoria principal. Para memoria secundaria la técnica que ha reportado mejores resultados es la de archivo invertido métrico, pero para tener una alta precisión requiere de revisar un alto porcentaje de la colección, además de tener posfiltrado con una función costosa computacionalmente de calcular. En este trabajo proponemos el uso de un cubrimiento del conjunto para la construcción de un índice métrico. En la fase de indexación la colección es descompuesta en subconjuntos de un tamaño acotado por una constante, tal que la unión de los subconjuntos es la colección. En una búsqueda, solo los subconjuntos relevantes a la consulta son activados y la unión de estos subconjuntos conforman los candidatos. Después, con un proceso simple de posfiltrado logramos filtrar los candidatos a un número considerablemente más pequeño. Mostramos experimentalmente con una colección de un millón de vectores de 128 dimensiones, que la estructura propuesta alcanza una precisión del $99\,\%$ revisando únicamente el $1\,\%$ de la colección. Para el problema de identificación del parlante, se requiere conocer su identidad persona utilizando únicamente su voz, se diseñó una solución que permite la indexación, por lo tanto, puede aplicarse en bases de datos grandes. Evaluamos experimentalmente la calidad de la identificación con una base de datos pública formada mediante la extracción de audio de una colección de videos de YouTube de 1,000 oradores distintos. Con segmentos de audio de 20 segundos, pudimos identificar a un parlante con un $95\,\%$ de precisión, cuando el entorno de grabación no está controlado, y con un $99\,\%$ de precisión para entornos de grabación controlados.

Palabras Clave: **Búsqueda en espacios métricos, indexación, indentificación del parlante**

Abstract of the thesis presented by **Fernando Luque Suárez** as a partial requirement to obtain the Master of Science degree in Doctor in Computer Science in Computer Science.

Aproximate probabilistic index for the search of similarity in secondary memory.

Abstract approved by:

Dr. Edgar Leonel Chávez González
Thesis Director

Searching in a collection for objects similar to a query object is a problem that occurs in many applications in several areas of computer science, such as pattern recognition, multimedia recovery, biometric identification, etc. In this work we focus on the more general model for solving the problem that is based on the notion of metric space, where the only information available is the distance between any pair of objects in the collection. With the aim of being efficient in a search, the organization of the objects in the collection has been proposed in a structure called index, to only review a part of the collection in a query. Due to the effect of the curse of the dimension the faster indices, sacrifice some correct answers to be able to remain efficient even when the representation of the objects in the collection is in high dimensions, these are the so-called approximate indexes. The most efficient approximate indexes in the literature are based on navigation on a graph that approximates a Delaunay triangulation,but are efficient only in main memory. For secondary memory, the technique that has reported the best results is the metric inverted file, but to have a high precision it requires reviewing a high percentage of the collection, in addition to having post-filtering with a computationally expensive function to calculate. In this work we propose the use of a covering of the set for the construction of a metric index. In the indexing phase, the collection is broken down into subsets of a size bounded by a constant, such that the union of the subsets is the collection. In a search, only the subsets relevant to the query are activated and the union of these subsets make up the candidates. Then, with a simple post-filtering process, we managed to filter the candidates to a considerably smaller number. We show experimentally with a collection of one million vectors of 128 dimensions, that the proposed structure reaches an accuracy of $99\,\%$ by reviewing only $1\,\%$ of the collection. For the problem of identification of the speaker where a person's voice is used, it is required to know their identity, a solution was designed that allows indexing, therefore, it can be applied in large databases. We experimentally evaluate the quality of identification with a public database formed by extracting audio from a collection of YouTube videos from 1,000 different speakers. With 20 second audio segments, we were able to identify a speaker with $95\,\%$ accuracy, when the recording environment is not controlled, and with $99\,\%$ accuracy for controlled recording environments.

Keywords: **Searching in metric spaces, indexing, speaker identification**

Dedicatoria

A Alejandra, mis padres y familia

Agradecimientos

Agradezco a mi novia Alejandra Silva por su apoyo y comprensión durante el doctorado, gracias por todo.

Agradezco a mí director de tesis Dr. Edgar Leonel Chávez González por la asesoría en estos años. También, a los miembros del comité de tesis, por sus comentarios y sugerencias durante el desarrollo de este trabajo.

A mis compañeros en el departamento, por todo el apoyo y convivencia, que hicieron el tiempo en el posgrado muy agradable.

Al Centro de Investigación Científica y de Educación Superior de Ensenada por permitirme estudiar este posgrado.

Finalmente, al Consejo Nacional de Ciencia y Tecnología (CONACyT) por brindarme el apoyo económico para realizar mis estudios de doctorado.

Tabla de contenido

Página

Tabla de contenido (continuación)

Lista de figuras

Lista de figuras (continuación)

Lista de tablas

Capítulo 1. Problema

1..1. Introducción

La búsqueda de objetos similares a un objeto de consulta en una colección, es una tarea que se presenta de forma ubicua en muchas áreas de la computación como la recuperación de información, visión por computadora, aprendizaje de máquina, por citar algunas. En esta tarea se comparte el mismo principio de búsqueda en colecciones de objetos muy grandes donde el único comparador disponible es la similitud entre cualquier par de objetos. Un ejemplo es el caso de imágenes (ver Figura:1) donde dada una imagen consulta se requiere obtener imágenes similares de una base de datos.

Compañías como Pinterest Pinterest (2009) con más de 291 millones de usuarios y valuada en 750 millones de dólares, basan su éxito en la exploración de contenidos en la web, buscando contenido con imágenes similares a una imagen de consulta. Spotify Spotify (2006) con 217 millones de usuarios y valuada en 5.98 billones de dólares, también utiliza búsqueda por similitud en su sistema de recomendaciones de canciones, estas recomendaciones se realizan buscando canciones con el contenido de música previamente escuchada por el usuario.

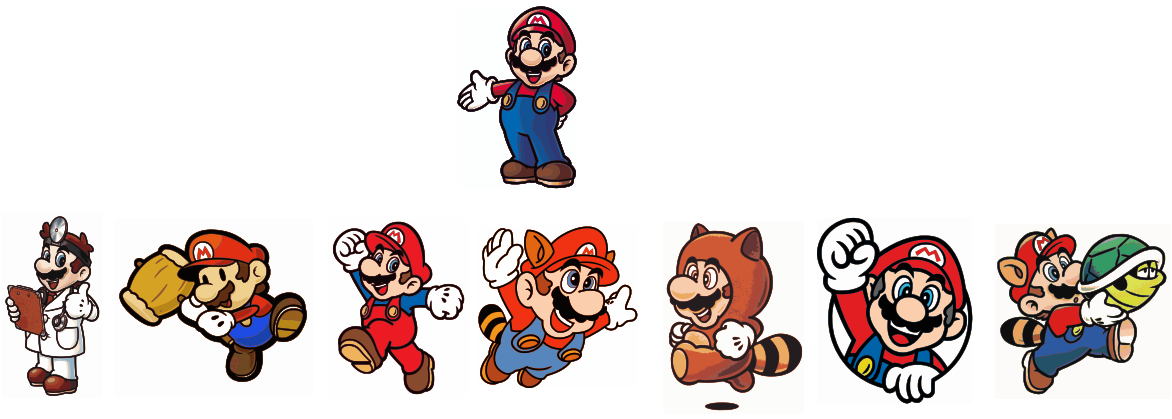


Figura 1. Con una imagen consulta (arriba) la tarea es seleccionar imágenes parecidas o cercanas a la consulta contenidas en una colección de millones de imágenes.

Los motores de búsqueda tradicionales como los basados en transacciones sql, e. g. mssql, mysql, oracle, etc., tienen como objetivo recuperar objetos exactamente iguales a un objeto de consulta, este concepto es llamado “búsqueda exacta”. En este caso, una base de datos es dividida en registros, donde cada uno cuenta con al menos una llave comparable. Cuando se realiza una consulta a la base de datos lo que se recupera son todos los registros cuyas llaves coincidan con la llave de consulta. Normalmente los registros son guardados utilizando tipos de datos

numéricos o alfanuméricos aunque se tiene la posibilidad de guardar tipos complejos como imagen o arreglos binarios, es importante mencionar que aún con estas representaciones el tipo de búsqueda es exacto.

Para buscar información en repositorios donde los objetos no tienen ningún orden lineal, e. g. colecciones de imágenes, sonido o cualquier objeto digital complejo, se podría estructurar los objetos (manualmente o automáticamente) en llaves y registros. Sin embargo, es muy complicado y además limita el tipo de búsquedas que se pueden realizar sobre la colección, lo que en general trunca las áreas de aplicación de una solución de este tipo.

Un concepto más general es el de "búsqueda por similitud", esto es, buscar en una base de datos los objetos que sean similiares o cercanos a un objeto de consulta. El modelo más general para estas búsquedas se basa en la noción matemática de espacio métrico donde un espacio X es normado por una función distancia d.

Formalmente una "búsqueda por similitud" en un espacio métrico X (usualmente $\mathbb{R}^n$) se define de la siguiente manera: dado un conjunto $S = \{u_1, ..., u_l\} \subset X$, una función distancia $d : X \times X \to \mathbb{R}$, un objeto de consulta $q \in X$ y un criterio de búsqueda, el objetivo es recuperar un subconjunto en S con los objetos similares o cercanos a q. Es necesario que para todo $x, y, z \in X$ se cumplan las propiedades de métrica:

i) $d(x, y) \geq 0 \;\; positiva$

ii) $d(x, y) = 0$, implica $x = y \;\; reflexiva$

iii) $d(x, y) = d(y, x) \;\; simétrica$

iv) $d(x, y) + d(y, z) \geqslant d(x, z) \;\; desigualdad \;\; del \;\; triángulo$

Probablemente las funciones de distancia más utilizadas en la literatura son los miembros de la familia de funciones de $Minkowski$ o también conocidas como distancias L^p, que se definen como: dados los objetos $x = (x_1, x_2, \ldots, x_n)$ y $y = (y_1, y_2, \ldots, y_n) \in \mathbb{R}^n$

$$d(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p} \tag{1}$$

para $p \geq 1$. Para $p = 1$ la distancia de $Manhattan$, para $p = 2$ la distancia $Euclidiana$. El número de operaciones requerido para evaluar estas funciones depende de la dimensión en la que se encuentran los objetos $O(n)$ para $\mathbb{R}^n$. En general, en espacios de dimensión baja se puede calcular la distancia rápidamente. Sin embargo, existen funciones en donde aún en dimensión baja se requiere un número elevado de operaciones para su cálculo, e. g. la distancia de $Haussdorf$ d_{H} donde dados los subconjuntos Y, Z del espacio métrico (X, d) se define como:

$$d_{\mathrm{H}}(Y, Z) = \max\{ \sup_{y \in Y} \inf_{z \in Z} d(y, z), \sup_{z \in Z} \inf_{y \in y} d(z, y) \}, \tag{2}$$

La distancia de Haussdorf es la mayor de las distancias desde un conjunto de puntos al punto más cercano en otro conjunto. Dos conjuntos son cercanos si cada punto de cualquier conjunto es cercano a algún punto en el otro conjunto. Debido a que $\sup \inf d(y, z)$ no es reflexivo (ver Figura 2), para calcular d_{H} se requieren $O(nN^2M^2)$ donde N, M es la cardinalidad de Y, Z respectivamente, esto significa que incluso en bajas dimensiones esta distancia es muy costosa computacionalmente.

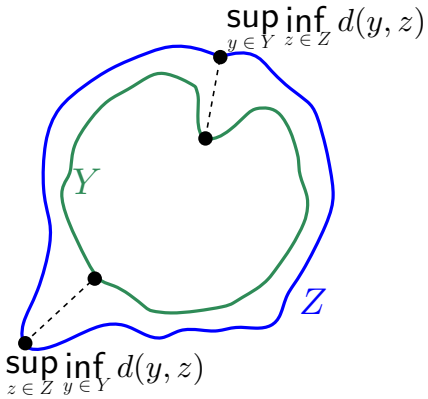


Figura 2. Ilustración de la distancia de Haussdorf

Los criterios de búsqueda más explorados en la literatura son: la búsqueda de los k vecinos más cercanos donde dado un número positivo k, la tarea es obtener un subconjunto en S de tamaño k con los objetos más cercanos a q, y la búsqueda por rango donde para un número real r dado, el objetivo es obtener el subconjunto de objetos en S tal que la distancia a q es menor igual a r (ver Figura 3).

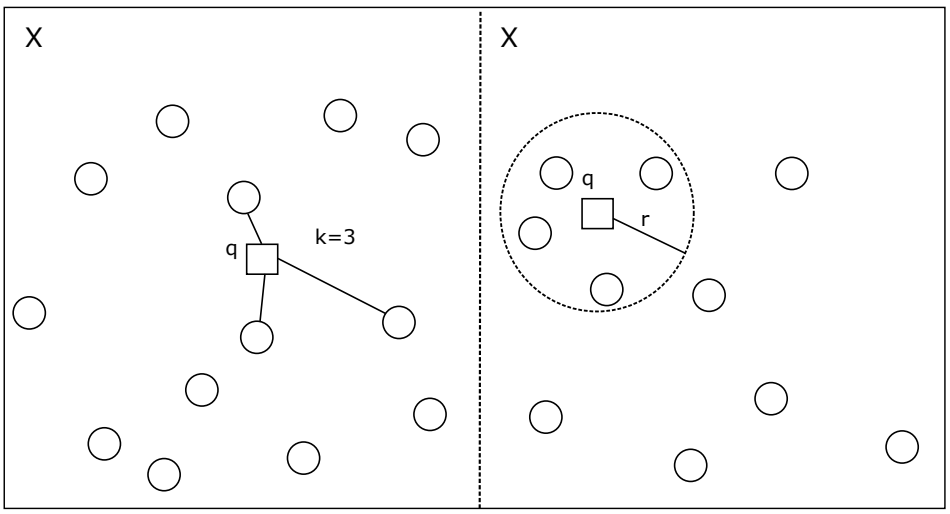


Figura 3. En la imagen izquierda se muestra una búsqueda de los k vecinos más cercanos con $k = 3$. En la derecha un ejemplo de búsqueda por rango con un radio r.

1..2. Búsqueda secuencial

Cualquier tipo de búsqueda puede responderse correctamente evaluando la distancia entre la consulta q y todos los objetos en S para luego ordenar los objetos con respecto a la distancia obtenida en orden ascendente y reportar los primeros objetos en el orden. A este barrido sobre S se le conoce como búsqueda secuencial y es una solución viable únicamente cuando el tamaño de S es pequeño. Esto se debe a que el tiempo en responder una consulta depende mayormente del número de distancias que se tienen que evaluar en una búsqueda, que para este método es igual al número de elementos en S, es decir tiene un tiempo de consulta $O(nN)$ donde N es el tamaño de S y n es la dimensión en la que se encuentra S.

En una colección de un millón objetos en dimensión 128, cuando se realiza una búsqueda con la distancia $Euclidiana$, se deben evaluar alrededor de $256,000,000$ operaciones para obtener una respuesta. Experimentalmente en Aumüller *et al.* (2017) se observa que con este método se completan 6.5 consultas en un segundo, lo que en algunos escenarios de aplicación no es un tiempo aceptable, además que este tiempo incrementará de forma lineal con respecto al tamaño N de la colección.

Es por esto, que en la literatura se han propuesto estructuras llamadas índices donde en un proceso previo a una búsqueda, se organizan los elementos en S con el objetivo de evadir el barrido sobre la colección completa al responder en una consulta.

1..3. Índices exactos para la búsqueda de similitud.

Las primeras estructuras propuestas fueron los llamados índices exactos para búsqueda de similitud, estos tienen como característica principal, responder correctamente a una consulta comparándola únicamente con una parte de la colección. Para medir el rendimiento de un índice dada una entrada de N puntos en $\mathbb{R}^n$ se analiza el tiempo de construcción del mismo, el peor tiempo de consulta posible y el espacio requerido, todo caracterizado como cota asintótica superior en función de N.

El caso más simple se presenta cuando los objetos están en una dimensión, es decir, en un conjunto con un orden total como los números enteros, reales o cadenas. Si se requiere encontrar el número más cercano a un número consulta dentro del listado, se podría utilizar el método de búsqueda secuencial pero el tiempo de consulta dependerá linealmente del tamaño de la lista $O(N)$.

Otra manera es, previo a una búsqueda ordenar el listado S de N números tal que $u_1 \leq ... \leq u_l$. El siguiente paso es encontrar el vecino más cercano a q en S, como se describe en el Algoritmo 1

Algoritmo 1 Búsqueda binaria

Entrada: S conjunto de números, q número consulta
Salida: El número en S más cercano a q

```
Función BUSQUEDABINARIA(S,q)
    L = 1, R = |S|, min_d = ∞ y a min_i = −1.
    Mientras L ≤ R Hacer
        m = ⌊(L + R)/2⌋.
        Si min_d > |S[m] − q| Entonces
            min_d = |S[m] − q|
            min_i = m
        Fin Si
        Si S[m] < q Entonces
            L = m + 1
        Fin Si
        Si S[m] > q Entonces
            R = m − 1
        Fin Si
    Fin Mientras
     Regresa S[min_i]
Fin Función
```

La idea en este algoritmo es descartar un subarreglo con la mitad de los elementos en $S[L...R]$

que inicialmente es el arreglo completo, para luego iterar con el subarreglo que no se descartó, siempre comparando la consulta q con el número en la posición $\lfloor (L+R)/2 \rfloor$ hasta llegar a la frontera de la lista. A este método se le conoce como búsqueda binaria y es efectivo para dimensión $n = 1$.

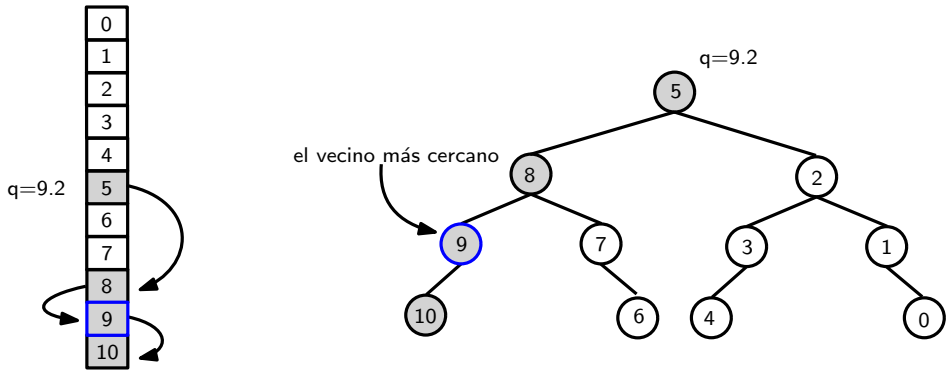


Figura 4. Ilustración de la búsqueda binaria.

Para ordenar el listado se requiere un tiempo $O(Nlog(N))$ usando algún algoritmo basado en comparaciones, e. g. *mergesort*. En el peor de los casos se tiene que realizar $O(log(N))$ comparaciones en una consulta y como la estructura es el mismo listado inicial ordenado el espacio requerido es $O(N)$.

Para el caso $n > 1$ existen una gran variedad de técnicas en la literatura Chávez *et al.* (2001), por ejemplo, el *kdtree* presentado en Friedman *et al.* (1976). La idea detrás de este algoritmo es la construcción de un árbol binario con cada nodo representando un hiperplano que separa en dos partes el espacio $\mathbb{R}^n$, los puntos que quedan a la izquierda del hiperplano son representados por el subárbol izquierdo de ese nodo, y los puntos a la derecha del hiperplano son representados por el subárbol de la derecha. Cada nodo es asociado a una de las n dimensiones y genera un hiperplano que pasa por el nodo y es ortogonal al eje de la dimensión asociada.

Dado que hay muchas formas de elegir hiperplanos de división alineados con ejes, hay muchas maneras diferentes de construir *kdtrees*. El método canónico de construcción de árboles *kdtrees* tiene las siguientes restricciones:

- A medida que uno se mueve hacia abajo en el árbol, uno recorre los ejes utilizados para seleccionar los hiperplanos de división. Por ejemplo, en un árbol tridimensional, la raíz

tendría un hiperplano alineado en x, los hijos de la raíz tendrían hiperplanos alineados en y, los nietos de la raíz tendrían hiperplanos alineados en z, los bisnietos de la raíz tendrían hiperplanos alineados en x, los tataranietos de la raíz tendrían hiperplanos alineados en y, y así sucesivamente.

- Los nodos se insertan seleccionando la mediana de los puntos en un subárbol con respecto a sus coordenadas en el eje que se utiliza para crear el plano de división.

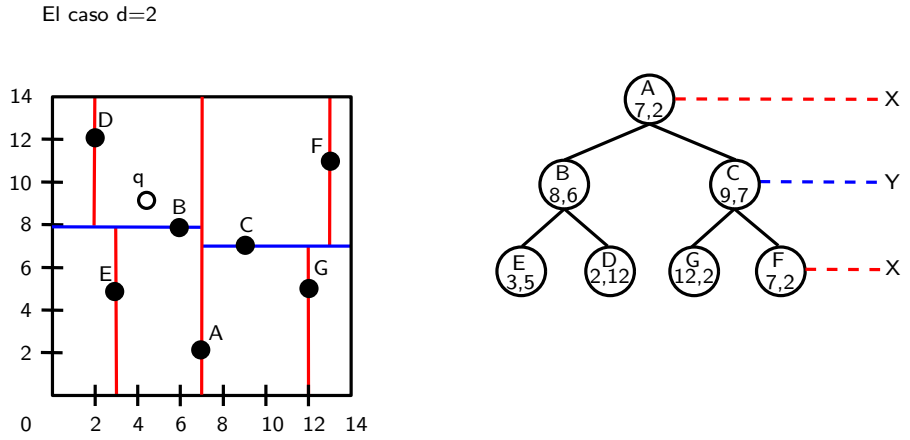


Figura 5. Construcción canónica del *kdtree*.

Algoritmo 2 Construcción kdtree

Entrada: S conjunto de números, $prof$ profundidad
Salida: El nodo raíz del kdtree
Función CONSTRUCCIONKDTREE(S,$prof$)
$coord = prof \mod (n)$.
$mediana =$ Seleccionar la mediana con respecto a $coord$ en S
$nodo.coordenadas = mediana$
$nodo.hijoizquierdo =$ kdtree(puntos en S antes de la $mediana$, $prof + 1$)
$nodo.hijoderecho =$ kdtree(puntos en S despúes de la $mediana$, $prof + 1$)
Regresa $nodo$
Fin Función

La búsqueda del vecino más cercano puede realizarse eficientemente tomando ventaja de tres propiedades del árbol:

1. Empezando con el nodo raíz, el algoritmo se mueve hacia abajo en el árbol de forma recursiva de la siguiente manera: va hacia la izquierda o hacia la derecha dependiendo de si el punto consulta es menor ó mayor que el nodo actual en la dimensión asociada al nivel.

2. Una vez que el algoritmo llega a un nodo, verifica la distancia entre la consulta y el nodo y si la distancia es menor que el “mejor actual”, el punto del nodo se guarda como el “mejor actual”.

3. El algoritmo deshace la recursión del árbol, realizando los siguientes pasos en cada nodo:

    a) Si el nodo actual está más cerca que el “mejor actual”, se convierte en el “mejor actual”.

    b) El algoritmo verifica si existe algún punto al otro lado del hiperplano de división que esté más cerca del punto de búsqueda que el “mejor actual”. Esto se hace validando si el hiperplano de división intersecta con una hiperesfera alrededor del punto de búsqueda que tiene un radio igual a la distancia al “mejor actual”.

        - Si la hiperesfera cruza el plano de división, podría haber puntos más cercanos en el otro lado del plano, por lo que el algoritmo debe moverse hacia la otra rama del árbol.
        - Si la hiperesfera no cruza con el plano de división, entonces el algoritmo continúa caminando por el árbol y se elimina toda la rama en el otro lado de ese nodo.

4. Cuando el algoritmo finaliza el proceso a partir del nodo raíz, se completa la búsqueda y el “mejor actual”será la respuesta.

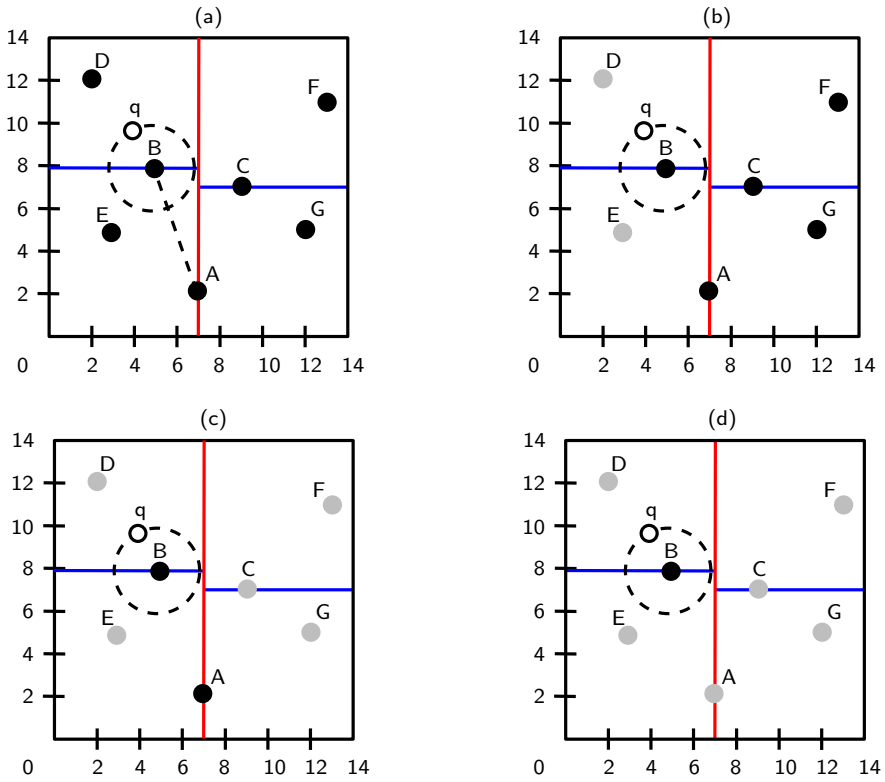


Figura 6. a) Se busca por profundidad la hoja B como candidato "mejor actual" a una distancia $d(q, B)$. b) Se descartan los objetos vinculados D y E por estar más lejanos que el "mejor actual". c) Se deshace la recursión y se revisa la intersección entre el hiperplano y la hiperesfera, se descarta la rama derecha de A. d) La búsqueda termina en la raíz con B como "mejor actual"

El árbol tiene $O(log(N))$ niveles y en cada nivel para agrupar los puntos de un lado de la mediana se requieren $O(N)$ operaciones, entonces el tiempo de construcción está acotado por $O(Nlog(N))$. El primer paso en una búsqueda para llegar al último nivel del árbol toma un tiempo $O(log(N))$, pero cuando se deshace la recursión del árbol puede darse el caso de tener que revisar todas las ramas del árbol. Si la dimensión es $n = 1$ entonces el número de ramas en el árbol sería por lo menos 2, para $n = 2$ por lo menos 4, para $n = 3$ por lo menos 8, es decir el número de ramas crece 2^n y en el peor caso hay que revisarlas todas, entonces el tiempo de consulta es $O(2^n + log(N))$. En la evaluación Aumüller *et al.* (2017) se muestra experimentalmente que en una colección de un millón de puntos en 128 dimensiones el *kdtree* tiene menor eficiencia que la búsqueda secuencial. Todos los índices exactos conocidos sufren de este comportamiento que es una de las consecuencias de un fenómeno conocido como la "maldición de la dimensión".

Algoritmo 3 Búsqueda del vecino más cercano en el kdtree

Entrada: q punto consulta, T raíz del kdtree, $prof$ profundidad, hp hiperplano
Salida: El vecino más cercano de q en el kdtree

```
Función NNKDTREE(S,T,prof,hp)
    Si T == ∅ ó distancia(q, hp) > mejoractual Entonces Regresa
    Fin Si
    prof = prof  mód (n).
    dist = distancia(q, T.coordenadas)
    Si dist < mejoractual Entonces
        mejor = T.coordenadas
        mejoractual = dist
    Fin Si
    Si q[prof] < T.coordenadas[prof] Entonces
        NNKDTREE(q,T.hijoizquierdo,prof + 1,obtenerhp(prof, T.coordenadas))
        NNKDTREE(q,T.hijoderecho,prof + 1,obtenerhp(prof, T.coordenadas))
    De otro modo
        NNKDTREE(q,T.hijoderecho,prof + 1,obtenerhp(prof, T.coordenadas))
        NNKDTREE(q,T.hijoizquierdo,prof + 1,obtenerhp(prof, T.coordenadas))
    Fin Si
     Regresa mejor
Fin Función
```

En Chávez *et al.* (2001) se realizó una clasificación de algoritmos de indexación exacta, donde principalmente se dividen las técnicas que particionan el espacio en algo similar a un diagrama de Voronoi (triangulación Delaunay) con el objetivo de que en tiempo de consulta solo se revise la región en la que cae la consulta y los vecinos Voronoi de esa región, e. g. Bisector tree (BST) (Kalantari y McDonald, 1983), generalized near neighbor tree (GNAT) (Brin, 1995), spatial aproximation tree (SAT) (Navarro, 1999), list of clusters (LC) (Chávez y Navarro, 2000), etc. Otro tipo de técnicas utiliza un conjunto de k puntos llamados pivotes para mapear el espacio métrico en $\mathbb{R}^n$ a $\mathbb{R}^k$ utilizando la distancia L_∞, e. g. Burrkard and Keller tree (BKT) (Baeza-Yates *et al.*, 1994), fixed height query tree (FHQT) (Baeza-Yates, 1997), linear approximating eliminating search algorithm (LAESA) (Micó *et al.*, 1994), vantage point tree (VPT) (Uhlmann, 1991b), etc. Por lo general, estos algoritmos utilizan árboles como estructuras de datos y en algunos casos matrices de distancias como en LAESA. El análisis del espacio, tiempo de construcción y tiempo de consulta con respecto al tamaño N del conjunto se muestra en la Tabla 1.

Tabla 1. La complejidad solo toma en cuenta N ningún otro parámetro como la dimensión. α es un número entre 0 y 1. k es el número de pivotes, h es la altura y m el número de centroides. * si $h = log(N)$. ** resultado experimental sin análisis. *** solo se satisface si la búsqueda cumple ciertos parámetros.

Tipo	Estructura	Espacio	Construcción	Consulta
Pivotes	BKT	$O(N)$	$O(Nlog(N))$	$O(N^{\alpha})*$
	FHQT	$O(Nh)$	$O(Nh)$	$O(log(N))*$
	LAESA	$O(kN)$	$O(kN)$	$k + O(1)$**
	VPT	$O(N)$	$O(Nlog(N))$	$O(log(N))$***
Partición del espacio	BST	$O(N)$	$O(Nlog(N))$	sin analizar
	GNAT	$O(Nm^2)$	$O(Nmlog_m(N))$	sin analizar
	LC	$O(N)$	$O(N^2/m)$	$O(N)$
	SAT	$O(N)$	$O(Nlog(N)log(log(N)))$	$O(N^{1-O(1/log(log(N)))})$

1..4. Maldición de la dimensión.

Desafortunadamente en espacios métricos de alta dimensión cualquier pareja de puntos tienden a estar alejados entre sí y además las distancias en el espacio tienden a ser iguales. Este comportamiento es consecuencia de la pérdida de relación entre área y volumen al incrementar la dimensión de un espacio métrico, conocida como la maldición de la dimensión. Para ilustrar lo anterior supongamos un hipercubo unitario U (ver Figura 7) donde insertaremos de manera uniforme un conjunto de N puntos S en el espacio, deseamos observar cual es el área requerida para cubrir los k vecinos más cercanos de un punto q.

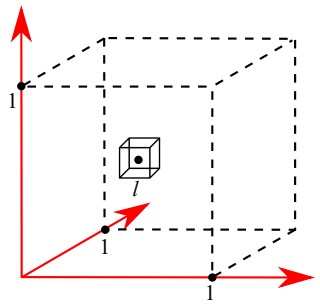


Figura 7. Ilustración de hipercubo unitario.

Tenemos un hipercubo unitario $[0,1]^n$, además todos los puntos en el espacio se encuentran dentro de este hipercubo distribuidos de manera uniforme, i. e. $\forall i, x_i \in [0,1]^n$, y vamos a considerar los k vecinos más cercanos al punto q. Ahora supogamos que l es el tamaño del lado del hipercubo V que contiene los k vecinos más cercanos de q, entonces el volumen de V está dado por $l^n \approx \frac{k}{N}$ (debido a la distribución uniforme de los N puntos en U) lo que implica que $l \approx \left(\frac{k}{N}\right)^{1/n}$. Si

fijamos $N = 1000$ y $k = 10$ se puede ver en la Tabla 2 el rápido crecimiento de l con respecto a la dimensión n.

Tabla 2. Dimensión n, tamaño de lado l, tamaño del conjunto $N = 1000$ y los $k = 10$ vecinos más cercanos.

n	l
2	.1
10	.63
100	.955
1000	.9954

Cuando n se incrementa casi toda el área del espacio es cubierta para encontrar los k vecinos más cercanos de q, esto tiene como consecuencia una concentración de los puntos fuera de los k vecinos más cercanos en el espacio restante entre el hipercubo V y el hipercubo U (ver Figura 8), además es muy probable que los k vecinos se encuentren en la frontera del hipercubo V. Esto contradice la suposición de que los k vecinos más cercanos a un punto q se encuentran particularmente más cerca de q que los otros puntos en el espacio.

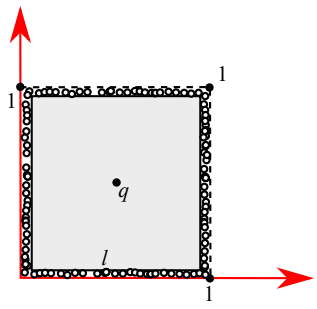


Figura 8. Ilustración de la concentración en altas dimensiones.

Se podría pensar que incrementando el número de puntos en S hasta que los k vecinos más cercanos estén realmente cerca de q se resolvería el problema. ¿Pero cuántos puntos se requieren para que l sea realmente pequeño?. Si fijamos $l = \frac{1}{10} = 0.1 \Rightarrow N = \frac{k}{l^n} = k \times 10^n$ lo que claramente crece exponencialmente. Para tener una idea sobre este crecimiento para $n = 100$ se requieren $k \times 10^{100}$ puntos, que es un número más grande que los 10^{79} electrones en el universo.

El fenómeno de la maldición de la dimensión en la práctica se puede ver en la Figura 9. Los histogramas muestran la distribución de las distancias entre todas las parejas de puntos, de 1000 puntos generados aleatoriamente en n dimensiones. Se puede observar que cuando se incrementa n, los puntos se alejan y las distancias quedan dentro de un rango muy pequeño.

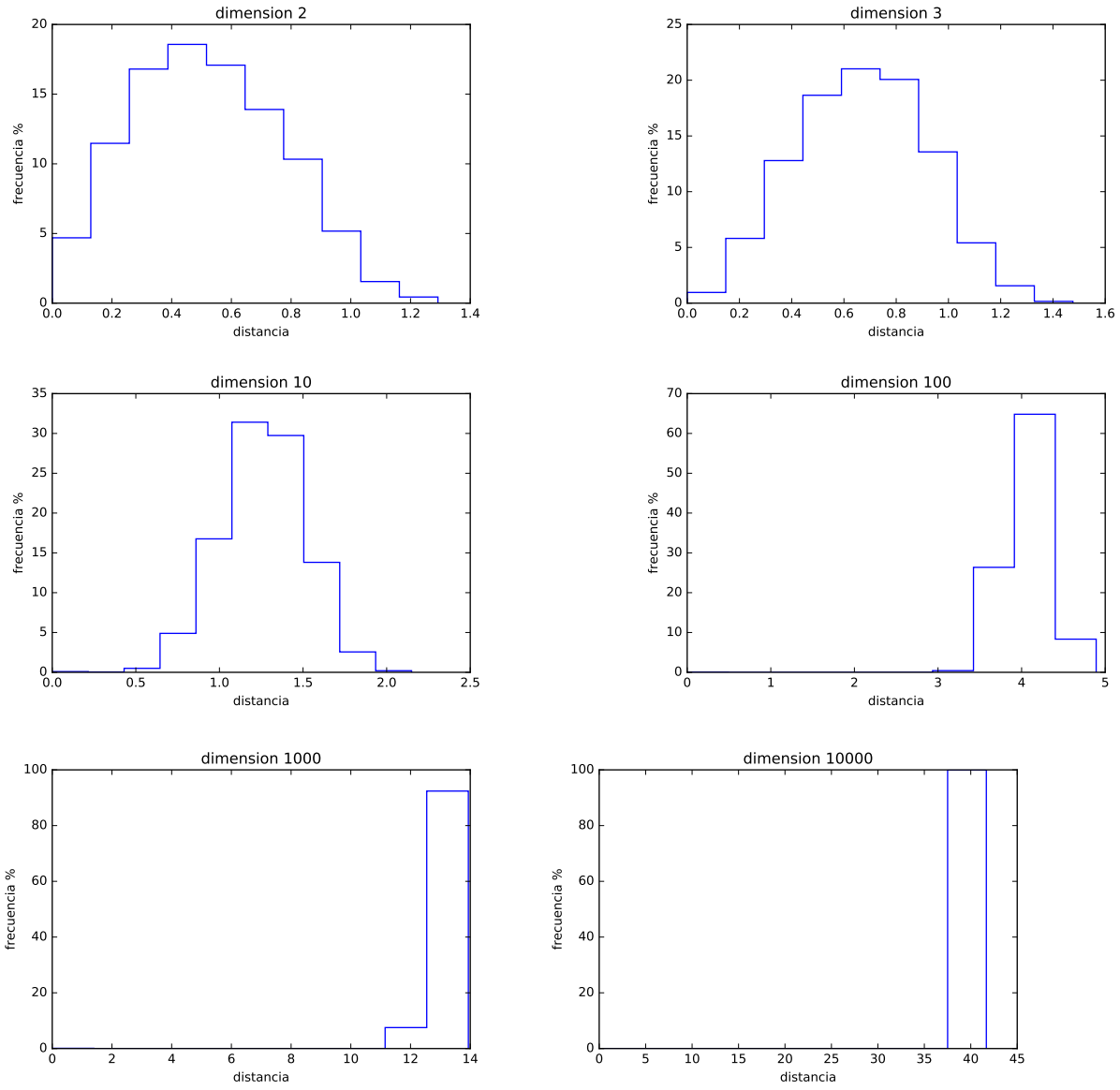


Figura 9. Histogramas de distancias en n dimensiones.

En algúnas ocasiones se tienen espacios de búsqueda que aún cuando existan en una dimensión baja se encuentran en un ambiente de alta dimensión (ver Figura 10), la “maldición de la dimensión” se presenta cuando la dimensión intrínseca (el mínimo número de dimensiones con las que se puede representar el espacio) es alto. En Chávez *et al.* (2001) se propuso medir la dimensión intrínseca $\rho = \frac{\mu^2}{2\gamma^2}$ donde μ y γ^2 son la media y la varianza del histograma de distancias.

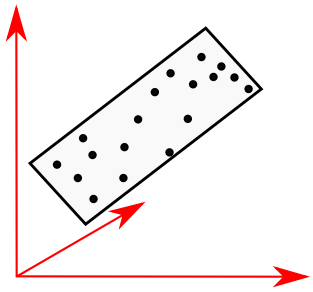


Figura 10. El espacio de búsqueda está en $\mathbb{R}^3$ aún cuando solo se requieren 2 dimensiones para representarlo.

1..5. Índices aproximados para búsqueda de similitud.

Para el problema en bajas dimensiones existen índices exactos que lo solucionan e.g. Yianilos (1993b) Navarro (1999) Chávez y Navarro (2000), entonces el problema principal es tratar con la maldición de la dimensión. El problema fue planteado originalmente en Minsky y Papert (1969) y a pesar de décadas de esfuerzos con soluciones basadas en índices exactos, los resultados se encuentran lejos de ser satisfactorios, incluso en altas dimensiones pueden llegar a tener peor tiempo de consulta con respecto a la búsqueda secuencial. Esta situación llevó a explorar la versión aproximada de la búsqueda del vecino más cercano (ANN por sus siglas en inglés) donde se sacrifican algúnas respuestas correctas con el objetivo de mejorar el tiempo de consulta.

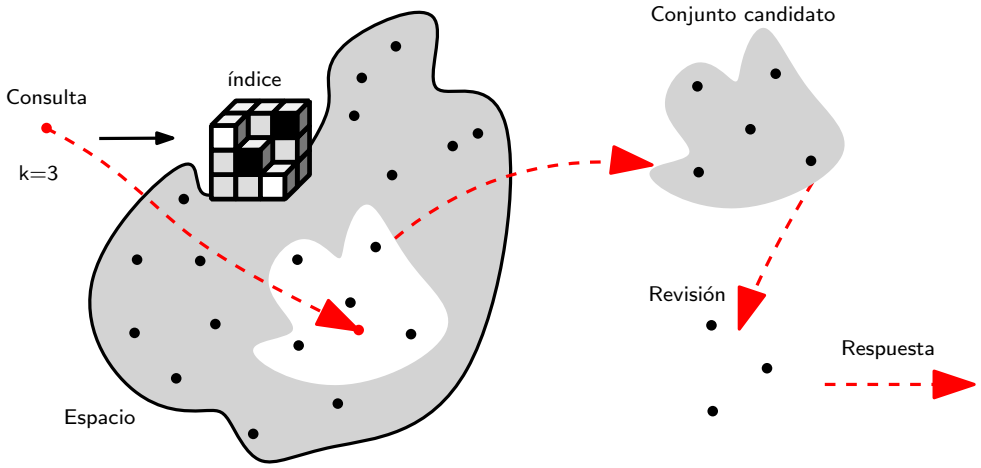


Figura 11. Ilustración del proceso de búsqueda en ANN.

El proceso en general para los índices aproximados se muestra en la Figura 11 donde se tiene

como interfaz entre una consulta y el conjunto P al índice, el cual tiene como tarea recuperar un subconjunto de puntos candidatos $C \subset P$ con $|C| \geq k$, donde k es el número de vecinos. Para que un índice sea eficiente en tiempo de consulta, el tamaño del conjunto candidato $|C|$ debe mantenerse lo más cercano posible a k debido a que en el módulo de revisión para cada elemento c del conjunto C se evalua su distancia $d(q, c)$ y después se ordenan de manera ascendente para tomar como respuesta los primeros k del ordenamiento.

Los índices aproximados son evaluados con la medida *recall* definida como:

$$recall = \frac{|C \ \cap \ k \ vecinos \ más \ cercanos|}{|k \ vecinos \ más \ cercanos|},$$

este es el indicador de eficacia de la estructura. Para analizar la rapidez de los algoritmos se observa el número de consultas que se pueden realizar por segundo, por lo general, existe un compromiso entre el recall y el tiempo de consulta, es decir a menor tiempo de consulta menor será el recall y viceversa (ver Figura 12.a). Otro aspecto importante es el tamaño del índice, en general, entre más grande es la estructura se espera un mejor recall (ver Figura 12.b).

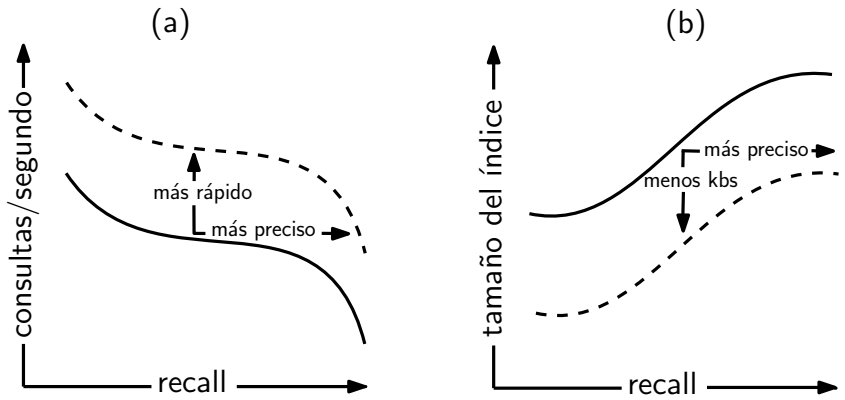


Figura 12. Comportamiento de algoritmos aproximados.

1..5.1. Índices epsilon-aproximados para búsqueda de similitud.

Existe un subconjunto de técnicas en los índices aproximados que tienen la propiedad garantizar el encontrar un punto $p \in S$ que está en un vecindario $\epsilon-$aproximado de un punto consulta q, donde, para todo $p' \in S$, $d(p, q) \leq (1 + \epsilon)d(p', q)$. Estos algoritmos son particularmente eficaces en aplicaciones donde no se requiere recuperar el vecino más cercano y es suficiente con un vecino bastante cercano, e. g. las recomendaciones en un sistema de ventas como ebay.

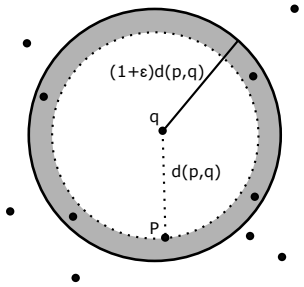


Figura 13. Cualquier punto dentro del $\epsilon-$vecindario puede ser considerado como respuesta a la consulta q.

En (Indyk y Motwani, 1998) se propuso la técnica conocida como “Locality-sensitve hashing”(LSH) la cual se basa en la proyección aleatoria del espacio original a una tabla *hash*, donde si dos puntos están cercanos en el espacio original colisionaran en un bin de la tabla *hash* con alta probabilidad.

Supongamos una familia de funciones hash H en un espacio métrico $M = (X, d)$ se dice que H es $(r, (1+\epsilon)\, r, p_1, p_2)$ $-$sensitiva si para todo $x, y \in X$ se cumple:

$$\begin{aligned} &\bullet \;\; si \;\; d(x,y) \leq r, \;\; entonces \;\; Pr_{h\sim H}\left[h\left(x\right) = h\left(y\right)\right] \geq p_1 \\ &\bullet \;\; si \;\; d(x,y) \geq (1+\epsilon) r, \;\; entonces \;\; Pr_{h\sim H}\left[h\left(x\right) = h\left(y\right)\right] \leq p_2 \end{aligned} \tag{3}$$

donde $\epsilon > 0$ y $0 < p_1, p_2 < 1$ y para que H tenga sentido se tiene que cumplir $p_1 > p_2$. Informalmente la idea es que entre más cerca estén dos puntos, es más probable que colisionen en la tabla hash.

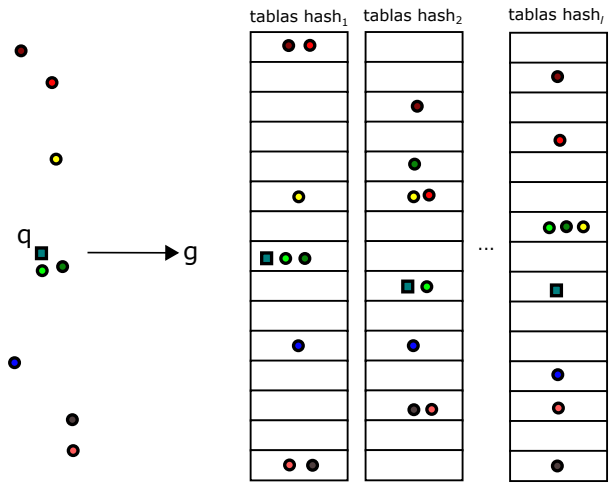


Figura 14. Ilustración de consulta en LSH.

La estructura LSH funciona de la siguiente manera. Se tiene una familia $H\left(r,\left(1+\epsilon\right)r,p_1,p_2\right)-$sensitiva de la cual se construye la familia G de la siguiente manera:
Se muestrean k funciones $h_1,h_2,...,h_k$ de H de manera independiente, de tal forma que para un punto x y una función $g\in G$ se tiene $g(x)=\left(\left(h_1(x)\right),\left(h_2(x)\right),...,\left(h_k(x)\right)\right)$. Se puede observar que la nueva familia H^k es también $\left(r,\left(1+\epsilon\right)r,p_1^k,p_2^k\right)-$sensitiva.

Se fija el parámetro $k=log_{(1/p_2)}N$ para obtener las siguientes probabilidades de colisión entre dos puntos:

- $\frac{1}{N}$ a una distancia $(1+\epsilon)r$
- $\frac{1}{N^\rho}$ a una distancia r

donde $\rho=\frac{ln1/p_1}{ln1/p_2}$. Durante el preproceso, cada $p\in S$ se guarda en el bin $g(p)$. Cuando se realiza una consulta del objeto q, se busca en el bin $g(q)$. Para analizar la estructura, se puede observar que el número esperado de puntos que se encuentran a una distancia mayor que $(1+\epsilon)r$ es a lo mas 1 debido a la elección de k. Por otra parte, la probabilidad de encontrar un vecino a una distancia a los más r es $\frac{1}{N^\rho}$, y para aumentar esta probabilidad y hacerla constante, se construyen N^ρ tablas *hash* de forma independiente. Como resultado se tiene una estructura de datos con un espacio de $O(nN+N^{1+\rho})$ y un tiempo de consulta $O(N^\rho)$ con $\rho=\frac{ln1/p_1}{ln1/p_2}$.

Como ejemplo en Indyk y Motwani (1998) construyen una familia de funciones H para un hipercubo $\{0,1\}^n$ normado por la distancia de Hamming, con $H=\{h_1,h_2,...,h_n\}$, donde $h_i(x)=x_i$. Se puede comprobar que esta familia es $(r,(1+\epsilon)r,1-r/n,1-cr/n)$-sensitiva. Con esta familia y tomando el resultado del párrafo anterior resuelven el problema $\epsilon-$ANN para la distancia de Hamming en espacio $O(N^{1+1/1+\epsilon})$ y con tiempo de consulta $O(N^{1/1+\epsilon})$. Además se mostró que para las métricas L_p con $1\leq p\leq 2$ se puede lograr $\rho=1/(1+\epsilon)$. Despúes en (Andoni y Indyk, 2006) se mostró que para l_2 se puede obtener $\rho=1/(1+\epsilon)^2$. Por último en (O'Donnell *et al.*, 2014) se encontró una cota inferior para l_1 con $\rho\geq 1/(1+\epsilon)-o(1)$ y para l_2 con $\rho=1/(1+\epsilon)^2-o(1)$ que emparejan las cotas superiores obtenidas en (Indyk y Motwani, 1998) y (Andoni y Indyk, 2006), es decir estas cotas son óptimas.

1..5.2. Índices aproximados probabilísticos para búsqueda de similitud.

Para dar respuesta al problema de búsqueda del vecino más cercano recientemente el enfoque ha sido construir índices aproximados probabilísticos, los cuales no tienen garantías de distancia y miden su rendimiento experimentalmente.

1..5.2.1. Índices aproximados probabilísticos basados en árboles

Muchas técnicas para la búsqueda por similitud aproximadas han aplicado el uso de árboles como método de acceso, donde la idea es parar la búsqueda de manera prematura o descartar la revisión de regiones con posibilidades limitadas con el objetivo de mantener candidatos relevantes a una consulta. Los algoritmos que se basan en árboles con mejor rendimiento son rpforest (Rpforest, 2015), mrpt (Hyvonen *et al.*, 2016) y annoy (Annoy, 2013).

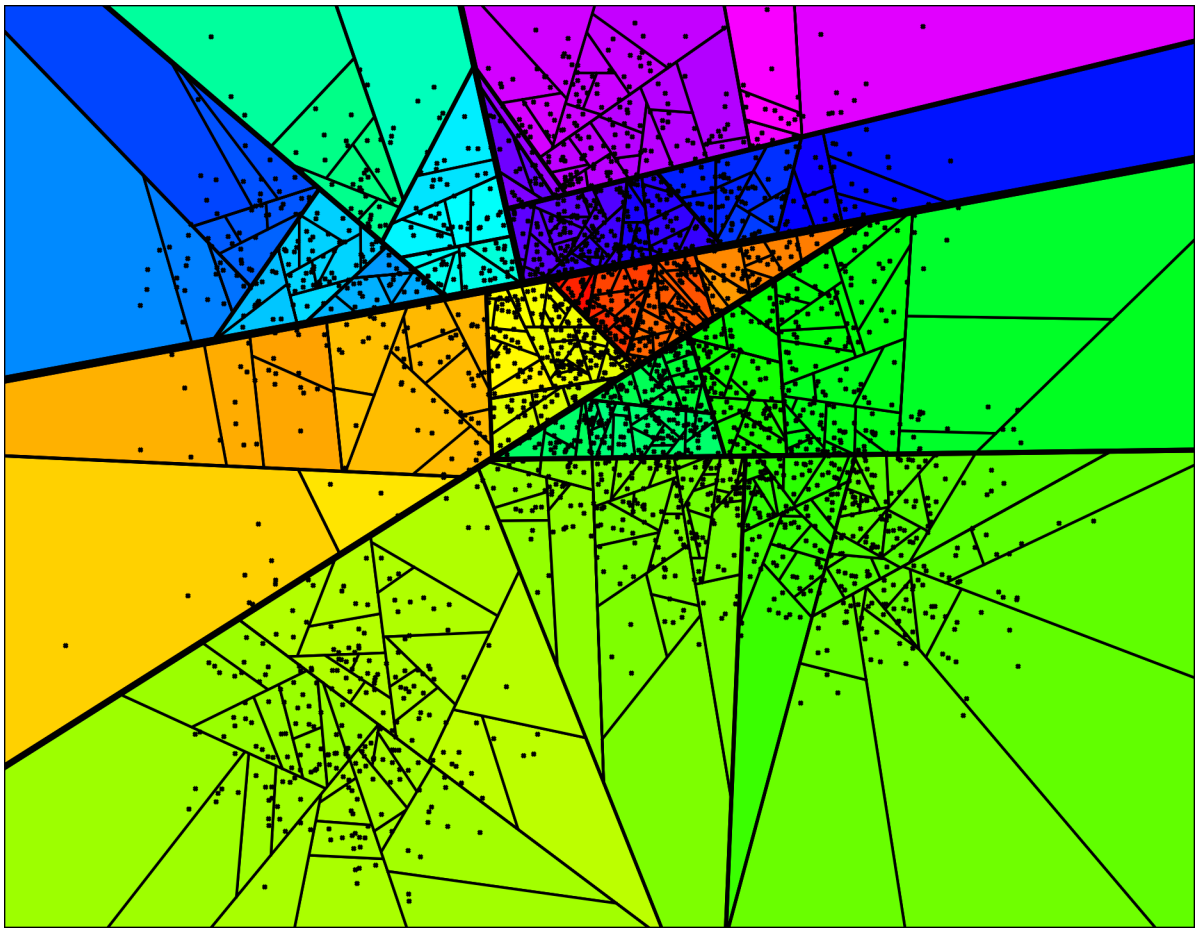


Figura 15. Representación del árbol *annoy*.

La idea en estas técnicas es la construcción de un árbol binario donde cada nodo interno representa la partición del espacio utilizando un hiperplano aleatorio. Un hiperplano es generado eligiendo dos puntos aleatoriamente y tomando el hiperplano equidistante entre ellos. Posteriormente, los puntos que queden a la izquierda del hiperplano serán parte del subárbol izquierdo y los que queden del lado derecho del subárbol derecho. En la Figura 15 se muestra un ejemplo de las particiones en $\mathbb{R}^2$.

La construcción de un árbol *annoy* se muestra en el Algoritmo 4, una de las restricciones en cada hoja del árbol es que solo un máximo número de m objetos son vinculados a la hoja. Para realizar una búsqueda en este árbol se comienza en la raíz y se navega hacia el subárbol derecho ó izquierdo dependiendo de que lado se encuentre el punto de consulta con respecto al hiperplano de la raíz, este mismo procedimiento se realiza recursivamente con los siguientes subárboles hasta llegar a una hoja. Al llegar a una hoja se toman los objetos de esa hoja como el conjunto de candidatos y se evalúa la distancia entre la consulta y cada uno de los candidatos y reporta como respuesta los k más cercanos a la consulta.

Algoritmo 4 Construcción de annoy

Entrada: S conjunto de puntos, m máximo número de puntos en una hoja, r nodo padre
Salida: T raíz del árbol annoy

```
Función CONSTRUIRANNOY(S,m,r)
    Si (|S| < m) Entonces
        r →Agregar(S)
        Regresa
    Fin Si
    Generar el hiperplano v y nodo n_v
    izquierdo,derecho=∅
    Para todo p ∈ S Hacer
        Si signo(p · v) > 0 Entonces
            izquierdo ∪ p
        De otro modo
            derecho ∪ p
        Fin Si
    Fin Para
    n_v →AgregarHijo(ConstruirAnnoy(izquierdo,m,n_v))
    n_v →AgregarHijo(ConstruirAnnoy(derecho,m,n_v))
     Regresa n_v
Fin Función
```

Un problema de esta estructura es que aún cuando dos objetos sean cercanos pueden estar en diferentes lados de un hiperplano aleatorio y como consecuencia terminar en diferentes hojas. Para solucionar este problema se proponen varias heurísticas para revisar varias hojas y cons-

truir K árboles para definir un bosque de árboles *annoy*. El objetivo de este bosque es unir los conjuntos candidatos de cada árbol y responder con los k más cercanos a la consulta.

Algoritmo 5 Búsqueda en annoy

Entrada: q objeto de consulta, T árbol annoy
Salida: O vecino más cercano a q

Función BUSCARANNOY(q,T)
$candidatos = BuscarCandidatos(q, T)$
ordenar candidatos
Regresa el primer candidato
Fin Función
Función BUSCARCANDIDATOS(q,T)
$resultado = \{\}$
Si $(T == \emptyset)$ **Entonces**
Regresa
Fin Si
El hiperplano $v = T.v$
Si $signo(q \cdot v) > 0$ **Entonces**
$BuscarCandidatos\,(q, T.izquierdo)$
De otro modo
$BuscarCandidatos\,(q, T.derecho)$
Fin Si
Regresa $resultado \cup T.puntos$
Fin Función

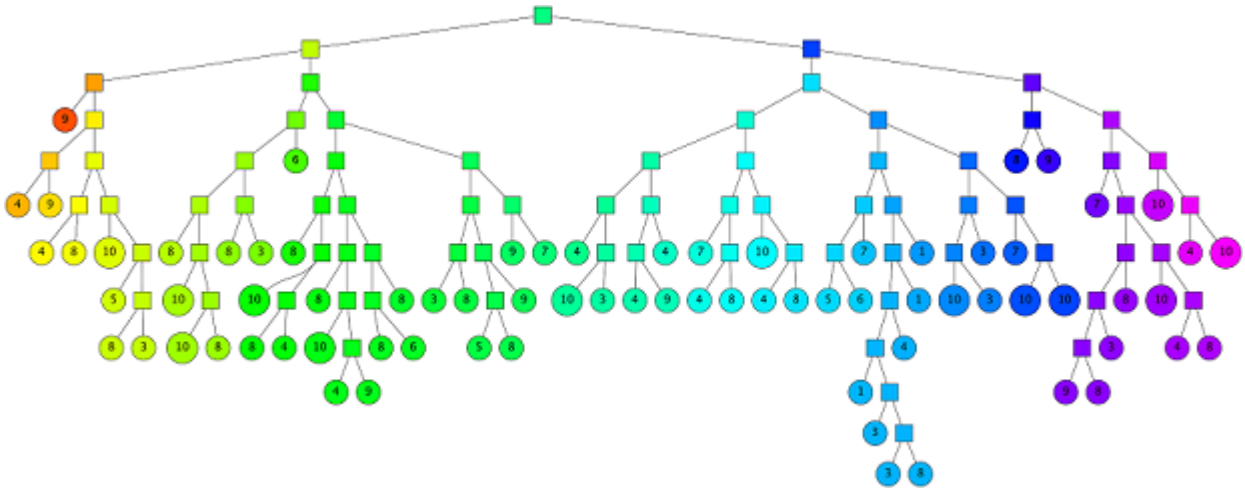


Figura 16. Ilustración de árbol *annoy*.

1..5.2.2. Índices aproximados probabilísticos basados en hashing

LSH es la técnica más popular en este tipo de índexado, la idea es que objetos cercanos sean asignados, con alta probabilidad, en el mismo bin en el índice o tabla hash Indyk y Motwani (1998)

Andoni y Indyk (2006) suponiendo que la familia de funciones hash utilizada sea local-sensitiva. Son varias las familias de funciones definidas que cumplen con las condiciones LSH bajo varios tipos de distancias entre puntos.

La familia de funciones con mejor rendimiento es el hiperplano-LSH(simHash) Charikar (2002b) la cual se diseñó con la idea de aproximar la distancia de coseno entre dos vectores. La idea básica de esta técnica es elegir un hiperplano aleatorio (definido por un vector unitario r) y usar el hiperplano para hacer hash en los vectores de entrada.

Dado un vector de entrada v y un hiperplano definido por r, definieron la función $h(v) = signo(v \cdot r)$. Esto es $h(v) = \pm 1$ dependiendo de que lado del hiperplano se encuentra v. Cada posible elección de r define una nueva función. Supongamos la familia de funciones H con todas las funciones de este tipo y escogemos de aquí una función h aleatoriamente. Se demostró que para dos vectores u, v, $Pr[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}$, donde $\theta(u, v)$ es el ángulo entre los vectores u y v. $1 - \frac{\theta(u,v)}{\pi}$ es una aproximación muy buena de $\cos(\theta(u, v))$ para el rango de 0 a 1.

Para una función h se genera solamente un bit, la idea es construir k funciones y representar los vectores como cadenas de k bits. Los bits de dos vectores coinciden con probabilidad proporcional al coseno del ángulo entre ellos. Al igual que *annoy* un punto puede ser cercano y quedar en un lado distinto de un hiperplano, entonces para solucionar esto se construyen varias tablas hash. En una consulta se busca en cada tabla hash y se unen los candidatos de cada tabla para formar un conjunto candidato de todos las tablas. De este conjunto candidato se extraen los k más cercanos a la consulta como respuesta.

1..5.2.3. Índices aproximados probabilísticos basados en permutaciones

Los índices basados en permutaciones inicalmente se propusieron en (Chavez Gonzalez *et al.*, 2008) como un nuevo paradigma para la búsqueda aproximada del vecino más cercano. En estos índices los objetos en la colección y el objeto de consulta son representados como una permutación apropiada de un conjunto de objetos de referencia, y la distancia es aproximada comparando su representación. La idea es que si las permutaciones son parecidas entonces los objetos serán similares.

En el método propuesto en Chavez Gonzalez *et al.* (2008), las permutaciones son generadas ordenando los objetos de referencia de forma ascendente con respecto a la distancia a los objetos.

En una búsqueda se representa un objeto de consulta q de la misma forma, i. e. se ordenan los objetos de referencia con respecto a la distancia a q. Luego utilizando una medida de similitud entre dos permutaciones, es posible determinar los objeto más cercanos a q. Los autores utilizaron una medida basada en la distancia de Sperman Rho con la cual realizan un ordenamiento ascendente de una porción de la colección para dar una respuesta a una consulta con los primeros k en el ordenamiento.

Otro trabajo que utiliza la representación con permutaciones es MI-File Amato *et al.* (2014b), la diferencia con (Chavez Gonzalez *et al.*, 2008) es que utilizan un índice invertido. Un índice invertido es una estructura de datos que brinda el acceso eficiente a objetos descritos por un conjuntos de valores discretos (e. g. números o letras), esto se hace mapeando cada posible valor con la lista de todos los objetos con ese valor. Para este trabajo los valores descriptores son los objetos de referencia. Además, limitan el tamaño de la permutación de los objetos con los primeros k_x de la permutación completa y la permutación de la consulta con k_q donde $k_q < k_x$, esto con el objetivo de mejorar en espacio en memoria y tiempo de consulta.

1..5.2.4. Índices aproximados probabilísticos basados en el gráfo KNN

Recientemente los métodos basados en gráfos han ganado atención (Malkov *et al.*, 2014) (Malkov y Yashunin, 2018) (Harwood y Drummond, 2016), la filosofía detrás de estos métodos es que “el vecino más cercano de mi vecino muy probablemente sea mi vecino más cercano”. Estos métodos construyen un gráfo KNN previo a una consulta que aproxima una triangulación Delaunay, en una búsqueda encuentran un conjunto candidato de vecinos para un punto consulta de alguna manera, para después revisar iterativamente si los vecinos de estos candidatos son más cercanos a la consulta.

En (Malkov *et al.*, 2014) construyeron un gráfo aproximado KNN manteniendo únicamente un número máximo m de enlaces para cada nodo, que idealmente si se cubrieran todos los vecinos voronoi no se tendría el problema de caer en un mínimo local (ver Figura 17). Sin embargo los algoritmos conocidos para determinar una triangulación Delaunay exacta en $\mathbb{R}^n$ requieren un tiempo de construcción $O(NlogN + N^{\lfloor \frac{n}{2} \rfloor})$, además se requiere un espacio de $N^{\lfloor \frac{n}{2} \rfloor}$. Para no caer fácilmente en un mínimo local y afectar considerablemente el recall de la técnica, en (Harwood y Drummond, 2016) intentaron resolver el problema comenzando la búsqueda no en un punto aleatorio sino en un punto elegido previamente a la navegación del gráfo en una consulta. En (Malkov

y Yashunin, 2018) se reinicia varias veces la búsqueda para mejorar la calidad de las respuestas compartiendo información entre búsquedas de nodos visitados para no revisar dos veces el mismo nodo.

Posteriormente, en (Malkov y Yashunin, 2018) se modificó (Malkov *et al.*, 2014) por una estructura de capas jerárquicas que separa los enlaces largos entre nodos en capas superiores y conforme se baja en la estructura aparecen enlaces cada vez más cortos entre nodos. Con esta modificación y al igual que en (Malkov y Yashunin, 2018) afinando el punto de entrada en cada capa se mejora significativamente el recall al punto que esta técnica es considerada el estado del arte en índices aproximados probabilísticos.

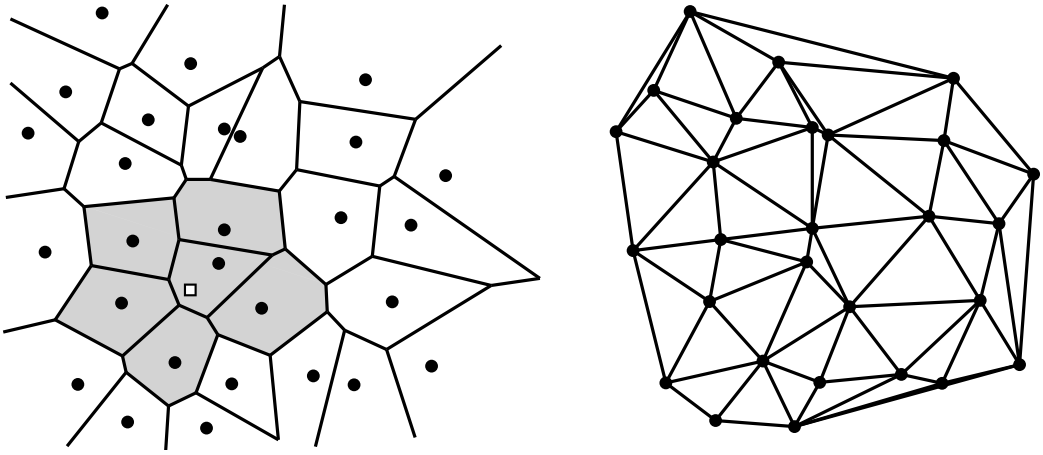


Figura 17. Diagrama voronoi y triangualción Delaunay. A la izquierda el cuadrado representa una consulta y en gris las regiones que se requieren revisar para responder la consulta. A la derecha la triangulación Delaunay (grafo Voronoi).

Capítulo 2. Estado del arte

Los algoritmos considerados en el estado del arte son los que reportan mejor recall y menor tiempo de consulta. Estudiamos dos clases de técnicas, las que se implementan en memoria principal (RAM) y las que se implementan en memoria secundaria ya que aún en un disco de estado sólido (SSD) la lectura y escritura es ordenes de magnitud más lento.

2..1. Memoria principal.

El estado del arte en índices en memoria principal es llamado HNSW (Hierarchical Navigable Small Worlds) propuesto en Malkov y Yashunin (2018). La idea es la construcción de un gráfo KNN aproximado y realizar las búsquedas con las rutas formadas por las aristas entre nodos. Una de las principales propiedades de este trabajo es la estructura en capas jerárquicas del gráfo, esto es todos los puntos aparecen en la capa 0 o nivel inferior. Cada punto aparecerá en los l siguientes niveles donde para cada punto se define $l = \lfloor -\ln(unif(0,1)\rfloor \cdot m_l$ con m_l el nivel máximo, es decir, entre más alto sea el nivel menos probable es que aparezca un punto.

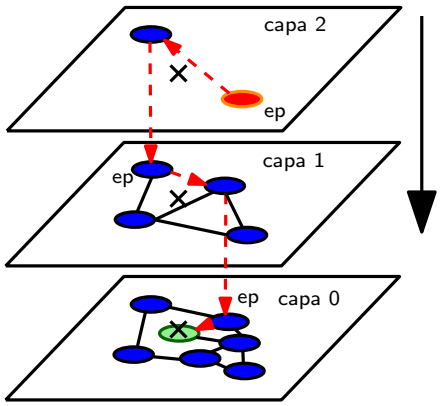


Figura 18. Ilustración de idea en el HNSW.

El objetivo de esta estructura jerárquica es poder encontrar un nodo de entrada ep que es el más cercano en un cierto nivel a q, para después en un nivel inferior comenzar la búsqueda en el nodo ep, que se espera resulte en tener que revisar menos nodos en el nivel inferior para terminar la búsqueda. En cada nivel se mantiene una lista temporal de ef nodos más cercanos en la capa y se va actualizando conforme la búsqueda desciende en la estructura. Despúes de terminar la búsqueda en el nivel inferior se toman como respuesta los k más cercanos a q.

Para construir el grafo se insertan secuencialmente todos los elementos del conjunto P. Una inserción puede ser vista como una búsqueda de los k vecinos en cada nivel y el uso de una heurística que depende del parámetro $efConstruction$. Por cada elemento encontrado como vecino de un punto en una capa se puede considerar la existencia de una arista entre el punto y su vecino, pero para mantener bajo el número de aristas en el gráfo se consideran únicamente un número máximo M_{max0} de aristas para la capa inferior y M_{max} para cualquier otra capa, los autores recomiendan $M_{max0} = 2M_{max}$ para obtener un recall satisfactorio. Es importante mantener este parámetro lo más bajo posible ya que el mayor consumo de memoria en la estructura está dado por el número de conexiones entre los nodos, esto es $M_{max0} \cdot m_l \cdot M_{max}$ para cada elemento. Se recomienda asignar M_{max} en el rango de 6 a 48 lo que hace que cada nodo tenga un peso de 60 y 450 bytes en conexiones por elemento en P (ver el Algoritmo 7).

Algoritmo 6 Seleccionar Vecinos

Entrada: elemento base q, candidatos C, número de vecinos M
Salida: los M vecinos más cercanos a q
  Función SELECCIONARVECINOS(q,C,M)
    Regresa los M vecinos más cercanos de q en C
  Fin Función

Algoritmo 7 Insertar en hnsw

Entrada: grafo $hnsw$, nuevo elemento q, número de conexiones M, número máximo de conexiones para cada elemento por capa $Mmax$, tamaño de la lista dinámica de candidatos $efConstruction$, factor de normalización para generación de nivel ml

Salida: $hnsw$ actualizado con el elemento q insertado

Función INSERTARHNSW($hnsw$,q,M,$Mmax$,$efConstruction$,ml)

$W \leftarrow \emptyset$ lista temporal para los elementos más cercanos encontrados

$ep \leftarrow$ punto de entrada a $hnsw$

$L \leftarrow$ nivel del punto de entrada ep o capa más alta del $hnsw$

$l \leftarrow \lfloor -\ln(unif(0..1))ml \rfloor$ el nivel del elemento q

Para $l_c \leftarrow L \ldots l+1$ **Hacer**

$W \leftarrow BuscarEnCapa(q, ep, ef = 1, l_c)$

$ep \leftarrow$ el vecino más cercano en W a q

Fin Para

Para $l_c \leftarrow min(L, l) \ldots 0$ **Hacer**

$W \leftarrow BuscarEnCapa(q, ep, efConstruction, l_c)$

$vecinos \leftarrow SeleccionarVecinos(q, W, M, l_c)$

agregar conexiones bidireccionales de vecinos a q en la capa l_c

Para cada $e \in vecinos$ reduce conexiones si es necesario **Hacer**

$eConn \leftarrow vecindad(e)$ en la capa l_c

Si $|eConn| > Mmax$ se reducen las conexiones de e, si $lc = 0$ $Mmax = Mmax0$ **Entonces**

$eNewConn \leftarrow SeleccionarVecinos(e, eConn, Mmax, l_c)$

actualizar el vecindario de e en la capa l_c

Fin Si

Fin Para

$ep \leftarrow W$

Fin Para

Si $l > L$ **Entonces**

tomar ep como el punto de entrada al $hnsw$

Fin Si

Regresa $hnsw$

Fin Función

Algoritmo 8 Buscar en capa

Entrada: elemento consulta q, punto de entrada ep, número de elementos cercanos a q a regresar ef, capa l_c
Salida: los ef vecinos más cercanos a q en la capa l_c

```
Función BUSCARENCAPA(q, ep, ef, l_c)
    v ← ep conjunto de elementos visitados
    C ← ep conjunto de candidatos
    W ← ep lista dinámica de los vecinos más cercanos encontrados
    Mientras |C| > 0 Hacer
        c ← obtener el elemento más cercano a q en C
        f ← obtener el elemento más lejano a q en W
        Si distancia(c, q) > distancia(f, q) Entonces
            salir todos los elementos en W se evaluaron
        Fin Si
        Para cada e ∈ vecindario(c) en la capa l_c Hacer
            Si e ∉ v Entonces
                v ← v ∪ e
                f ← obtener el elemento más lejano a q en W
                Si distancia(e, q) < distance(f, q) o |W| < ef Entonces
                    C ← C ∪ e
                    W ← W ∪ e
                    Si |W| > ef Entonces
                        quitar el elemento más lejano a q en W
                    Fin Si
                Fin Si
            Fin Si
        Fin Para
    Fin Mientras
    Regresa W
Fin Función
```

Una búsqueda empieza en el nivel superior, donde se espera haya muy pocos nodos, de esta capa se obtiene un lista L de candidatos temporales de tamaño $efSearch$ que se requiere sea mayor a los k vecinos que se estan buscando, entre mayor sea $efSearch$ se espera mejor recall, incrementando el tiempo de búsqueda. En la siguiente capa el punto de entrada ep será el punto más cercano en L a la consulta, en esta capa se espera haya más nodos que en la capa superior. Posteriormente se actualiza L con los vecinos más cercanos a la consulta en esta capa y este proceso se repite hasta llegar al nivel inferior. Al terminar de actualizar la lista L en la capa 0 se obtienen los k objetos más cercanos a la consulta en la lista L como respuesta (ver Algoritmo 9).

Algoritmo 9 Búsqueda de los k vecinos más cercanos

Entrada: grafo $hnsw$, elemento consulta q, número de elementos más cercanos K, tamaño de la lista dinámica $efSearch$
Salida: los K vecinos más cercanos a q
  Función BUSQUEDAKNN($hnsw$, q, K, $efSearch$)
    $W \leftarrow \emptyset$ conjunto dinámico de elementos más cercanos
    $ep \leftarrow$ el punto de entrada del $hnsw$
    $L \leftarrow$ nivel del ep
    Para $lc \leftarrow L \ldots 1$ **Hacer**
      $W \leftarrow BuscarEnCapa(q, ep, ef = 1, lc)$
      $ep \leftarrow$ el vecino más cercano a q en W
    Fin Para
    $W \leftarrow BuscarEnCapa(q, ep, efSearch, lc = 0)$
    Regresa los K vecinos más cercanos a q en W
  Fin Función

Para observar el rendimiento del $hnsw$ con respecto a algunos algoritmos en la bibliografía utilizamos un conjunto de un millón de vectores en un espacio Euclidiano de 128 dimensiones. Construimos índices con diferentes parámetros para cada técnica y realizamos 1000 consultas en cada índice. En la Figura 19 se puede observar que se pueden realizar cerca de 1000 consultas en un segundo con un *recall* casi perfecto. En la Figura 20 se observa que el tamaño requerido para este *recall* es de alrededor de un gigabyte. Este algoritmo es considerado el estado del arte en memoria principal pero debido al requerimiento en tamaño no escala a bases de datos más grandes, por ejemplo, para una base de datos de $1,000,000,000$ de vectores en $\mathbb{R}^{128}$ se requieren $1,000$ gigabytes . Además técnicas como $hnsw$ no toman en cuenta los accesos a memoria, por lo tanto, no tienen buen rendimiento en memoria secundaria. Las técnicas para memoria secundaria generalmente limitan los accesos a memoria utilizando la estructura de índice invertido.

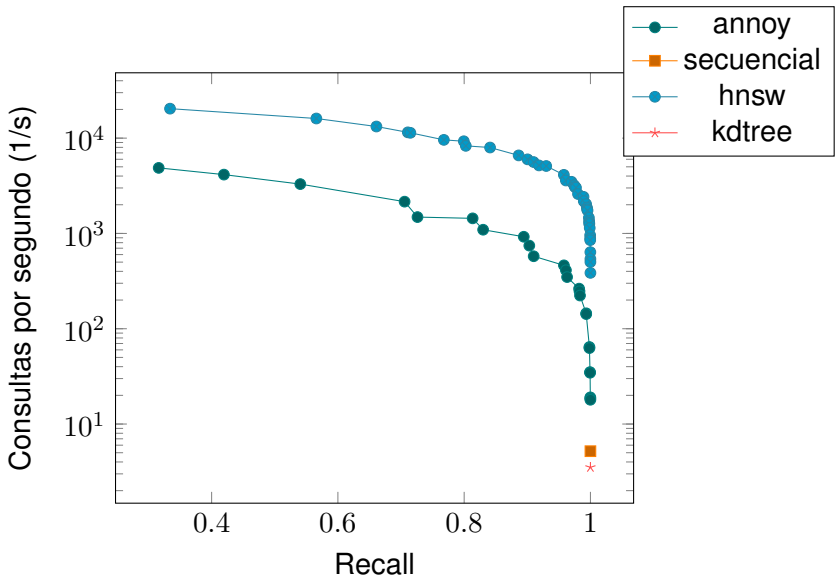


Figura 19. Recall-Consultas por segundo (1/s)

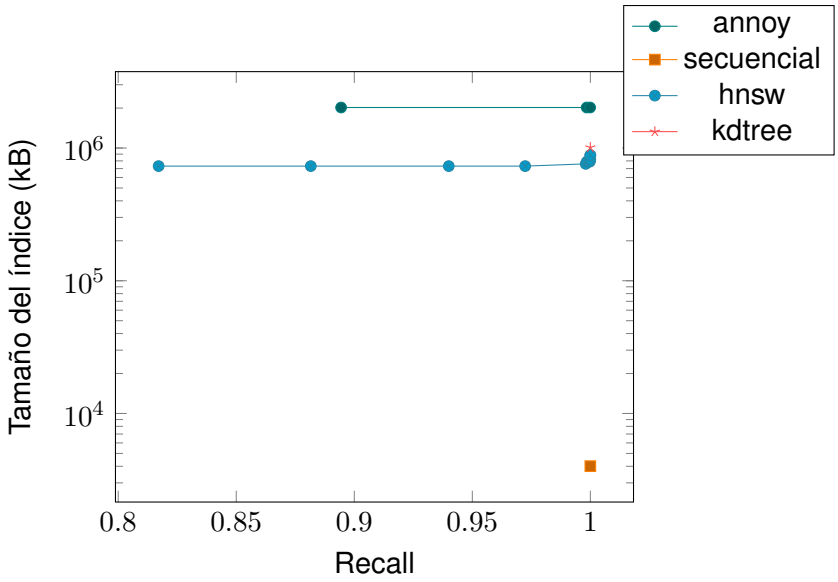


Figura 20. Recall- Tamaño del índice (kB)

2..2. Memoria secundaria.

En memoria secundaria los algoritmos en la literatura se basan en *product quantization (pq)* y el algoritmo con el mejor rendimiento actualmente se presenta en (Jégou *et al.*, 2011). La idea en ese trabajo es descomponer el espacio en un producto de subespacios de dimensión más baja y cuantizar cada subespacio por separado. Cada vector es representado por un código en cada uno de los subespacios. La distancia Euclidiana entre dos vectores puede estimarse de manera eficiente con la distancia entre sus códigos.

Un cuantizador es una función f que mapea un vector $x \in \mathbb{R}^n$ a un vector $f(x) \in C = \{c_i; i \in I\}$, donde el conjunto de índices $I = \{1, ..., k\}$. Los valores de c_i son llamados centroides. El conjunto de valores que puede tomar C se le conoce como libro de códigos y es de tamaño k.

El conjunto de vectores V_i mapeados a un índice i puede ser visto como una celda voronoi y definida como

$$V_i := \{x \in \mathbb{R}^n : f(x) = c_i\}. \tag{4}$$

Para ser eficiente ante la maldición de la dimensión se propone partir el espacio $\mathbb{R}^n$ en m subespacios de dimensión $n* = n/m$. Los subvectores se cuantizan utilizando m cuantizadores distintos. Un vector x se mapea de la siguiente manera:

$$(x_1, ..., x_n) = (f_1(u_1(x)), ..., f_m(u_m(x))) \tag{5}$$

donde $u_1(x) = (x_1, ..., x_{n*}), ..., u_m(x) = (x_{n-n*+1}, ..., x_n)$ y f_j es el cuantizador de baja dimensión asociado con el $j-$ésimo subvector. El libro de códigos es definido por el producto cartesiano:

$$C = C_1 \times ... \times C_m \tag{6}$$

En ese trabajo se propone el uso de una aproximación a la distancia Euclidiana que se define por:

$$\hat{d}(x, y) = d(x, f(y)) = \sqrt{\sum_j d(u_j(x), q_j(u_j(y)))^2} \tag{7}$$

Para mejorar en tiempo de consulta proponen utilizar la aproximación de la distancia y un índice invertido. La idea es utilizar el código de la cuantización de un vector como entrada a un índice invertido, con el objetivo de obtener un conjunto candidato rápidamente. Una vez que se tiene el conjunto candidato se refina la búsqueda utilizando una cuantización f_p del residuo el cual se define como:

$$r(y) = y - f(y) \tag{8}$$

El estimador de $d(x, y)$, donde x es la consulta y y un vector en la base de datos, está dado como $\ddot{d}(x, y)$ la cual se define por:

$$\ddot{d}(x, y) = d\left(x - f(y), f_p(y - f(y))\right) \tag{9}$$

En la práctica se calcula de la siguiente manera:

$$\ddot{d}(x, y)^2 = \sum_j d\left(u_j(x - f(y)), f_{pj}(u_j(y - f(y)))\right)^2 \tag{10}$$

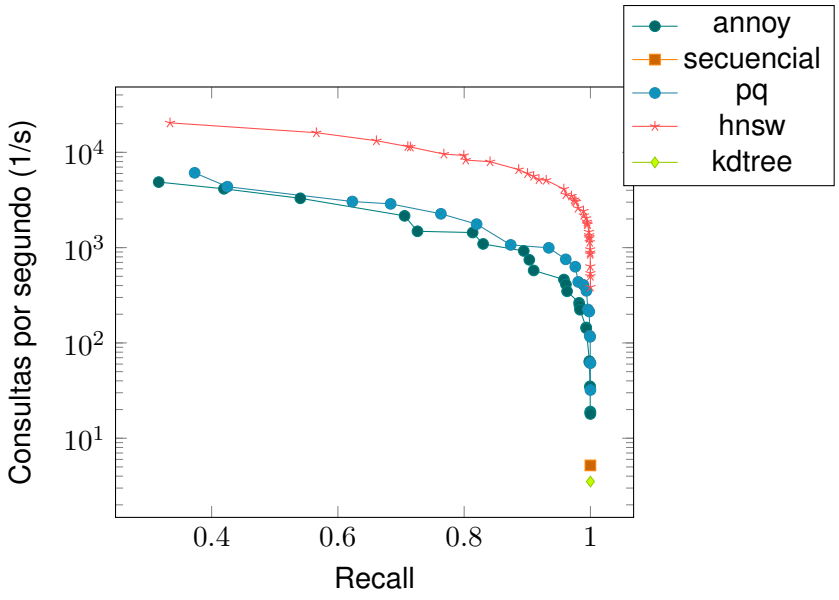


Figura 21. Recall-Consultas por segundo (1/s)

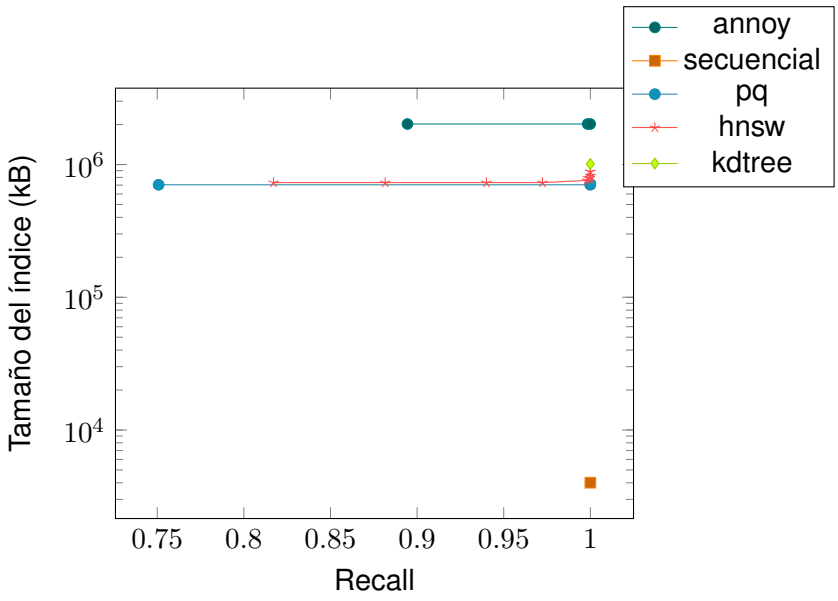
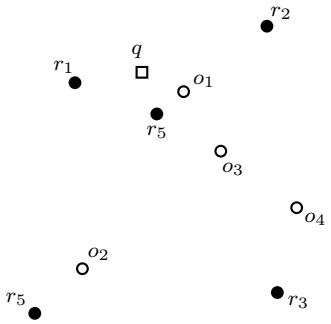


Figura 22. Recall-Tamaño del índice (kB)

La técnica *pq* es considerada el estado del arte en índices para memoria secundaria, el rendimiento comparado con otras técnicas de la literatura se muestra en las Figuras 21 y 22. Se puede observar en la Figura 21 que con un *recall* casi perfecto se pueden lograr alrededor de 500 consultas. Además se requiere de un índice de casi 800 megabytes en tamaño para obtener un *recall* casi perfecto. Esta técnica tiene la desventaja de que solamente se puede aplicar en espacios vectoriales pero no en otros espacios métricos, por ejemplo, en nubes de puntos.

Para espacios métricos generales, es decir, donde la única información disponible es la distancia entre cualquier pareja de puntos, la técnica que ha reportado mejor rendimiento es MI-File (Amato *et al.*, 2014a). Dado el dominio X, la función distancia $d : X \times X \to \mathbb{R}$ y el conjunto de referencias $R \subset X$ de tamaño m $(r_1, ..., r_m)$ elegidos aleatoriamente de X.



$x_1^* = (3, 2, 4, 5, 1)$ $\quad d^*(q, x_1^*) = 1 + 1 + 0 + 0 + 0 = 2$
$x_2^* = (4, 5, 2, 1, 3)$ $\quad d^*(q, x_2^*) = 2 + 1 + 1 + 0 + 0 = 4$
$x_3^* = (4, 2, 3, 5, 1)$ $\quad d^*(q, x_3^*) = 2 + 0 + 3 + 0 + 1 = 6$
$x_4^* = (4, 3, 1, 5, 2)$ $\quad d^*(q, x_4^*) = 2 + 2 + 2 + 4 + 2 = 12$
$q = (2, 3, 4, 5, 1)$

Figura 23. Ejemplo del mapeo al espacio de permutaciones. a) los puntos negros son los puntos referencia, los puntos blancos los objetos en la base de datos y el cuadro blanco una consulta. b) La transformación de los objetos al espacio de permutaciones. c) La distancia SFD en el mapeo.

Se representa $x \in X$ como el ordenamiento de los objetos de referencia R de acuerdo con la distancia a x. Formalmente, dado $x \in X$, x es representado como el objeto $x^* = p_1, ..., p_m$, donde p_i es la posición del objeto referencia r_i en el ordenamiento. Por ejemplo, si $x^*(4) = 3$, la referencia r_4 es la tercera más cercana a x.

La similitud entre dos representaciones x_1^* y x_2^* es calculada con la distancia Spearman Footrule (SFD), que se define de la siguiente manera:

$$SFD(x_1^*, x_2^*) = \sum_{i=1...m} |x_1^*(i) - x_2^*(i)| \tag{11}$$

Adicionalmente, probaron con otras distancias para permutaciones como la distancia de Kendall Tau y la distancia Spearman Rho sin observar cambios en la calidad de las respuestas.

La técnica utiliza el índice invertido para implementarse en memoria secundaria. Las llaves de acceso a las listas invertidas son los objetos referencia en R. Las listas invertidas asociadas a una llave $r_i \in R$ es una lista de duplas $(x, x^*(i))$ para algún $x \in X$. Por ejemplo, si la dupla $(x, 7)$ se encuentra en la lista invertida asociada a r_i, indica que la referencia r_i es la séptima referencia más cercana a x (ver Figura 24).

$r_1 \rightarrow (x_1, 3), (x_2, 4), (x_3, 4), (x_4, 4)$
$r_2 \rightarrow (x_1, 2), (x_2, 5), (x_3, 2), (x_4, 3)$
$r_3 \rightarrow (x_1, 4), (x_2, 2), (x_3, 3), (x_4, 1)$
$r_4 \rightarrow (x_1, 5), (x_2, 1), (x_3, 5), (x_4, 5)$
$r_5 \rightarrow (x_1, 1), (x_2, 3), (x_3, 1), (x_4, 2)$

Figura 24. El índice invertido que se obtuvo del ejemplo en la Figura 23.

Para mejorar en tiempo de consulta y en espacio, limitan las representaciones de los objetos en la base de datos y la consulta. Esto lo hacen considerando únicamente las k_x referencias más cercanas para los objetos en la base de datos y los k_q referencias más cercanas a la consulta. Formalmente, $\Phi_{k_x}(x) = \{r_i | x^*(i) \leq k_x\}$, el proceso es el mismo para el objeto consulta con k_q. Para esta reducción de las representaciones se propone el uso de una aproximación de la distancia Spearman Footrule, la distancia Footrule Inducida (IFD), definida como:

$$IFD(x^*, q^*) = \sum_{r_i \in \Phi_{k_q}(q)} |x^*(i) - q^*(i)| \tag{12}$$

Hay que notar que la función IFD obtiene la distancia calculando la suma de la diferencia de posiciones de los objetos referencia en las representaciones de los objetos consulta y los objetos en la base de datos. Sin embargo, ahora un objeto referencia puede aparecer en la representación de una consulta y no en la representación de un objeto en la base de datos. Cuando esto ocurre se suma k_x a la suma en la función (ver la Figura 25).

Dado un objeto referencia r_i y su posición p en la representación de una consulta q. Proponen acceder solo a una parte de la lista invertida asociada a r_i. Se utiliza un umbral de diferencia máxima en la posición (MPD), para filtrar las duplas donde la posición de r_i en la representación se encuentre el rango $[p - MPD, p + MPD]$.

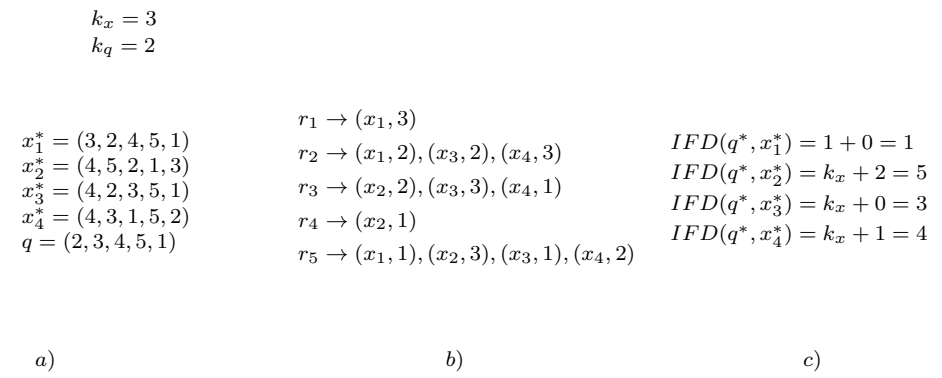


Figura 25. Indexando con las k_x referencias más cercanas a los objetos en la base de datos y consultando con las k_q referencias más cercanas. a) Representación completa de los objetos. b) Índice invertido. c)Distancia a la consulta en la transformación.

Para un conjunto de un millón de vectores de 64 dimensiones con $m = 2000$, $k_x = 200$, $k_q = 50$, $MPD = 50$ y revisando alrededor de 1000 distancias se reporta un recall de $90\,\%$. Esta técnica al no alcanzar niveles de recall cercanos al $100\,\%$, se propone para el uso de recuperación multimedia, en particular para la recuperación de imágenes.

Algunas de las desventajas de este trabajo son las siguientes:

- El número de referencias es limitado para las representaciones. Si se quisiera tener mejor recall aumentando las referencias en la permutación se requiere más memoria y tiempo en una consulta.
- La búsqueda de las referencias más cercanas se podrían acelerar utilizando un algoritmo auxiliar en memoria principal.
- El número de candidatos que se obtienen al filtrar una lista con el parámetro MPD no está acotado. Si el número de candidatos es grande, el tiempo de evaluación de IFD para cada candidato, incrementa significativamente el tiempo en responder una consulta.
- No reportan el número de consultas que se pueden realizar en un segundo. No se puede determinar si es competitivo con los índice para memoria principal.

MI-File es considerado el estado del arte en índices para memoria secundaria para espacios métricos generales. Aún cuando la técnica tiene limitaciones en tiempo de respuesta en una

consulta con respecto a otras técnicas como pq, MI-File tiene como ventaja el solo requerir de la existencia de una métrica para comparar parejas de objetos en un dominio, por ejemplo, el dominio de las cadenas con la distancia de Levenshtein.

Capítulo 3. Justificación

El diseño e implementación de índices aproximados para la búsqueda de similitud, es motivado por la cantidad de aplicaciones donde se presenta el problema. Además el incremento en el tamaño de las bases de datos favorece el interés por índices eficientes en memoria secundaria. A continuación, listamos algunas de estas aplicaciones.

3..1. Reconocimiento de patrones

En el reconocimiento de patrones, el algoritmo de los k vecinos más cercanos ($k-NN$) es un método no paramétrico utilizado para clasificación. La entrada consiste en un conjunto de entrenamiento con elementos etiquetados, y la salida es una etiqueta o membresía de una clase. Un objeto se clasifica por un voto de mayoría con sus vecinos, es decir, el objeto se asigna a la clase más común entre sus k vecinos más cercanos. Si $k = 1$, entonces el objeto simplemente se asigna a la clase de su vecino más cercano.

Los objetos de entrenamiento son vectores en un espacio de características multidimensionales, cada uno con una etiqueta de clase. En la fase de clasificación, k es una constante, y un vector sin etiquetar (una consulta o punto de prueba) se clasifica asignando la etiqueta que es más frecuente entre las k muestras de entrenamiento más cercanas a ese punto de consulta.

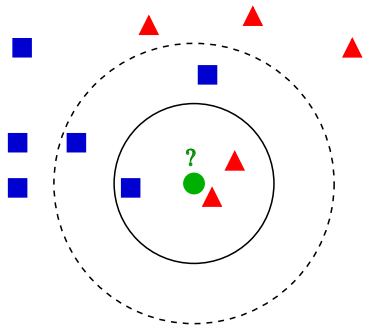


Figura 26. Ejemplo de clasificación $k-NN$.

En la Figura 26 se muestra un ejemplo de clasificación $k-NN$. La consulta (punto verde) debe clasificarse en cuadrados azules o triángulos rojos. Si $k = 3$ (círculo de línea continua) se asigna a los triángulos rojos porque hay dos triángulos y solo un cuadrado dentro del círculo interno. Si $k = 5$ (círculo de línea discontinua) se asigna a los cuadrados azules (3 cuadrados contra 2 triángulos dentro del círculo exterior).

3..2. Recuperación de multimedia

Una aplicación de recuperación de multimedia la una búsqueda con un fragmento de sonido consulta que puede ser dado por un tarareo, un silbido, tocando un instrumento o reproducción de una grabación. Para recuperar audio similar a la consulta, por lo general, se utilizan características geométricas tomadas del espectro del audio grabado. Típicamente, la representación del audio se realiza en dos dimensiones: horizontalmente la frecuencia (para el caso de música nota) y en el eje vertical la amplitud (el tono para música). Esta representación del audio pueden ser vista como una secuencia de puntos en el plano, sin embargo, resulta que esta representación no es robusta a ruido, traslaciones ni a escalas. Para realizar una recuperación robusta, generalmente se tiene que representar el audio como una secuencia de puntos en altas dimensiones.

Para poder realizar una búsqueda eficiente en una colección de millones de grabaciones de audio, se construye un índice con las secuencias representantes. Cada punto se inserta como una tripleta con el identificador del audio, las coordenadas del punto e información del tiempo en el que ocurre cada punto. En tiempo de búsqueda para cada punto en la representación de la consulta, se busca por su respectivo vecino más cercano y se recupera la tripleta correspondiente. Con la información del tiempo se realiza un ordenamiento de candidatos y se da una respuesta.

3..3. Filtrado colaborativo

El filtrado colaborativo proviene de la idea de que las personas a menudo obtienen las mejores recomendaciones de alguien con gustos similares a ellos. El filtrado colaborativo abarca técnicas para relacionar personas con intereses similares y hacer recomendaciones.

Los algoritmos de filtrado colaborativo a menudo requieren (1) la participación activa de los usuarios, (2) una manera fácil de representar los intereses de los usuarios y (3) algoritmos que pueden relacionar a personas con intereses similares.

Por lo general, el flujo de trabajo de un sistema de filtrado colaborativo es:

- Un usuario selecciona sus preferencias calificando elementos (por ejemplo, libros, películas o CD) en un sistema. Estas calificaciones se pueden ver como una representación aproximada del interés del usuario en el dominio correspondiente.

- El sistema compara las calificaciones de este usuario con las de otros usuarios y encuentra a las personas con gustos “similares”.
- El sistema recomienda elementos que los usuarios similares han calificado pero aún no han sido calificados por este usuario.
- Un problema clave del filtrado colaborativo es combinar y ponderar las preferencias de los usuarios vecinos. A veces, los usuarios pueden calificar inmediatamente los elementos recomendados. Como resultado, el sistema obtiene una representación cada vez más precisa de las preferencias del usuario a lo largo del tiempo.

En cualquier aplicación donde se requiere recuperar objetos similares a una consulta, como las descritas, y otras como la compresión de datos, computación geométrica, correción ortográfica, etc, en cualquiera se puede realizar la tarea de forma eficiente utilizando un índice. La técnica que se debe implementar depende principalmente de la cantidad de memoria principal disponible y el tamaño del conjunto.

Capítulo 4. Propuesta

Supongamos que se requiere realizar una búsqueda eficiente del vecino más cercano en una base de datos muy grande, tanto que cualquier índice para memoria principal no cabe en la memoria principal disponible. Una posible solución es implementar en memoria secundaria una técnica para memoria principal. El problema es que un acceso a un registro a memoria principal es del alrededor de 1000 veces más rápido comparado con el acceso en disco de estado solido. Índices como el $hnsw$ que no pueden inferir un patrón acceso registros en memoria en una búsqueda. En consecuencia se espera que el tiempo de consulta sea alrededor de mil veces más lento implementado en memoria secundaria. En este trabajo proponemos una estructura que puede limitar el número de accesos a memoria para poder solucionar el problema del vecino más cercano eficientemente en memoria secundaria.

La idea principal en esta propuesta es organizar los elementos de un conjunto S en un archivo invertido, para esto primero generamos un subconjunto finito R de elementos a partir de S (también llamados referencias o etiquetas) de un tamaño mucho más pequeño que el tamaño de S, las etiquetas pueden verse como claves o descriptores en la estructura del archivo invertido. A continuación, organizamos los elementos de S en su respectiva lista invertida utilizando el criterio de la referencia más cercana como clave. Con el propósito de acelerar la búsqueda de la referencia más cercana, utilizamos una estructura de índice auxiliar con muy alta precisión para mantener un proceso de alta calidad de construcción. Como este criterio no puede controlar el número de elementos para la lista invertida, forzamos esta propiedad al fijar el número máximo de elementos en una lista, también utilizamos una cola de prioridad en cada lista invertida para mantener solo los elementos más cercanos a una referencia. La desventaja de limitar el tamaño de las listas es no tener la garantía de que todos los elementos en S se encuentren en alguna lista invertida, para superar este problema, repetimos el proceso varias veces hasta que no haya ningún elemento en S fuera del archivo invertido. Experimentalmente observamos que con la presencia de todos los elementos en una sola lista invertida no es suficiente para obtener un alto recall, por lo tanto, dejamos que los elementos aparezcan en varias iteraciones, lo que resulta en suscribir un elemento varias veces en la lista invertida de su referencia más cercana, a esta propiedad de nuestro enfoque la llamamos traslape.

Dado un conjunto P (la base de datos), un cubrimiento del conjunto es una colección de subconjuntos $\mathbb{C} = \{C_1, \ldots, C_\ell\}$ tal que $\bigcup_{i=1}^{\ell} C_i = S$. Si además $C_i \cap C_j = \emptyset, j \neq i$, entonces la cubierta es una partición. Si $|C_i| \leq m$ con m un entero constante, se dice que el conjunto que

cubre $\mathbb{C}$ es de tamaño acotado.

Los cubrimientos de conjuntos, en particular las particiones, han sido utilizados previamente en el diseño de índices métricos. Por ejemplo, en Chávez y Navarro (2000) la lista de clusters para la indexación exacta, el cubrimiento del conjunto es acotado. En MI-File Amato *et al.* (2014a) se realizó un cubrimiento no acotado para la indexación aproximada, los elementos dentro de cada subconjunto están agrupados en listas invertidas del archivo invertido. Además de la regla de descomposición, debe haber una regla para activar las partes relevantes en una búsqueda.

En la lista de clusters el cubrimiento es una lista ligada de $4-$tuplas $(R_i, r_i, C_i, i+1)\ i = 1, ..., \ell$, donde $R_i \in P$, $C_i \subset P$, $r_i = \max_j d(R_i, c_{ij}), c_{ij} \in C_i$ y $i+1$ es el identificador para el siguiente subconjunto en la lista. Los subconjuntos del cubrimiento son bolas con centro R_i y radio r_i. La regla de activación requiere un radio de consulta, que hace que la consulta también sea una bola. Los conjuntos activados son aquellos que se intersectan con la bola de consulta. Las desventajas en la lista de clústers son el tiempo de construcción y el costo de encontrar los conjuntos a activar (candidatos), que en el peor de los casos es lineal con respecto al tamaño de la base de datos. Según los autores, esta estructura de datos es el índice exacto que hace el menor número de cálculos de distancia en la práctica.

Para MI-File el cubrimiento es una partición. Cada subconjunto se representa por la pareja (R_i, C_i) donde R_i es un centro y C_i sus elementos asociados, tal que $d(R_i, c_{ij}) \leq d(R_\ell, c_{ij}) \forall \ell \neq i$. La regla de activación para MI-File son las k referencias R_i a la consulta. Una desventaja de esta técnica es no tener control sobre el tamaño de los subconjuntos.

4..1. Cubrimiento acotado del conjunto

El objetivo de nuestra propuesta es construir un cubrimiento con lo mejor de las dos técnicas anteriores. La construcción es explicada con detalles técnicos a continuación. El cubrimiento no será una partición, es decir, los subconjuntos pueden tener intersección, aunque uno de los parámetros considerados en la construcción es el grado de traslape. Los subconjuntos son ternas (R_i, C_i, r_i), donde R_i será el centroide asociado, C_i los elementos asociados y r_i el radio del cubrimiento. Cada objeto de la base de datos aparecerá como máximo en m subconjuntos. La regla de activación seleccionará las k referencias más cercanas a la consulta.

4..1.1. Construcción

El cubrimiento de nuestra propuesta es similar al de la lista de clústers. Cada centroide estará asociado con todos los objetos dentro de un cierto radio. Si el tamaño de los subconjuntos está limitado por una constante, entonces el número de centroides requeridos para el cubrimiento es lineal. Un problema en la construcción es calcular la distancia del centro a cada objeto, esto requiere un tiempo casi cuadrático.

Hay que tener en cuenta que cada objeto en la base de datos debe estar asociado a la referencia o centro más cercano. Una forma de resolver el problema es indexar las referencias y luego consultar el índice con cada objeto en la base de datos. Un índice aproximado competitivo tendrá un comportamiento sublineal en tiempo de consulta. Por lo tanto, todo el proceso será sublineal. Sin embargo, esto construirá subconjuntos de tamaño ilimitado. Para resolver esto, podemos limitar el tamaño de cada subconjunto, ordenando todos los objetos asociados con respecto a la distancia al centroide R_i y luego tomar solo los ℓ elementos más cercanos a R_i .

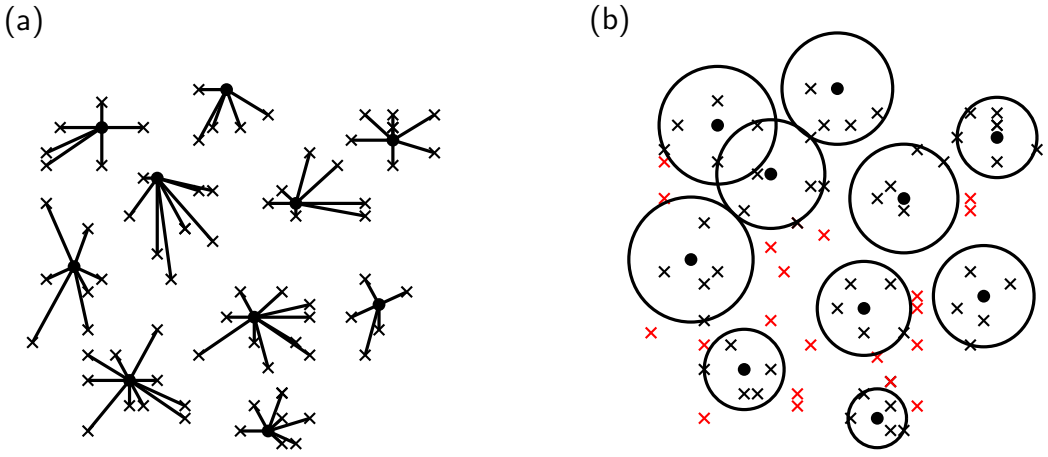


Figura 27. Primeros pasos para el cubrimiento del conjunto acotado. a) cada objeto se asocia con su referencia más cercana. b) la referencia limita su cobertura con los ℓ más cercanos.

Observe que no se puede garantizar que todos los subconjuntos tengan el mismo tamaño. Esto no debería ser un problema porque en el momento de la consulta, nuestro objetivo es tener solo un tamaño acotado en los subconjuntos del cubrimiento. Esto se ilustra en la Figura 27

4..1.2. Traslape

El procedimiento anterior causa otro problema. Muchos objetos de la base de datos pueden no estar cubiertos por ninguna de las referencias. Para resolver esto, usamos otro lote de referencias y repetimos el proceso con aquellos objetos que aún no están asociados a una referencia y aquellos que se encuentran en menos de t subconjuntos. Este proceso continúa hasta que todos los objetos estén asociados a al menos una referencia. Entonces un objeto se puede encontrar en 1 y hasta en t subconjuntos. Al parámetro t lo llamamos traslape (ver Figura 28).

La idea del traslape es que si un punto en la base de datos es cercano a un objeto de consulta es probable que pertenezcan a los mismos subconjuntos. Esta propiedad permite mejorar el tiempo de consulta, y a realizar un posfiltrado de los candidatos sin tener que evaluar distancias, además de no tener que compensar en *recall*.

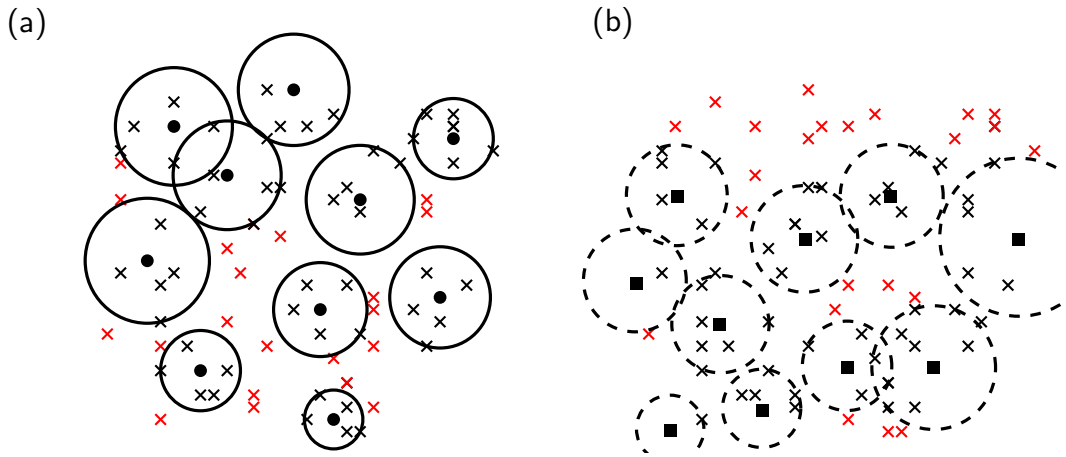


Figura 28. Dos iteraciones del cubrimiento, con dos lotes distintos de referencias.

4..1.3. Estructura

La estructura que proponemos agrupa los elementos de los subconjuntos del cubrimiento en las lista invertidas de un archivo invertido. Luego la clave de acceso al índice es un identificador único de cada centroide en el cubrimiento, para esto, todos los subconjuntos C_i generados en diferentes iteraciones de la construcción son tomados como un solo cubrimiento del conjunto.

En una consulta, para no buscar secuencialmente por las k referencias más cercanas al objeto de consulta, construimos un índice probabilístico de las referencias del cubrimiento en particular utilizamos *HNSW*, con esto se mejora en tiempo sacrificando muy poco en *recall*.

La estructura típica de lista ligada de una lista invertida vinculada a una clave en el archivo invertido, se remplazó por una cola de prioridad. Con esto se puede mantener eficientemente los elementos más cercanos a un centroide cuando se construye el índice (ver Figura 29).

Algoritmo 10 Construcción de cubrimiento acotado

Entrada: S conjunto de objetos, t_l tamaño de lista, t traslape, n_r referencias por iteración
Salida: I índice invertido, $HNSW$ índice auxiliar para consultas

Función BUILDCUBRIMIENTO(S,t_l,t,n_r)
I el índice invertido
R el conjunto de referencias
$T[|S|]$ arreglo temporal para mantener el traslape
Se inicializa T en ceros
Mientras Algún objeto en S está fuera del cubrimiento **Hacer**
Se asigna R_I muestreando aleatoriomente n_r objetos en S
Para $o \in S$ **Hacer**
Se asigna r_n como la referencia más cercana en R_I a o
Si ($|I[r_n]| \leq t_l$ o $d(r_n, o) \leq d(r_n, I[r_n][t_l])$) y $T[o] < t$ **Entonces**
Se inserta o en la cola de prioridad $I[r_n]$
$T[o] = T[o] + 1$
Fin Si
Fin Para
$R \cup R_I$
Fin Mientras
Se construye el índice de consulta $HNSW$ con R
Regresa I,$HNSW$
Fin Función

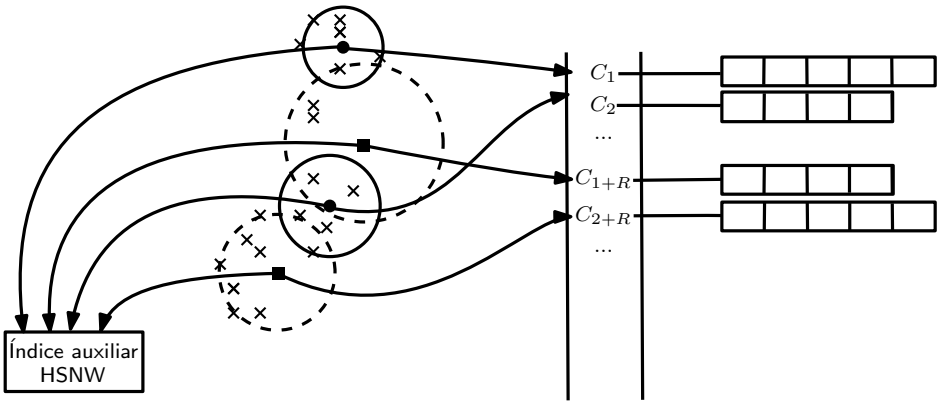


Figura 29. Construcción de un cubrimiento acotado del conjunto usando 2 referencias por iteración

4..1.4. Consultas

Una vez que se construye la estructura de datos, las consultas son bastante simples. Todas las referencias se indexan con un índice probabilístico competitivo. Como el conjunto de referencias es pequeño, el índice probabilístico se implementa en la memoria principal (RAM). La búsqueda de las k referencias más cercanas se puede hacer en tiempo casi constante. Luego de activar ciertas listas, el conjunto candidato se forma de la unión de estas listas. Un problema es que el conjunto candidato puede ser grande y por lo tanto es necesario tener que evaluar muchas distancias en una consulta. Para evitar esto, utilizando propiedades de la lista de candidatos realizamos un proceso de posfiltrado que reduce a muy pocos candidatos finales. El proceso completo se observa en la Figura 30.

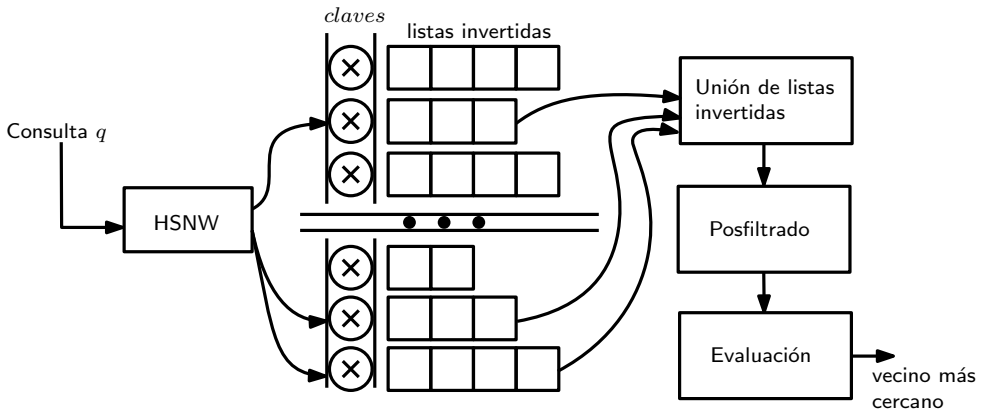


Figura 30. Proceso de consulta en el cubrimiento acotado del conjunto

Algoritmo 11 Búsqueda en el cubrimiento acotado

Entrada: I índice invertido, $HSNW$ índice auxiliar, q consulta, q_k número de referencias más cercanas,$q_{k'}$ top posfiltrado
Salida: El vecino más cercano a q
Función BUSQUEDANNCUBRIMIENTO(I,$HSNW$,q,q_k,$q_{k'}$)
W el conjunto de candidatos
Se asigna R_q con las q_k referencias más cercanas con $HSNW$.
Para $r \in R_q$ **Hacer**
$W \cup I[r]$
Fin Para
$W^* = posfiltrado(W, q_{k'})$
Evaluación(W^*)
Regresa $W^*[0]$
Fin Función

4..1.5. Posfiltrado

Una vez que se tiene el conjunto de candidatos W, una manera de contestar a la consulta q es evaluar las distancias $d(q, w_i)$ para $i = 1, ..., |W|$. Sin embargo, observamos experimentalmente que el conjunto candidato W puede tener un tamaño considerablemente grande. Por lo cual, es necesario reducir el número de candidatos y construir un nuevo conjunto W^* a partir de W, a este proceso se le conoce como posfiltrado.

En esta etapa del proceso proponemos un procedimiento simple basado en la construcción de un histograma de frecuencias. La idea es que si la consulta y un objeto en la base de datos son cercanos muy probablemente se encuentren en los mismos subconjuntos de cubrimiento o al menos en algunos. Hay que tener en cuenta que el número máximo de traslape es t, por lo tanto, esta es la frecuencia máxima en el histograma.

En este proceso los elementos del histograma pueden ser vistos como una lista de parejas (w, fr) donde w es un elemento candidato en W y fr la frecuencia con que aparece en la lista. Esta lista la ordenamos con respecto a la fr y tomamos en cuenta solo los primeros k' (parámetro de consulta) elementos, con los que se forma el conjunto W^*, donde $|W^*| = k'$ y se espera sea mucho más pequeño que W.

4..1.6. Memoria secundaria

Una de las ventajas de nuestra estructura es que el método de activación tiene un número constante de accesos al índice invertido, es decir, en una búsqueda se recuperan las q_k listas vinculadas a las q_k referencias más cercanas. Esto es importante porque en técnicas para memoria primaria el patrón de acceso no se puede predecir y por lo tanto, para realizar una búsqueda eficiente, toda la estructura se tiene que mantener en memoria primaria.

Además la estructura de índice invertido ha sido implementada en la libreria Apache Hadoop `www.https://hadoop.apache.org/`, la cual contiene métodos óptimos para la construcción de índices invertidos. En combinación con la libreria Apache Spark `www.https://spark.apache.org/` se pueden generar soluciones que construyen índices invertidos de forma paralela y distribuida.

4..2. Experimentos

4..2.1. Base de datos y parámetros

Los experimentos se realizaron utilizando una base de datos pública *TexMex bigann*. Consiste en un conjunto de mil millones de vectores sift extraídos aproximadamente de un millón de imágenes. También tiene 10000 consultas seleccionadas al azar, y los 1000 vecinos más cercanos para cada consulta. Las consultas y sus vecinos se proporcionan sobre todo el conjunto, pero también para subconjuntos de diferentes tamaños. Estos subconjuntos son los primeros N vectores del conjunto original, en los experimentos utilizamos un subconjunto de $1M$. La base de datos se puede descargar en `ftp://ftp.irisa.fr/local/texmex/cofrpus/bigann_base.bvecs.gz`

Para evaluar la calidad de los resultados aproximados, utilizamos un conjuntos de consultas Q_s de 500 elementos de la base de datos, y también el vecino más cercano de cada consulta. Específicamente, para cada consulta, seleccionamos un conjunto candidato W usando la estructura de índice. Para esto obtenemos q_k referencias más cercanas a la consulta, lo que da como resultado conjuntos candidatos acotados por $|W| = q_k * listSize$, se asignó q_k con 1, 10, 50 y 100.

El primer experimento que realizamos tiene como objetivo observar el *recall* en el conjunto candidato W. El parámetro de tamaño de lista se fijo en 200 y las referencias por iteración en 10000. También se asignó el parámetro del traslape t en 5, 8, 10, 12 y 15.

En el segundo experimento utilizamos el posfiltrado para observar cuantos elementos se pueden descartar del conjunto candidato W. Para esto en el posfiltrado se asignó $q_{k'}$ con los valores 100, 200, 500, 1000, 1500, 2000, 3000 y 5000. Hay que notar que el recall máximo para este experimento es el que se obtiene en el primer expermiento.

Los experimentos se realizaron en una computadora Asus Rog G751GM que cuenta con un cpu *i7* de *2.50* ghz, 8 gigabytes de memoria RAM y un disco de estado solido de 500 gigabytes. Los experimentos se ejecutarón en memoria principal para poder compararlo con técnicas en la literatura.

4..2.2. Resultados

La estructura propuesta tiene como objetivo principal una respuesta eficiente, por lo tanto, es importante observar el compromiso que existe entre tamaño del conjunto consulta/tiempo y

el *recall*. Otra característica importante a observar del índice es el crecimiento del tamaño con respecto al *recall*.

El resultado del primer experimento se muestra en detalle en la Tabla 3, se puede observar que a partir de la activación de 100 subconjuntos del cubrimiento para cualquier asignación de parámetros se alcanza un *recall* casi perfecto. La diferencia es que entre mayor traslape se asigne a la estructura se requieren más subconjuntos para el cubrimiento, esto incrementa el tamaño del conjunto de candidatos en una búsqueda.

Tabla 3. Resultados del primer experimento

	traslape=5			traslape=8		
	tamaño de lista=200			tamaño de lista=200		
	tamaño=289.5Mbs			tamaño=461Mbs		
k en búsqueda	referencias	candidatos	recall	referencias	candidatos	recall
1	99772	92.93	18.2	126201	106.58	27.4
10	99772	949.68	71.8	126201	1033.64	79.6
50	99772	4609.12	95.2	126201	5157.39	97.2
100	99772	9083.88	99	126201	10263.4	99.2

	traslape=10			traslape=12		
	tamaño de lista=200			tamaño de lista=200		
	tamaño=476.5Mbs			tamaño=481Mbs		
k en búsqueda	referencias	candidatos	recall	referencias	candidatos	recall
1	144532	117.27	25.4	149797	131.572	27.6
10	144532	1157.29	83	149797	1292.09	85.8
50	144532	5732.12	99.4	149797	6435.69	99.6
100	144532	11334.1	99.6	149797	12827.2	100

	traslape=15		
	tamaño de lista=200		
	tamaño=483Mbs		
k en búsqueda	referencias	candidatos	recall
1	150000	136.68	34.6
10	150000	1372.52	84.6
50	150000	6811.91	99
100	150000	13500.5	99.8

En el segundo experimento se aplicó el posfiltrado propuesto a los mejores resultados de los primeros experimentos. En la Tabla 4 se observa que a partir de un traslape de 10 se puede filtrar el $90\,\%$ de los candidatos sin perder prácticamente nada en *recall*.

Tabla 4. Resultados del segundo experimento

Top k'	t=5	t=8	t=10	t=12	t=15
100	49.8	72.8	78.6	77.4	79.2
200	63.6	82	86	86.4	88.2
500	77.8	89.8	96	94.4	94
1000	87.2	93.2	97.6	97.8	97.8
1500	90.8	95.4	98.4	99	99
2000	92	95.6	98.6	99.2	99.6
3000	93	96.8	98.6	99.2	99.6
5000	96	98	98.6	99.4	99.6

En la Figuras 31 y 32 se muestra en resumen el comportamiento del cubrimiento acotado comparado con técnicas del estado del arte como *hnsw* (Malkov y Yashunin, 2018), *annoy* (Annoy, 2013), *pq* (Jégou *et al.*, 2011) y *kdtree* (Friedman *et al.*, 1976). En los experimentos utilizamos las implementaciones de Aumüller *et al.* (2017). Para annoy se asignó el parámetro de número de árboles con 100, 200 y 400 y el tamaño del conjunto candidato 100, 200, 400, 1000, 2000, 4000, 10000, 20000. Para el hnsw el parámetro de enlaces por nodo m se asignó con los valores 8, 12, 16, 24, 36, 48, 64 y 96, $efConstruccion$ con 500 y el parámetro de búsqueda $efSearch$ con 10, 20, 40, 80, 120, 200, 400, 600 y 800. En la comparación se observa que nuestra propuesta tiene un recall casi perfecto en un tiempo competitivo con la mitad del tamaño requerido por las estructuras contra las que se comparó.

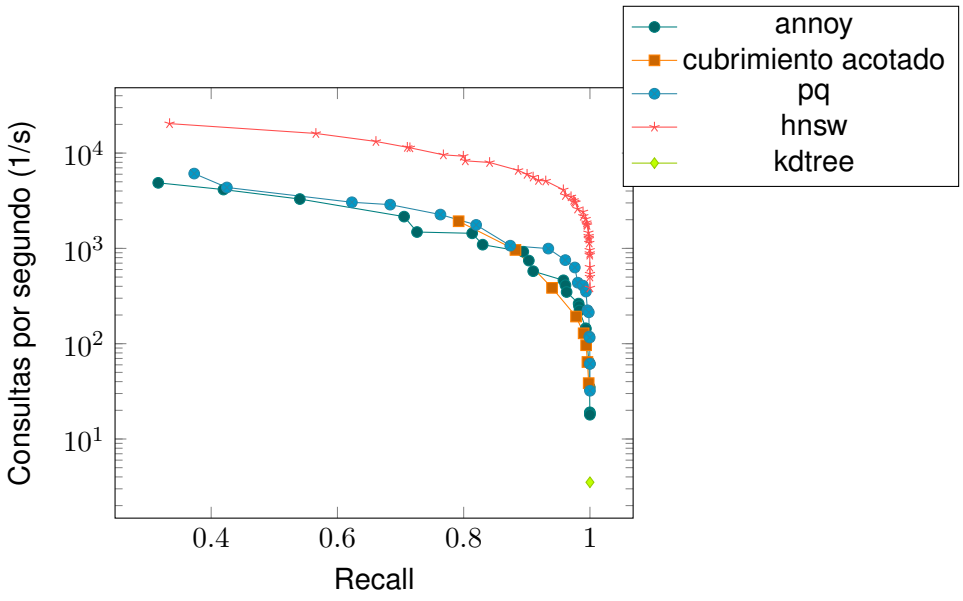


Figura 31. Recall-Consultas por segundo (1/s)

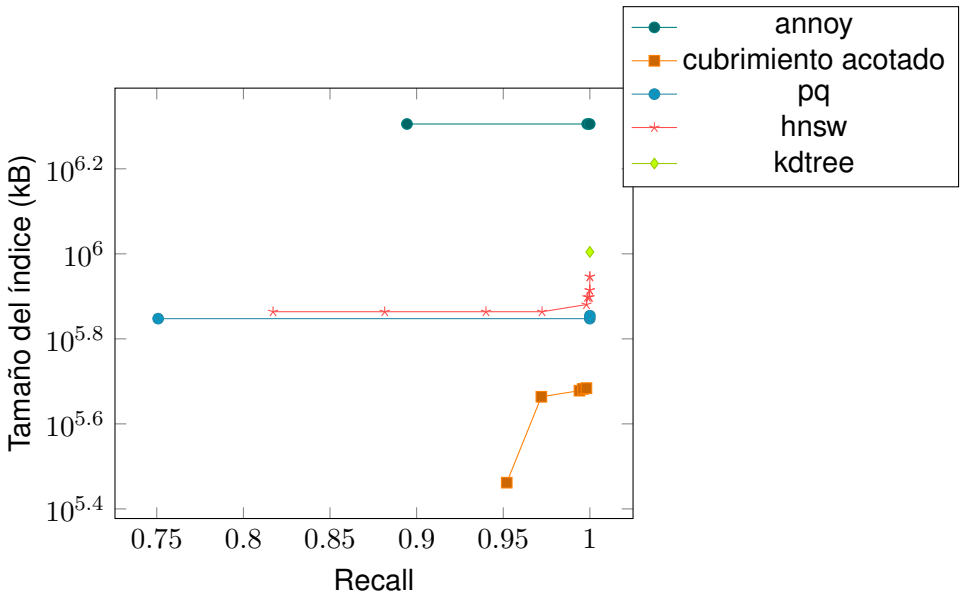


Figura 32. Recall-Tamaño del índice (kB)

En este trabajo presentamos un nuevo tipo de algoritmo para memoria secundaria con pocos parámetros. La idea central es utilizar una descomposición de la base de datos en subconjuntos de tamaño delimitado por una constante. Esto permite la planificación de consultas y la asignación de recursos en el momento de la consulta. En los experimentos realizados se demostró la solidez de nuestra propuesta, nuestro enfoque es competitivo frente a los índices aproximados en el ANN-Benchmark `http://ann-benchmarks.com/`.

4..2.3. Análisis

Los parámetros de construcción son el tamaño de lista t_l, el traslape t y parámetros externos utilizados por el índice auxiliar. Para la búsqueda son el número q_k de listas invertidas que se activan y el tamaño $q_k^{'}$ del subconjunto a considerar en el posfiltrado. Nuestra propuesta tiene pocos parámetros comparado con la técnica MI-File. Además, el traslape puede mejorar el recall sin un sacrificar tiempo en una consulta, por que el tiempo solo depende del número q_k de subconjuntos del cubrimiento que se revisen y el tamaño de lista t_l.

En una prueba se comparó el tiempo de lectura de un disco Ram de 8 Gigabytes contra un disco interno SSD SATA III Samsung 830 de 500 Gigabytes y un disco HDD Western Digital Black de 2 Terabytes. En una computadora i7 3.5Ghz con 16 Gigabytes de RAM total. Las pruebas se hicieron utilizando el software CrystalDiskMark `www.CrystalDiskMark.com` instalado en windows

8.1. Las prueba fue leer muchos archivos de diferentes tamaños y observar el número de Megabytes que se pueden leer en un segundo en las diferentes tecnologías. Se probo con lectura secuencial y aleatoria para tamaño de bloque de 512 bytes y 4 Kilobytes.

El resultado de la prueba se observa en la Figura 33, con un pico de 6.9 Gigabytes por segundo en lectura secuencial con memoria RAM es 1,701 en porcentaje más rápido que el disco Samsung 830 SSD y el mismo comportamiento se observa para lectura aleatoria. Con este ejemplo se puede dar una idea de que un índice para memoria principal que no toma en cuenta el número de lecturas a registros en memoria puede ser eficiente, pero no en memoria secundaria.

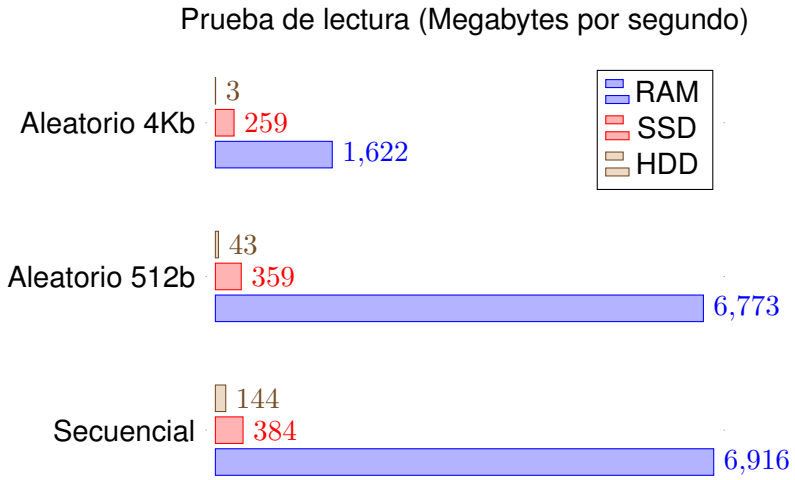


Figura 33. Comparación de tiempo de lectura de RAM. SSD y HDD.

En una búsqueda el número de accesos a disco es limitado por el parámetro q_k y el número total de sectores del disco que se leen por acceso esta acotado por t_l, es decir, el acceso a registros en disco es constante. Si se quisiera mejorar el tiempo se podría tener tantos q_k discos y entonces reducir el tiempo a un acceso a disco.

Comparando nuestra propuesta con MI-File, la técnica considerada el estado del arte en memoria secundaria para espacios métricos generales, se observa que:

- Un mejor control del acceso a memoria que MI-File. En la propuesta limitamos el tamaño de las listas invertidas. MI-File no limita el número duplas que se recuperan de cada lista invertida.
- Tiene menor tamaño que MI-File. Las listas invertidas en el cubrimiento acotado son listas de números. En MI-File son duplas, por que utilizan información adicional para posfiltrar los

candidatos.

- El recall es mejor que el reportado por MI-File. El mejor recall obtenido en nuestra propuesta es $99\,\%$. Mi-File reportó $90\,\%$.

Esta técnica fue diseñada para la ejecución en memoria secundaria pero si se ejecuta en memoria principal es competitiva en tiempo de consulta. Con esto se puede deducir que realizando experimentos en la escala de mil millones, esta técnica podría se considera el estado del arte en memoria secundaria para espacios métricos generales.

Capítulo 5. Identificación del parlante

Específicamente en esta tesis se trabajó en el área de identificación por biométricos en el problema de identificación del parlante. Este problema se define de la siguiente manera, dada la señal de voz de un parlante desconocido y una colección de señales etiquetadas de parlantes conocidos, el objetivo es encontrar a quien pertenece la locución.

Los casos de uso de identificación de parlantes son numerosos. Considérese el siguiente caso: se intercepta una llamada de amenaza terrorista y existe una necesidad crítica de saber quién realizó la amenaza entre varias células criminales identificadas. Reducir el número de candidatos de miles, a una breve lista de docenas de posibles delincuentes es esencial para acelerar la prevención de un ataque. Otra aplicación para la identificación del locutor es la búsqueda de contenido del parlante en grandes bases de datos web como YouTube.

La gran diversidad de las fuentes de la variabilidad en la grabación de voz hacen que el modelado y la identificación de parlantes sea un verdadero desafío. La variabilidad intrínseca debida a factores como la edad, la salud y el estado de ánimo complica aún más el problema. La reducción del efecto de esta variabilidad y deformaciones en la precisión de la identificación requiere varias herramientas matemáticas. El ruido aditivo y las variaciones en el canal de transmisión han sido de particular interés recientemente.

La mayoría de los sistemas de identificación de parlantes tienen tres módulos principales, ver la Figura 34. El módulo de extracción de características recibe pequeños segmentos de voz con duraciones entre 30 y 250 milisegundos de audio con una traslape del 50 por ciento entre segmentos consecutivos. De estos segmentos, se extraen las caracterśticas que están destinadas a ser únicas para un parlante, las propiedades deseables para tales características son:

- Alta variación entre diferentes parlantes y baja variación entre grabaciones realizados por el mismo parlante.
- Robustez a los intentos de falsificación.
- Robustez al ruido aditivo y variaciones del canal de transmisión, variaciones del mismo parlante, etc.
- Baja complejidad para extracción y comparación.

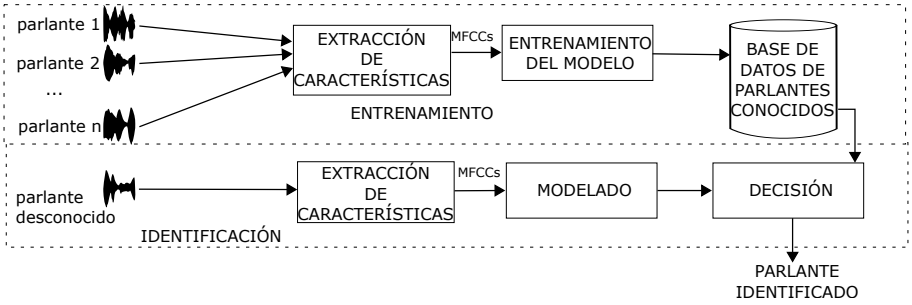


Figura 34. Solución genérica del problema de identificación del parlante.

En términos generales, el modelado es el proceso de encontrar la mejor manera de describir a un parlante con fines de identificación. Este módulo debe proporcionar los medios para comparar parlantes. Un modelo es robusto cuando no se ve afectado por distorsiones. Idealmente, si las características son lo suficientemente robustas para evitar variaciones internas de un parlante y permiten la mejor discriminación entre parlantes, entonces un modelo simple sería suficiente.

Finalmente, en el módulo de decisión, se aplica un conjunto de estrategias entre los modelos de parlantes conocidos y el modelo de un parlante desconocido. Es en esta etapa donde se realiza la comparación entre modelos. Al final, la identidad de la mejor coincidencia será la respuesta.

El proceso que utilizamos es el siguiente: representamos segmentos de voz en vectores de alta dimensión y utilizamos el enfoque de la bolsa de palabras junto con restricciones de tiempo como la única medida de similitud entre las voces de personas. El problema pasa a contar los segmentos de voz compartidos entre la consulta y las muestras de voz previamente almacenadas de los parlantes candidatos. El enfoque es de alguna manera similar al discutido en Liu *et al.* (2016).

Utilizamos el concepto de entropía espectral como la característica principal de la señal de voz. Según Claude Shannon Mok, Mandy Man (2009), la entropía mide la cantidad de información contenida en un mensaje. Shannon propuso la fórmula de Boltzmann (13) para medir la compresibilidad, más información implica menos compresibilidad. Formalmente, para una variable aleatoria X con posibles valores $x_1, ..., x_M$ se define la entropía espectral por:

$$H(X) = -\sum_{i=1}^{M} P\left[X = x_i\right] log\left(P\left[X = x_i\right]\right) \tag{13}$$

Dado que los humanos identificamos personas bastante bien, tiene sentido intentar reproducir

la forma en que funciona el oído humano, al menos en la fase de extracción de características. Una banda crítica es el rango de frecuencias donde el oído no puede distinguir un tono de otro, en Barks cada banda crítica tiene un ancho de Bark, pero en Hertz, la escala de Bark es logarítmica por encima de 500 Hz. La Ecuación 14 se usa para convertir Hertz a Barks,

$$Barks = 13arctan\left(\frac{0.75f}{1000}\right) + 3.5arctan\left(\left(\frac{f}{7500}\right)^2\right) \tag{14}$$

donde f es la frecuencia en Hertz.

La señal de voz tiene componentes de frecuencia en el rango de 80 Hertz a 4000 Hertz. Seleccionamos solo las bandas críticas contenidas en ese rango. La primera banda crítica corresponde al intervalo $[20 - 100]$ Hertz, la banda 18 al intervalo $[3700 - 4400]$ Hertz, las bandas 19 a 25 cubren el resto del rango audible hasta 20000 Hertz pero están fuera del rango dinámico del habla y no son de interés para propósitos de identificación. Por lo tanto, calculamos la entropía espectral para cada banda crítica entre 2 y 17 de acuerdo con la escala de Bark, estas bandas cubren el rango entre 100 y 3700 Hertz. Obtenemos un vector de características de 16 dimensiones por cada 11 milisegundos de audio ya que nuestros segmentos de 32 ms se traslapan en 21 ms. Nuestro modelo de parlantes es solo el conjunto de estos vectores de características, se puede interpretar como una nube de puntos de alta dimensión. Sabemos que esta nube de puntos de vectores de entropía espectral es robusta al ruido aditivo, a las variaciones del canal de transmisión y a las mismas variaciones de los locutores.

La identificación de un parlante desconocido modelado por un conjunto de vectores de características de entropía se puede hacer con una búsqueda secuencial en una colección de modelos de parlantes conocidos, este enfoque tiene un tiempo de ejecución lineal con respecto al tamaño de la colección. Este tiempo de búsqueda se mejora mediante el uso de índices. Reducimos la clasificación a recuperar los K vecinos más cercanos de cada vector de características de entropía de un modelo del parlante consulta entre todos los vectores de características de entropía de todos los modelos de parlantes conocidos. Luego los agrupamos e identificamos los hablantes C con la mayor probabilidad de ser el parlante de la consulta (ver Figura 36).

5..1. Trabajo relacionado

En la fase de extracción de características, la señal se simplifica eliminando frecuencias no audibles para humanos. Debido al envejecimiento, los problemas de salud, los cambios de canal o incluso los cambios conscientes o inconscientes en la forma en que el hablante se expresa (por ejemplo, susurros, gritos, cambios de humor, etc.) resultan en variaciones importantes de la voz producidas por el mismo parlante. Este hecho puede afectar la precisión de un sistema de identificación de parlantes si las características extraídas de la señal no son robustas. Los coeficientes cepstrales en la frecuencia mel (MFCC) Davis y Mermelstein (1980) es la característica más común utilizada en los sistemas de reconocimiento de parlantes, la idea es en este trabajo es la representación de la evolvente del espectro potencia de la señal. En Kinnunen y Li (2010) se presenta una revisión de características utilizadas para el reconocimiento de los parlantes.

Una vez mapeado a vectores en el espacio de características, se debe modelar la voz. Para ello se utilizan con frecuencia modelos generativos estocásticos, como los modelos de mezclas gaussianas (GMM). En Reynolds *et al.* (2000) se construye un modelo independiente para cada hablante, el modelo se determina a partir de una mezcla gaussiana (UBM) independiente del parlante adaptando sus parámetros con un método de Máximo a posteriori (MAP), al modelo generado le llaman GMM-UBM. Sus experimentos muestran que usando únicamente los valores de las medianas, los resultados son tan efectivos como usar las mezclas gaussianas. A un solo vector cuyos componentes son esos valores lo llaman *supervector*. Se han propuesto varias técnicas efectivas para trabajar en el espacio de *supervectores*, la mayoría de ellas basadas en el análisis de factores (FA) que tiene el objetivo de expresar la variabilidad observada en altas dimensiones de manera compacta. *Eigenvoices* y *eigenchannels* Kenny *et al.* (2003) fueron los primeros modelos que usaron FA y se convirtieron en el Joint Factor Analysis (JFA) Kenny (2005) donde el espacio de voz de los altavoces se divide en dos subespacios, estos subespacios modelan independientemente los factores asociados con la variación de la sesión y los relacionados con las mismas variaciones del parlante.

El modelo JFA evolucionó al Modelo de Variabilidad Total (TVM) Dehak *et al.* (2011), donde todas las fuentes de variación se modelan en un solo espacio de baja dimensión. En el enfoque de TVM, el vector de baja dimensión con los factores importantes en un segmento de audio se conoce como *i-vector* y se consideró el estado del arte.

Recientemente los sistemas de reconocimiento de parlantes reemplazaron el modelo GMM-

UBM por Deep Neural Networks (DNN). Las DNN están entrenadas para modelar cada parlante con el objetivo de obtener estadísticas suficientes para la extracción del i-vector Snyder *et al.* (2016). El i-vector se usa junto con un análisis discriminante lineal probabilístico (PLDA) para obtener la similitud entre dos parlantes Greenberg *et al.* (2014). Una revisión sobre técnicas de modelado está disponible en Hansen y Hasan (2015).

5..2. Propuesta

Nuestro enfoque consta de tres módulos. *(i)* extracción de vectores de características de entropía; *(ii)* búsqueda de los vecinos *K* más cercanos; y textit(iii) decisión final basada en la frecuencia de los parlantes en la respuesta. En la Figura 35 se muestra el enfoque propuesto, cada módulo se describe a continuación:

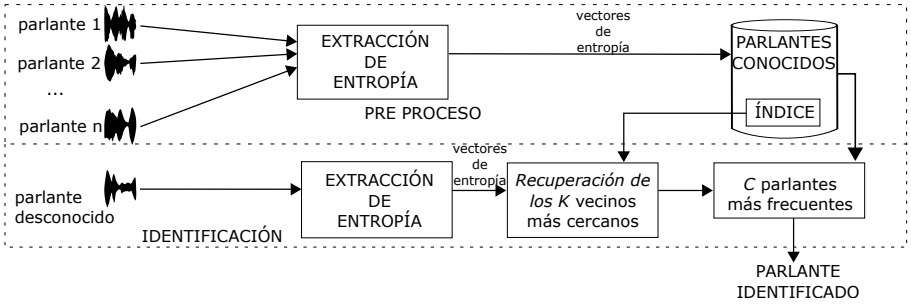


Figura 35. Enfoque propuesto para la identificación del parlante

5..2.1. Cálculo de la entropía

Necesitamos extraer de la señal de audio características robustas que nos permitan identificar un parlante. En nuestra propuesta, calculamos la entropía espectral para cada una de las 16 bandas críticas de acuerdo con la escala Bark y para cada segmento de la señal de voz. Esta secuencia de vectores de características de entropía pueden llamarse espectros de entropía. Ahora explicaremos en detalle el proceso para determinar la secuencia de vectores de características de entropía:

1. Preprocesamiento: Las muestras de la señal de audio se normalizan primero para estar en el rango de -1 a 1. Dado que el contenido de energía en la señal de la voz no se distribuye uniformemente sino que se concentra en las frecuencias bajas, aplicamos el filtro pre-énfasis

de acuerdo con la ecuación de diferencia (15), que actúa como un filtro de pasa altas de primer orden, esto ayuda a lograr un equilibrio entre las frecuencias bajas y altas.

$$y(n) = x(n) - 0.95x(n-1) \tag{15}$$

Tomamos cuadros cortos de 32 ms para una frecuencia de muestreo de 8000 Hz, lo que significa que el tamaño de segmento N equivale a 256 muestras. Los segmentos se traslapan $66\,\%$ para que se determine un vector de característica de entropía cada 10.66 ms. A cada segmento, aplicamos la ventana de Hamming que se define por la ecuación (16).

$$hamming(n) = 0.54 + 0.46cos(2\pi n/N) \tag{16}$$

2. Cálculo de entropía: extraemos un vector de característica de entropía para cada segmento de audio de la siguiente manera: Ponemos a cero el segmento de 256 muestras, por lo que la longitud de la señal resultante es 512, luego aplicamos la transformada rápida de Fourier y agrupamos los coeficientes de Fourier en bandas críticas de acuerdo con la escala Bark. Para cada banda calculamos la entropía usando (13) construyendo un vector de característica de entropía en $\mathbb{R}^{16}$. Para cada parlante, almacenamos un conjunto de vectores de características de entropía junto con la etiqueta que identifica al parlante de esta manera:

$$\{s_i, e_{j,1}, e_{j,2}, ..., e_{j,16} | 1 < j < n_i\} \tag{17}$$

donde s_i es el identificador para el i-ésimo parlante que es único en el conjunto de parlantes y n_i es el número de vectores de características de entropía para el i-ésimo parlante.

El modelo para un parlante consulta es representado como

$$\{e_{j,1}, e_{j,2}, ..., e_{j,16} | 1 < j < n_i\} \tag{18}$$

5..2.2. Búsqueda de los k vecinos más cercanos

Del conjunto de modelos de parlantes conocidos, construimos un índice para el espacio métrico donde los objetos son los vectores de características de entropía y la distancia Euclidiana se usa para medir qué tan diferente es un vector de otro. La idea básica es considerar cada vector $e_j \in \mathbb{R}^{16}$ del conjunto de vectores que pertenecen al modelo de un parlante como un objeto independiente para el que estamos interesados en encontrar su K vecinos más cercanos entre todos los vectores de características de todos los modelos del conjunto de parlantes conocidos (ver Figura 36).

5..2.2.1. Balltree

El *Balltree* es un índice para la búqueda de similitud exacto Uhlmann (1991a) Yianilos (1993a), que divide el espacio de búsqueda utilizando un pivote para cada nodo y construye un árbol binario. La construcción del índice comienza con la selección de un objeto arbitrario elegido como la raíz de un árbol. El resto de los objetos en la base de datos se dividen en dos subconjuntos disjuntos. Si la distancia de un objeto al pivote es mayor que la mediana de las distancias al punto pivote, se enruta al subárbol izquierdo. De lo contrario, se asigna al subárbol derecho. Se repite el mismo procedimiento de forma recursiva con los subárboles hasta tener los tamaños de hoja deseados.

5..2.2.2. Balltree aproximado

Para reducir el tiempo necesario de respuesta a una consulta, se puede utilizar una versión aproximada del balltree. Dos parámetros en el balltree pueden controlar la velocidad/recall. El primero es el número máximo de hojas visitadas permitidas en una búsqueda. El otro está relacionado con el podado del árbol durante una búsqueda. Supongamos que p_i sea un pivote del balltree, q una consulta, r el radio de la consulta y R es la mediana de las distancias desde el pivote al resto de los objetos. Como resultado de la desigualdad del triángulo, se puede realizar una poda del árbol de búsqueda, lo que implica descartar muchos objetos de los candidatos cuando $r \leq |R - d(p_i, q)|$. Si esta condición se cumple, solo se visita un subárbol, pero si $r > |R - d(p_i, q)|$ ambos subárboles deben ser revisados.

Relajando la condición de podado de acuerdo con un parámetro $\alpha > 1$ para que la condición

se convierta en $r \leq \alpha|R - d\,(p_i, q)\,|$, el tiempo de búsqueda se reduce a expensas de perder algunos objetos de la respuesta exacta.

5..2.3. Selección de los C parlantes más probables

Luego de recuperar los K vecinos más cercanos de cada vector de características de entropía de un modelo de parlante consulta que es un conjunto de vectores de características de tamaño n tenemos una secuencia de Kn identificadores (etiquetas del parlantes). A partir de esta secuencia, hacemos un histograma de parlantes y seleccionamos los C parlantes más frecuentes (ver Figura 36).

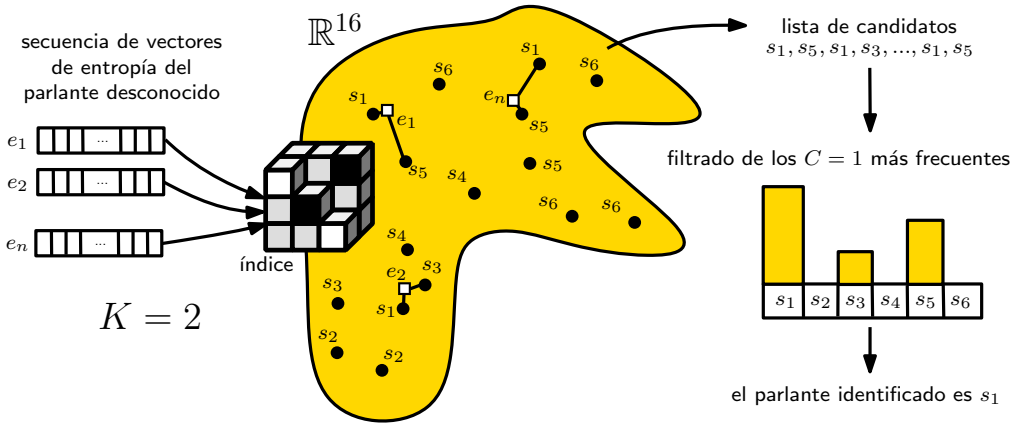


Figura 36. Ejemplo de identificación en la propuesta. Para cada e_i en la secuencia se buscan los $k = 2$ vecinos más cercanos y se construye una lista de candidatos sobre la cual se aplica un posfiltrado y se obtiene el $C = 1$ parlante más frecuente como respuesta.

5..3. Experimentos

El rendimiento de nuestro enfoque se valida utilizando bases de datos públicas, destacando la calidad de la identificación y la velocidad alcanzada.

5..3.1. Base de datos

Utilizamos una lista de videos del canal de Google Tech Talk de Youtube hecha por Schmidt (2014); está compuesta de 1111 videos con 998 parlantes, cada video tiene más de 30 minutos de audio y 74 parlantes aparecen en al menos dos videos distintos. La calidad del audio varía ya que se usaron diferentes equipos de grabación en cada video. Además, el ruido ambiental cambia, la

acústica del lugar varía, así como la distancia al micrófono. La lista de videos se puede descargar desde `http://people.csail.mit.edu/ludwigs/data/youtube_speakers_2013.txt`

Realizamos dos experimentos. En el primer experimento, para cada video, la señal de audio se dividió en un conjunto U de segmentos de 10 segundos. De U seleccionamos al azar 50 elementos y los tomamos como el conjunto de entrenamiento T. Luego en $U - T$ seleccionamos segmentos consecutivos de 10, 20 y 50 segundos para formar los conjuntos de prueba. Observe que, de esta manera, el conjunto de prueba y el conjunto de entrenamiento son disjuntos. Este experimento es equivalente a controlar el entorno de grabación porque tanto la sala como el equipo de grabación serán los mismos para cada parlante.

En el segundo experimento, para cada parlante que aparece en al menos dos videos, uno de ellos fue seleccionado al azar. A partir de esto, construimos el conjunto de entrenamiento de la misma manera que se describe en el primer experimento. Para formar el conjunto de prueba para cada parlante también extrajimos segmentos de 10, 20 y 50 segundos del video no indexado. Este experimento es equivalente a no controlar el entorno de grabación porque el equipo de grabación, la sala y los parámetros del canal son diferentes.

Hay que tener en cuenta que no hay fase de entrenamiento ni aprendizaje de parámetros en ninguna de las dos configuraciones. Solo el audio simple segmentado y convertido en una nube de vectores de entropía espectral. Nuestro proceso está libre de los problemas de sobreajuste atribuidos a otros métodos de aprendizaje automático.

5..3.2. Resultados

Tomamos una consulta como identificada correctamente si el parlante correspondiente aparece en el conjunto C de los parlantes más probables obtenidos después de la lista de identificadores de la concatenación de las n búsquedas de los k vecinos más cercanos (n es el número de vectores de características de entropía de la consulta).

Este trabajo apunta a una identificación eficiente, por lo tanto, un aspecto a observar es cómo se reduce la precisión a medida que aumenta la velocidad. Usamos balltree exacto como método de referencia y observamos las siguientes estadísticas:

Precisión en la identificación. La probabilidad que un parlante fue identificado correctamente.

Tiempo de búsqueda. Se midió el tiempo que tomó a cada algoritmo para recuperar los k vecinos más cercanos a una consulta.

Precisión relativa. Se define como la razón de la precisión del algoritmo aproximado sobre la precisión del algoritmo base.

Tiempo de búsqueda relativo. Se define como la razón del tiempo de búsqueda del algoritmo aproximado sobre el tiempo de búsqueda del algoritmo base.

La Tabla 5 muestra los resultados obtenidos. Se consideraron las mejores precisiones con un tiempo de búsqueda razonable. Los experimentos se realizaron con una CPU de 2.0 GHz. Se muestran los promedios de tiempo y precisión. El algoritmo Annoy logra ser hasta 2000 veces más rápido, con una pequeña reducción en la precisión y el balltree aproximado es hasta 400 veces más rápido sin ninguna reducción en la práctica. En todos los casos, la precisión promedio en la identificación del parlante es de al menos $90\,\%$.

Tabla 5. Resultados de los dos experimentos.

	Experimento					
	Primero			Segundo		
	10s	20s	50s	10s	20s	50s
Tamaño base de datos	118,747,956	118,747,956	118,747,956	110,829,034	110,829,034	110,829,034
Consultas	11,110	11,110	11,110	740	740	740
Balltree precisión	94.1 %	98.7 %	100 %	89.18 %	93.24 %	94 %
Balltree tiempo (ms)	76,200	136,800	271,200	73,150	120,316	250,624
Balltree aprox. precisión rel.	99 %	100 %	100 %	100 %	100 %	100 %
Balltree aprox. tiempo rel.	456x	419x	410x	334x	369x	262x
Annoy precisión	91 %	98.6 %	99 %	85.1 %	90.5 %	91.8 %
Annoy tiempo (ms)	69.6	138	243	33.4	62.2	143
Annoy precisión rel.	96.7 %	99 %	99 %	95.4 %	97 %	97.6 %
Annoy tiempo rel.	1,095x	991x	910x	2,190x	1,934x	1753x
Annoy árboles	5	5	5	5	5	5

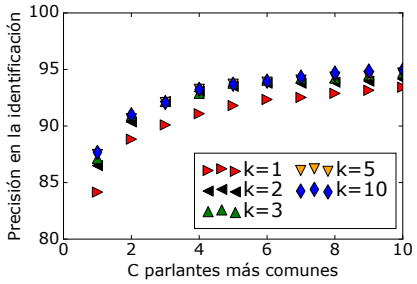
Las Figuras 37 y 38 muestran la precisión promedio en la identificación del parlante para el primer y segundo experimento, respectivamente. Consideramos una búsqueda como exitosa si la

identidad del parlante consulta se encontraba dentro de los C parlantes más comunes, se experimento con valores de $1 \leq C \leq 10$. Para cada vector de entropía extraido del parlante consulta se buscó por sus k vecinos más cercanos con $k = 1, 2, 3, 5, 10$. Se puede observar que para el primer experimento, se puede lograr una precisión casi perfecta con consultas de 20 segundos con cualquier índice con el parámetro $C = 10$ para cualquier valor de k. Para el segundo experimento , se logró una precisión arriba de $90\,\%$ con 20 segundos de consulta con cualquier índice con el parámetro $C = 10$ y $k = 10$.

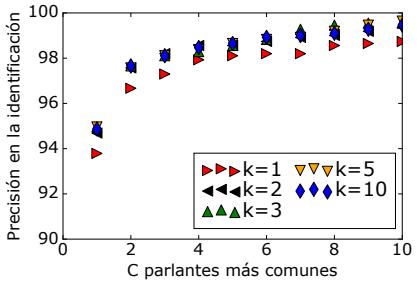
Para el segundo experimento, se puede lograr una precisión cercana a $95\,\%$ para consultas de 20 segundos incluso con la versión aproximada del Balltree. Este resultado superó los métodos en el estado del arte sin una fase de modelado. La identificación es robusta para el mismo parlante, dispositivos de grabación, acústica de sala y variaciones de canal.

Hay que tener en cuenta que nuestros resultados no son directamente comparables con otros sistemas de identificación de parlantes de la literatura ya que no estamos utilizando ningún modelo del parlante.

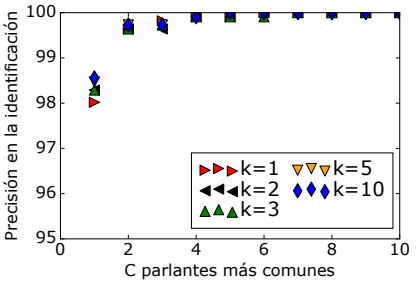
En este trabajo, propusimos un método eficiente para la identificación de parlantes en bases de datos de alta escala. Nuestro modelo del parlante es una nube de puntos formada con vectores de entropía espectral. Cada vector es un segmento de tiempo de la voz digitalizada, y el modelo de decisión consiste en contar el número de bloques coherentes en el tiempo compartidos entre la consulta y los parlantes candidatos. Para acelerar la tarea, utilizamos índices aproximados probabilísticos para buscar los k vecinos más cercanos de los segmentos de consulta. Los resultados de los experimentos realizados con bases de datos públicas muestran que el tiempo puede ser reducido sin mucha pérdida en la precisión.



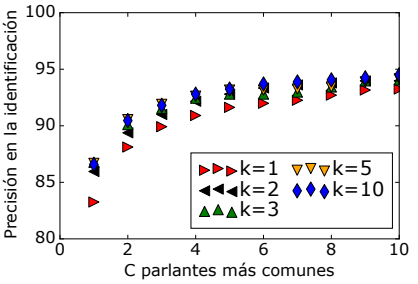
(a) Balltree exacto consulta 10 secs.



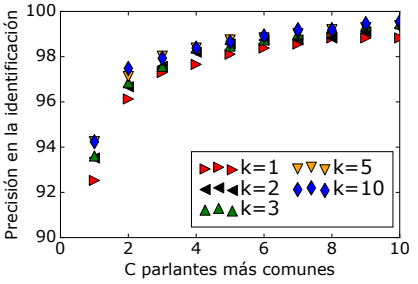
(b) Balltree exacto consulta 20 secs.



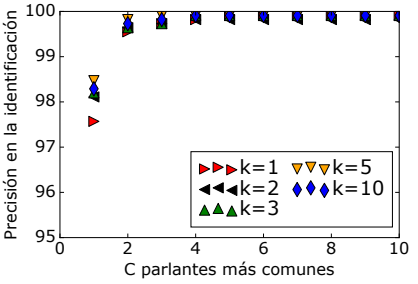
(c) Balltree exacto consulta 50 secs.



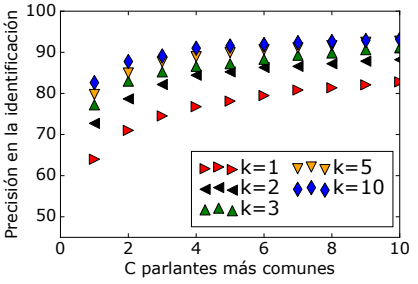
(d) Balltree approx. consulta 10 secs.



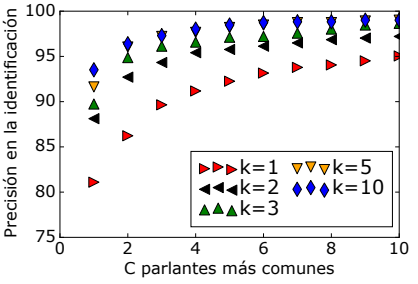
(e) Balltree approx. consulta 20 secs.



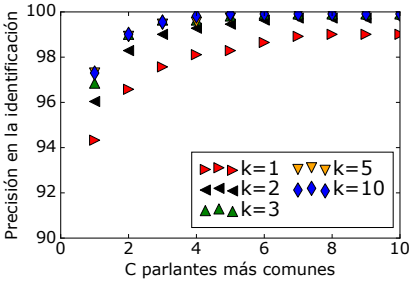
(f) Balltree approx. consulta 50 secs.



(g) Annoy consulta 10 secs.

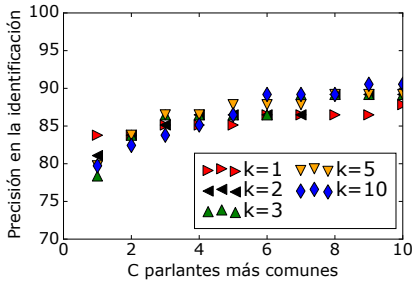


(h) Annoy consulta 20 secs.

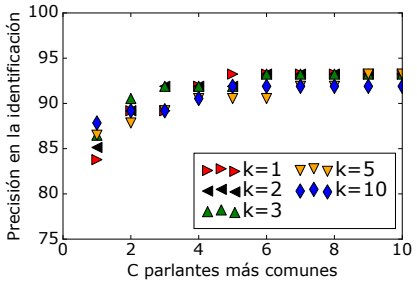


(i) Annoy consulta 50 secs.

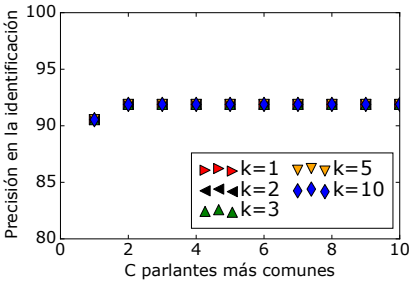
Figura 37. Precisión promedio en la identificación del parlante para diferentes parámetros y tamaños de consulta.



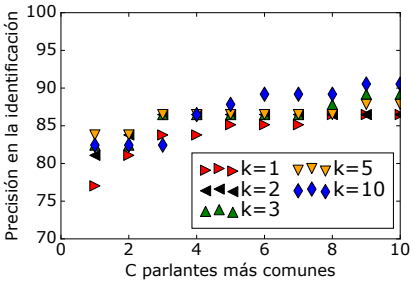
(a) Balltree exacto consulta 10 secs.



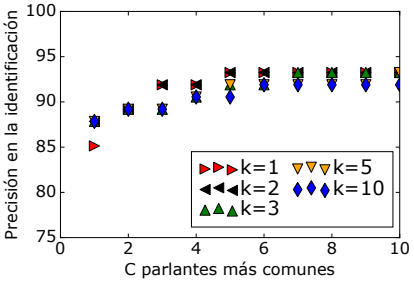
(b) Balltree exacto consulta 20 secs.



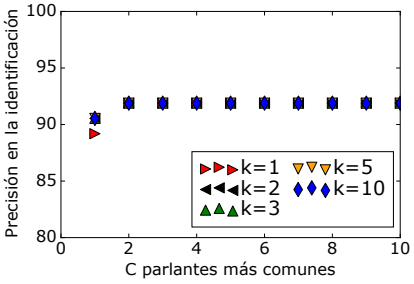
(c) Balltree exacto consulta 50 secs.



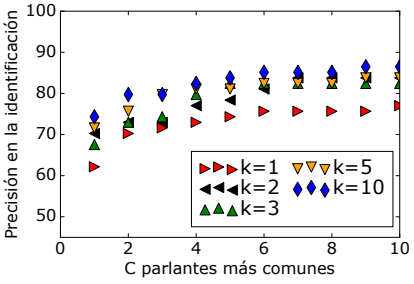
(d) Balltree Approx. consulta 10 secs.



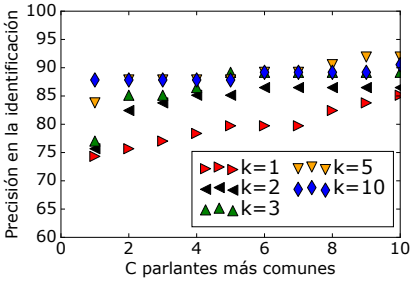
(e) Balltree Approx. consulta 20 secs.



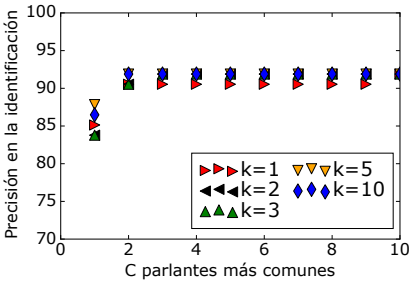
(f) Balltree Approx. consulta 50 secs.



(g) Annoy consulta 10 secs.



(h) Annoy consulta 20 secs.



(i) Annoy consulta 50 secs. query

Figura 38. Precisión promedio en la identificación del parlante para diferentes parámetros y tamaños de consulta.

Capítulo 6. Conclusiones

El objetivo principal de esta tesis fue el abordar el problema de la búsqueda aproximada del vecino más cercano en altas dimensiones, y aportar una solución que resuelva el problema en los casos donde la memoria principal no puede cumplir con el tamaño requerido para montar un índice del estado del arte.

La solución supera a la técnica considerada el estado del arte en mejor control del acceso a memoria, mejora el tamaño del índice, supera el recall reportado y además el tiempo de consulta es competitivo con técnicas para memoria principal.

Como objetivo particular se incluye la propuesta de una solución al problema de identifiación del parlante. Esta tiene como principal ventaja que no requiere de un paso complejo de modelado del parlante ni una fase de aprendizaje para el generador de modelos. Además se probó experimentalmente que utilizando 20 segundos de consulta y un índice aproximado se puede acelerar significativamente el proceso de identificación con un *recall* casi perfecto.

6..1. Aportaciones

La aportación principal es el diseño, implementación y validación experimental, de una estructura llamada cubrimiento acotado. Esta estructura está basada en el índice invertido para agrupar los elementos de una colección y poder limitar el número de accesos a memoria que se requieren para contestar una consulta, esto tiene como resultado la posibilidad de implementar la solución en memoria secundaria, con un requerimiento mínimo en memoria principal. Es el único índice para espacios métricos generales en memoria secundaria que obtiene tiempos comparables con índices para memoria principal.

La solución propuesta para la identificación del parlante se publicó en:

- Camarena-Ibarrola, Antonio, Fernando Luque, and Edgar Chavez. “Speaker identification through spectral entropy analysis.” 2017 IEEE International Autumn Meeting on Power, Electronics and Computing (ROPEC). IEEE, 2017.
- Luque-Suárez, Fernando, Antonio Camarena-Ibarrola, and Edgar Chávez. “Efficient speaker identification using spectral entropy.” Multimedia Tools and Applications 78.12 (2019): 16803-16815.

6..2. Trabajo a futuro

La técnica de cubrimientos acotados que se propuso se puede modificar y utilizarla como solución al problema de clasificación. En este problema dados un conjunto de clases y un objeto consulta el objetivo es responder a que clase pertenece el objeto consulta.

Dado un conjunto de entrenamiento X, donde cada objeto $x \in X$ está etiquetado de la forma(x, z_j) para algún j en el rango de $1 \leq j \leq |Z|$, donde Z es el conjunto de posibles clases. La idea es realizar el cubrimiento acotado con una modificación mínima para obtener un clasificador basado en el de k vecinos más cercanos. No es parámetrico, no esta limitado por el número de clases y necesita un espacio sublineal con respecto al tamaño de X.

La modificación es en el índice invertido de la siguiente manera:

- Cada lista invertida sera de tamaño $|Z|$, porque se agregara un elemento por cada clase $z \in Z$.
- Los elementos en la lista invertida son de la forma (z, fr), donde fr es el número de objetos que se encuentran en el subconjunto del cubrimiento asociado a la lista.
- Cada lista invertida puede ser interpretada como un histograma de las clases (ver Figura 40).

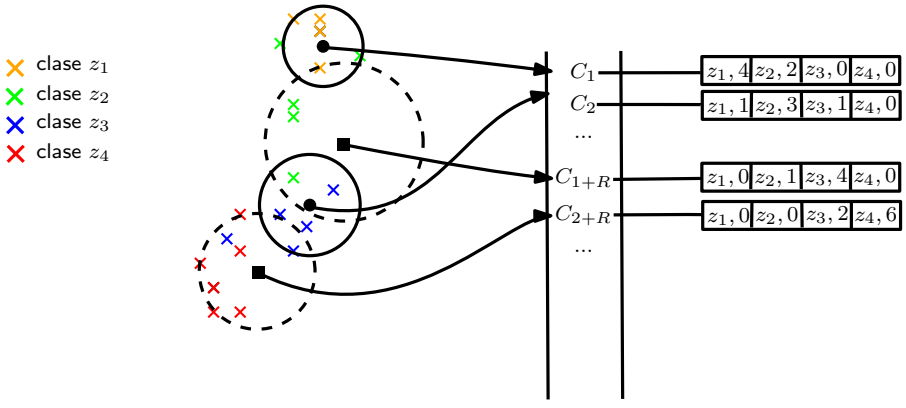


Figura 39. Clasificador con el cubrimiento acotado

Cuando se quiere clasificar un objeto q, se realiza el mismo procedimiento de búsqueda propuesta para el cubrimiento acotado. La única modificación es el proceso de posfiltrado. Para el

clasificador la idea es construir un histograma acumulador H_t a partir de la suma de las frecuencias de los histogramas h_i asociados a las listas invertidas recuperadas.

El histograma H_t contendrá todas las posibles clases $z \in Z$ y acumulara las frecuencias de cada histograma h_i para una clase z_i con $1 \leq i \leq |Z|$. Nuestra hipótesis es que la clase con el pico de frecuencia más alto es la clase a la que pertenece el objeto q (ver Figura).

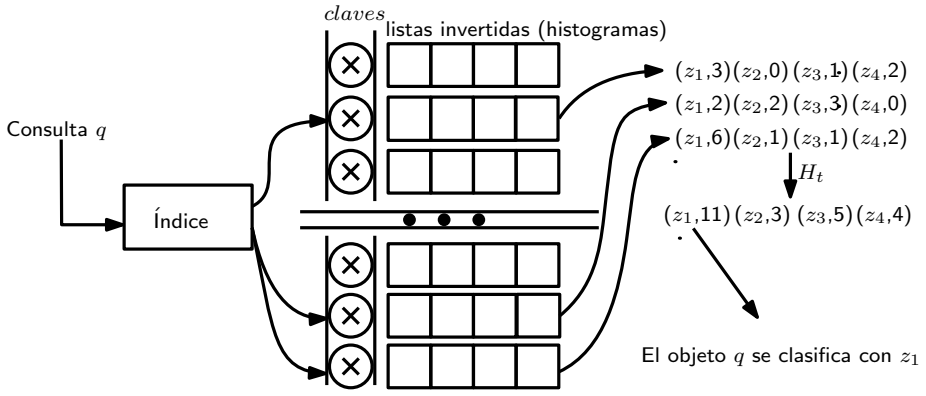


Figura 40. Ejemplo de clasificación con cubrimiento acotado

Literatura citada

Akutsu, T., Kanaya, K., Ohyama, A., y Fujiyama, A. (2003). Point matching under non-uniform distortions. *Discrete Applied Mathematics*, **127**(1 SPEC.): 5–21.

Amato, G., Gennaro, C., y Savino, P. (2014a). MI-File: Using inverted files for scalable approximate similarity search. *Multimedia Tools and Applications*, **71**(3): 1333–1362.

Amato, G., Gennaro, C., y Savino, P. (2014b). MI-File: Using inverted files for scalable approximate similarity search. *Multimedia Tools and Applications*, **71**(3): 1333–1362.

Andoni, A. y Indyk, P. (2006). Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. En: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS*. IEEE, pp. 459–468.

Annoy (2013). Recuperado en 2019 de https://github.com/spotify/annoy.

Arkin, E. M., Chew, L. P., Huttenlocher, D. P., Kedem, K., y Mitchell, J. S. (1990). An efficiently computable metric for comparing polygonal shapes. *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, **13**(3): 129–137.

Auer, T. y Held, M. (1996). RPG-Heuristics for the Generation of Random Polygons. En: *Proc. 8th Canada Conf. Comput. Geom. Ottawa, ...*. Carleton Univertisty Press, pp. 1–12.

Aumüller, M., Bernhardsson, E., y Faithfull, A. (2017). ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **10609 LNCS**: 34–49.

Baeza-Yates, R. (1997). Searching: an algorithmic tour. *Encyclopedia of Computer Science and Technology*, **37**: 331–359.

Baeza-Yates, R., Cunto, W., Manber, U., y Wu, S. (1994). Proximity matching using fixed-queries trees. En: *Annual Symposium on Combinatorial Pattern Matching*. Springer, pp. 198–212.

Bates, M. J. (2011). *Understanding information retrieval systems: Management, types, and standards*. CRC Press. pp. 1–712.

Beigi, H. (2011). *Speaker Recognition*. Springer. pp. 6–19.

Beigi, H. S. M., Maes, S. H., Chaudhari, U. V., y Sorensen, S. (1999). A hierarchical approach to large-scale speaker recognition. *European Conference on Speech Communication and Technology*, **5**: 2203–2206.

Brin, S. (1995). Near neighbor search in large metric spaces. *Proceedings of the 21th International Conference on Very Large Data Bases*, pp. 574–584.

Campbell, W. M., Campbell, J. P., Reynolds, D. A., Singer, E., y Torres-Carrasquillo, P. A. (2006a). Support vector machines for speaker and language recognition. *Computer Speech and Language*, **20**(2-3 SPEC. ISS.): 210–229.

Campbell, W. M., Sturim, D. E., y Reynolds, D. A. (2006b). Support vector machines using GMM supervectors for speaker verification. *IEEE Signal Processing Letters*, **13**(5): 308–311.

Charikar, M. S. (2002a). Similarity estimation techniques from rounding algorithms. En: *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, New York, New York, USA. ACM Press, pp. 380–388.

Charikar, M. S. (2002b). Similarity estimation techniques from rounding algorithms. En: *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*. ACM, pp. 380–388.

Chávez, E. y Navarro, G. (2000). An effective clustering algorithm to index high dimensional metric spaces. En: *Proceedings - 7th International Symposium on String Processing and Information Retrieval, SPIRE 2000*. IEEE, pp. 75–86.

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

Chávez, E., Chávez Cáliz, A. C., y López-López, J. L. (2017). Affine invariants of generalized polygons and matching under affine transformations. *Computational Geometry: Theory and Applications*, **58**: 60–69.

Chavez Gonzalez, E., Figueroa, K., y Navarro, G. (2008). Effective proximity retrieval by ordering permutations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**(9): 1647–1658.

Cheng, F. H. (1996). Point pattern matching algorithm invariant to geometrical transformation and distortion. *Pattern Recognition Letters*, **17**(14): 1429–1435.

Davis, S. B. y Mermelstein, P. (1980). Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, **28**(4): 357–366.

De Abreu Campos, V. y Guimarães Pedronette, D. C. (2016). Effective speaker retrieval and recognition through vector quantization and unsupervised distance learning. *MARMI 2016 - Proceedings of the 2016 ACM 1st International Workshop on Multimedia Analysis and Retrieval for Multimodal Interaction, co-located with ICMR 2016*, pp. 27–32.

Dehak, N. y Shum, S. (2011). Low-dimensional speech representation based on factor analysis and its applications. *Johns Hopkins CLSP Lecture*.

Dehak, N., Dehak, R., Kenny, P., Brummer, N., Ouellet, P., y Dumouchel, P. (2009). Support vector machines versus fast scoring in the low-dimensional total variability space for speaker verification. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, pp. 1559–1562.

Dehak, N., Kenny, P. J., Dehak, R., Dumouchel, P., y Ouellet, P. (2011). Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech and Language Processing*, **19**(4): 788–798.

Díaz, F. J., Sierra, G. H., y De Lara, J. C. (2014). A Gaussian selection method for speaker verification with short utterances. *Computacion y Sistemas*, **18**(2): 345–358.

Friedman, J. H., Bentley, J. L., y Finkel, R. A. (1976). An algorithm for finding best matches in logarithmic time. *ACM Trans. Math. Software*, **3**(SLAC-PUB-1549-REV. 2): 209–226.

Goshtasby, A. A. (2012). *Image registration: Principles, tools and methods.*. Springer Science & Business Media.

Greenberg, C. S., Bansé, D., Doddington, G. R., Garcia-Romero, D., Godfrey, J. J., Kinnunen, T., Martin, A. F., McCree, A., Przybocki, M., y Reynolds, D. A. (2014). The NIST 2014 speaker recognition i-vector machine learning challenge. *Odyssey 2014: Speaker and Language Recognition Workshop*, (June): 224–230.

Hafen, R. P. y Henry, M. J. (2012). Speech information retrieval: A review. *Multimedia Systems*, **18**(6): 499–518.

Hansen, J. H. y Hasan, T. (2015). Speaker recognition by machines and humans: A tutorial review. *IEEE Signal Processing Magazine*, **32**(6): 74–99.

Harwood, B. y Drummond, T. (2016). FANNG: Fast approximate nearest neighbour graphs. En: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-December, pp. 5713–5722.

Hermansky, H., Hanson, B., y Wakita, H. (1985). Perceptually based linear predictive analysis of speech. **10**: 509–512.

Hyvonen, V., Pitkanen, T., Tasoulis, S., Jaasaari, E., Tuomainen, R., Wang, L., Corander, J., y Roos, T. (2016). Fast nearest neighbor search through sparse random projections and voting. En: *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*. IEEE, pp. 881–888.

Indyk, P. y Motwani, R. (1998). Approximate nearest neighbors: Towards removing the curse of dimensionality. En: *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*. ACM, pp. 604–613.

Jégou, H., Douze, M., y Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **33**(1): 117–128.

Jeon, W. y Cheng, Y. M. (2012). Efficient speaker search over large populations using kernelized locality-sensitive hashing. En: *ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings*, mar. IEEE, número 3, pp. 4261–4264.

Jeon, W., Ma, C., y MacHo, D. (2012). Statistical utterance comparison for speaker clustering using factor analysis. *IEEE Transactions on Audio, Speech and Language Processing*, **20**(9): 2482–2491.

Kalantari, I. y McDonald, G. (1983). A data structure and an algorithm for the nearest point problem. *IEEE Transactions on Software Engineering*, (5): 631–634.

Kanagasundaram, A. (2014). *Speaker verification using I-vector features*. Tesis de doctorado, Queensland University of Technology.

Kenny, P. (2005). Joint factor analysis of speaker and session variability: Theory and algorithms. *CRIM, Montreal,(Report) CRIM-06/08-13*, pp. 1–17.

Kenny, P., Mihoubi, M., y Dumouchel, P. (2003). New MAP estimators for speaker recognition. *EUROSPEECH 2003 - 8th European Conference on Speech Communication and Technology*, pp. 2961–2964.

Kinnunen, T. y Li, H. (2010). An overview of text-independent speaker recognition: From features to supervectors. *Speech Communication*, **52**(1): 12–40.

Kinnunen, T., Karpov, E., y Fränti, P. (2006). Real-time speaker identification and verification. *IEEE Transactions on Audio, Speech and Language Processing*, **14**(1): 277–288.

Kuhn, R., Junqua, J. C., Nguyen, P., y Niedzielski, N. (2000). Rapid speaker adaptation in eigenvoice space. *IEEE Transactions on Speech and Audio Processing*, **8**(6): 695–706.

Kulis, B. y Grauman, K. (2012). Kernelized locality-sensitive hashing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **34**(6): 1092–1104.

Leszczynski, K., Loose, S., y Dunscombe, P. (1995). Segmented chamfer matching for the registration of field borders in radiotherapy images. *Physics in Medicine and Biology*, **40**(1): 83–94.

Liu, Y., Nie, L., Liu, L., y Rosenblum, D. S. (2016). From action to activity: Sensor-based activity recognition. *Neurocomputing*, **181**: 108–115.

Lukic, Y., Vogt, C., Durr, O., y Stadelmann, T. (2016). Speaker identification and clustering using convolutional neural networks. *IEEE International Workshop on Machine Learning for Signal Processing, MLSP*, **2016-November**: 1–6.

Mäkinen, V. y Ukkonen, E. (2016). *Point Pattern Matching*, pp. 1589–1592. Springer Science & Business Media.

Malkov, Y., Ponomarenko, A., Logvinov, A., y Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, **45**: 61–68.

Malkov, Y. A. y Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1.

Matas, J. y Chum, O. (2004). Randomized RANSAC with Td,d test. En: *Image and Vision Computing*. Vol. 22, pp. 837–842.

Micó, M. L., Oncina, J., y Vidal, E. (1994). A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters*, **15**(1): 9–17.

Minsky, M. y Papert, S. (1969). *Perceptrons*. MIT Press.

Mok, Mandy Man, K. (2009). The Relationship Between Distinctive Capabilities , Innovativeness , Strategy T ... *International Business*, **8**(11): 21–33.

Nagraniy, A., Chungy, J. S., y Zisserman, A. (2017). VoxCeleb: A large-scale speaker identification dataset. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, **2017-August**: 2616–2620.

Navarro, G. (1999). Searching in metric spaces by spatial approximation. *String Processing and Information Retrieval Symposium and International Workshop on Groupware, SPIRE 1999 and CRIWG 1999*, **11**(1): 141–148.

O'Donnell, R., Wu, Y., y Zhou, Y. (2014). Optimal lower bounds for locality-sensitive hashing (except when q is tiny). *ACM Transactions on Computation Theory*, **6**(1): 5.

Paganelli, C., Peroni, M., Riboldi, M., Sharp, G. C., Ciardo, D., Alterio, D., Orecchia, R., y Baroni, G. (2013). Scale invariant feature transform in adaptive radiation therapy: A tool for deformable image registration assessment and re-planning indication. *Physics in Medicine and Biology*, **58**(2): 287–299.

Pinterest (2009). Recuperado en 2019 de http://www.pinterest.com.

Ranade, S. y Rosenfeld, A. (1980). Point pattern matching by relaxation. *Pattern Recognition*, **12**(4): 269–275.

Reynolds, D. A., Quatieri, T. F., y Dunn, R. B. (2000). Speaker verification using adapted Gaussian mixture models. *Digital Signal Processing: A Review Journal*, **10**(1): 19–41.

Rpforest (2015). Recuperado en 2019 de https://github.com/lyst/rpforest.

Schmidt, L. (2014). Large Scale Speaker Identification. *2014 IEEE International Conference on Acoustic, Speech and Signal Processing (ICASSP)*, pp. 1669–1673.

Senoussaoui, M., Kenny, P., Dehak, N., y Dumouchel, P. (2006). An i-vector Extractor Suitable for Speaker Recognition with both Microphone and Telephone Speech. *Proc. Odyssey Speaker and Language Recogntion Workshop*, (C): 2005–2006.

Shea, J. (2004). *Handbook of fingerprint recognition [Book Review]*, Vol. 20. Springer Science & Business Media. pp. 45–45.

Snyder, D., Garcia-Romero, D., y Povey, D. (2016). Time delay deep neural network-based universal background models for speaker recognition. En: *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015 - Proceedings*, dec. IEEE, número 1232825, pp. 92–97.

Spotify (2006). Recuperado en 2019 de http://www.spotify.com.

Uhlmann, J. K. (1991a). Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, **40**(4): 175–179.

Uhlmann, J. K. (1991b). Satisfying general proximity/similarity queries with metric trees. *Information processing letters*, **40**(4): 175–179.

Veltkamp, R. C. y Hagedoorn, M. (2001). *State of the Art in Shape Matching*. Springer. pp. 87–119.

Wolfson, H. J. y Rigoutsos, I. (1997). Geometric hashing: An overview. *Computing in Science and Engineering*, **4**(4): 10–21.

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

Yianilos, P. N. (1993b). Data structures and algorithms for nearest neighbor search in general metric spaces. En: *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*. Vol. 93, pp. 311–321.