Comparación de Fuerza Bruta y K-D Tree para la Búsqueda de Vecinos Cercanos (KNN K-Nearest Neighbors)

Merisabel Ruelas Quenaya, Fiorela Villarroel Ramos, Jenny Huanca Anquise
Escuela de Ciencia de la Computación
Universidad Nacional de San Agustín de Arequipa
Arequipa, Perú
mruelasq@unsa.edu.pe, fvillarroel@unsa.edu.pe, jhuancaan@unsa.edu.pe

***Abstract*—El vecino más cercano , también conocido como búsqueda de proximidad, búsqueda de similitud o búsqueda del punto más cercano, es un problema de optimización para encontrar los puntos más cercanos en espacios métricos. La búsqueda por fuerza bruta es una técnica de resolución de problemas muy general que consiste en enumerar sistemáticamente todos los posibles candidatos a la solución y comprobar si cada candidato satisface el el enunciado del problema. Un algoritmo de fuerza bruta para un problema de coincidencia de cadenas tiene dos entradas a considerar: patrón y texto. Un árbol k-d, o árbol k-dimensional, es una estructura de datos utilizada para organizar un cierto número de puntos en un espacio con k dimensiones. Los árboles k-d son muy útiles para la búsqueda de rangos y vecinos más cercanos. En este trabajo, estudiamos y comparamos el algoritmo del árbol k-d y el algoritmo de fuerza bruta. Mas específicamente el uso del vecino más cercano aproximado con la estructura de datos del árbol K-d y la comparación de sus propiedades de rendimiento con el enfoque de fuerza bruta.**

El resultado del trabajo realizado en este documento reveló un mejor rendimiento utilizando el árbol k-d en comparación con el enfoque de fuerza bruta.

***Index Terms*—Estructura de Datos, Nearest neighbour, K-NN Aproximado, K-d tree, Fuerza Bruta**

I. Introducción

K Nearest Neighbor (KNN) es un algoritmo, fácil de entender y versátil.

Dado un conjunto de puntos, el problema del vecino más cercano consiste en encontrar los puntos más cercanos a un punto de consulta punto de consulta. Aparece en muchas aplicaciones diferentes. Una de ellas es la búsqueda de instalaciones, por ejemplo, la consulta de cuáles son los restaurantes o parques más cercanos en el teléfono móvil. Otra es la identificación automática reconocer automáticamente los dígitos escritos a mano en los sobres.

En este trabajo, nos centraremos en el problema del vecino más cercano en un contexto espacial, implementaremos varias estructuras de datos y algoritmos bien conocidos para encontrar para encontrar a los vecinos más cercanos y comparar su rendimiento. Una de estas estructuras de datos es el árbol Kd que es un árbol binario especializado y que se describe a continuación.

Adicionalmente, el trabajo tiene por objetivo comprender cómo un problema real en este caso el de vecinos mas cercanos puede ser implementado por diferentes estructuras de datos y algoritmos. Además de evaluar y contrastar el rendimiento de las estructuras de datos con respecto a diferentes escenarios de uso y datos de entrada.

II. Marco Teórico

A. Estructura de Datos

La estructura de datos es un almacenamiento que se utiliza para guardar y organizar datos. Es una forma de organizar los datos en un ordenador para poder acceder a ellos y actualizarlos de forma eficiente. [1]

Dependiendo de las necesidades y del proyecto, es importante elegir la estructura de datos adecuada para el proyecto. Por ejemplo, si se quiere almacenar datos de forma secuencialmente en la memoria entonces se podria optar por un arreglo o Array.

B. KNN de Fuerza Bruta

El algoritmo K-Nearest Neighbors (K-NN o KNN) es utilizado para resolver problemas de clasificación y regresión especialmente en tareas de aprendizaje automático. Comparado con otras técnicas que realizan las mismas tareas, este algoritmo es simple y fácil de implementar por lo que es uno de los más utilizados.

El algoritmo kNN clasifica un punto de datos según la clasificación de sus vecinos. Asume que existen cosas similares en la proximidad, y depende de que esta suposición sea lo suficientemente cierta como para que el algoritmo sea útil.

Los orígenes de kNN se pueden encontrar en investigaciones realizadas para las fuerzas armadas de EE. UU. realizadas por Evelyn Fix (1904–1965) y Joseph Lawson Hodges Jr. (1922–2000) en Berkeley. Juntos escribieron un informe de análisis técnico en 1951 para la USAF, introduciendo el análisis discriminante, un método de clasificación no paramétrico. Sin embargo, el artículo nunca se publicó oficialmente, muy probablemente por razones de confidencialidad a raíz de la Segunda Guerra Mundial.

Esta es la primera introducción conocida de clasificación no paramétrica y más tarde se convertiría en el famoso algoritmo kNN. En 1967, Thomas Cover y Peter Hart demostraron

una tasa de error de límite superior con clasificaciones kNN multiclase. Desde entonces, ha habido muchas mejoras como las desarrolladas por James Keller [2] que desarrolló una versión ”fuzzy” del algoritmo KNN con una tasa de error más baja.

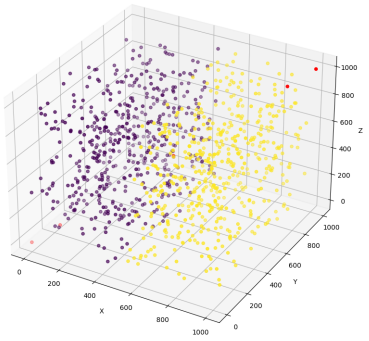
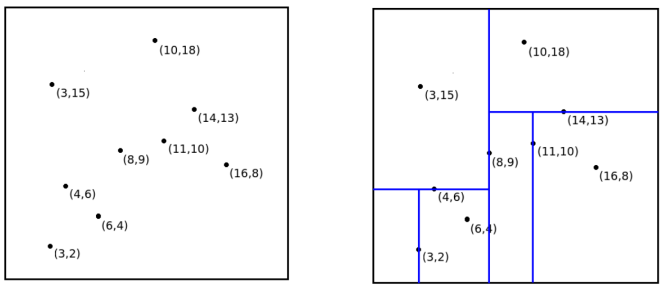
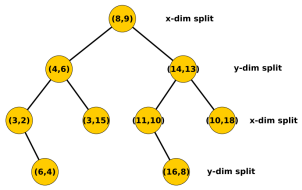


Fig. 1: K-NN para clasificación y predicción de un conjunto de datos



(a) Puntos en el plano

(b) Partición Kd-tree del espacio 2D



(c) Kd-tree

Fig. 2: Puntos del espacio 2D y árbol Kd correspondiente a los puntos

C. KD-Tree

Kd-tree es una estructura de datos en forma de árbol binario que permite realizar búsquedas más eficientes de vecinos más cercanos para consultas espaciales. Construye un índice para buscar rápidamente los vecinos más cercanos. Basándose en el conjunto de puntos de puntos a indexar, los Kd-trees particionan recursivamente un espacio multidimensional y, por lo general, garantizan un árbol binario equilibrado (lo que hace que la búsqueda sea relativamente rápida). árbol binario equilibrado (lo que hace que la búsqueda sea relativamente rápida). En esta tarea nos centramos en espacios 2D (algo con el que estamos familiarizados y que podemos visualizar). En el espacio 2D, tenemos las dimensiones x (horizontal) e y (vertical) dimensiones.

1) Construcción: Dado un conjunto de puntos en un espacio 2D, primero encontramos el punto que es la mediana en la dimensión del eje x. Trazamos una línea vertical que pase por este punto mediano. La mitad de los puntos están a la izquierda de esta línea, y la otra mitad de los puntos están a la derecha. Para cada una de las mitades, encontramos la mediana de los puntos en esas en la dimensión del eje Y, y trazamos una línea horizontal que la atraviese. Los puntos por encima de esta línea forman una partición, los puntos de la otra forman otra. Este proceso continúa para cada partición hasta que todas las particiones contengan un solo punto o estén vacías. Las particiones de un solo punto forman las hojas de nuestro árbol, mientras que las particiones vacías forman nodos nulos.

Como ejemplo, consideremos la figura 2. Inicialmente encontramos el punto medio en la dimensión x, que es (8,9), y dividimos los puntos restantes en los que están a su izquierda y los que están a su derecha, a la derecha del mismo. Los que están a su izquierda forman el subárbol izquierdo (puntos (3,2), (3,15), (4,6), (6,4)) y los otros puntos el subárbol de la derecha. Para los puntos del subárbol de la izquierda, encontramos la mediana en la dimensión y - podría ser (3,15) o (4,6), pero seleccionamos (4,6). Dividimos los puntos en la dimensión y alrededor de (4,6), con aquellos puntos cuyas coordenadas y son inferiores a 6 forman su subárbol izquierdo ((3,2) y (6,4)), mientras que (3,15) forma el subárbol derecho.

2) Búsqueda: Dada una consulta, queremos encontrar qué punto(s) está(n) más cerca de ella. El algoritmo de búsqueda básicamente busca en el árbol.

- Recordemos que cada nodo representa una división en la dimensión x o y. Por ejemplo, en la Figura 1c, el nodo raíz (8,9) es una división en la dimensión x y el nodo (4,6) es una división en la dimensión y.
- El proceso de búsqueda comienza en el nodo raíz. El algoritmo de búsqueda se mueve hacia abajo en el árbol recursivamente, y se desplaza a la izquierda o a la derecha en función del nodo visitado actualmente, de la dimensión que está dividiendo, y el punto de consulta. Por ejemplo, digamos que la consulta es (16,14). En el nodo raíz, que está dividido en la dimensión x, comparamos las coordenadas x - ¿es 16 ¿ 8? Sí, entonces vamos a la rama derecha. En el nodo (14,13), está dividido en la dimensión y, así que comprobamos si 14 ¿ 13, sí, así que bajamos por la rama derecha del subárbol
- Cuando el algoritmo de búsqueda alcanza un nodo hoja, detenemos la recursión y guardamos ese nodo como el punto actual más cercano a la consulta.

- El algoritmo de búsqueda desenrolla/retrocede la recursión del árbol. Básicamente busca si hay otros nodos/puntos predecesores que estén más cerca que el actual más cercano. Para cada nodo predecesor que el algoritmo de búsqueda rastrea hacia atrás:
  - Si el nodo/punto actual está más cerca que el más cercano guardado, el nodo actual se guarda como el nuevo más cercano.
  - El algoritmo de búsqueda debe comprobar si hay puntos al otro lado de la línea de división (del nodo actual) que puedan estar más cerca del punto de consulta. Este es el otro subárbol que no hemos visitado en el árbol kd. Esto ocurre cuando la distancia del punto de consulta al punto actual más cercano es mayor que la distancia más pequeña del punto de consulta a la línea de división. Por lo tanto, si es el caso, el algoritmo de búsqueda debe realizar la búsqueda por esta rama del subárbol no visitada, y luego rebobinar hacia arriba en este subárbol, haciendo las comprobaciones del punto más cercano. Si no es el caso, entonces la otra rama del subárbol no puede contener el punto más cercano y puede ser eliminada de la consideración.
  - Cuando el algoritmo de búsqueda llega al nodo raíz, entonces hemos terminado y el punto actual más cercano es el vecino más próximo/cercano al punto de consulta.

Para buscar los k vecinos más cercanos, mantenemos los k puntos más cercanos (en lugar de uno), y sólo eliminamos una rama del subárbol si los puntos de esa rama no pueden estar más cerca que cualquiera de los k puntos más cercanos actuales.

3) Adición: La adición de un punto al árbol Kd es similar a la construcción. En primer lugar, empezamos en el nodo raíz y determinamos la dimensión en la que se divide el nodo raíz y, a continuación, si hay que ir a la izquierda o a la subárboles de la derecha, en función del punto que se va a insertar. Esto se repite de forma recursiva hasta llegar a un nodo hoja y entonces determinamos si hay que añadir al hijo izquierdo o al derecho.

III. Metodología

En el presente trabajo se busca comparar el rendimiento de los algoritmos KNN (K-Nearest Neighbors) y KD-Tree para la búsqueda de los vecinos más cercanos. Ambos algoritmos fueron implementados en Python y se analizaron el costo computacional de las funciones de construcción, inserción y búsqueda, así como el tiempo de ejecución de cada una de estas funciones en ambos algoritmos.

Respecto al algoritmo de KNN se analizó la relación del incremento del parámetro k con los resultados anteriores. Por otro lado, para el algoritmo de KD-Tree se analizó la relación del incremento del tamaño de la hoja con los resultados anteriores. Para el análisis se usaron 3 conjuntos de datos tridimensionales de tamaños 1000, 10000 y 20000.

IV. Resultados

En este sección se presentan los resultados experimentales para analizar el coste computacional y calcular la distancia y el tiempo de ejecución de los vecinos más cercanos, con un número de puntos de referencia de datos en 3 dimensiones.

A. Análisis del costo computacional de la función de inserción y construcción

En el caso de KNN de Fuerza Bruta la inserción no hace uso de ninguna estructura adicional a una colección de los datos como vectores. Los datos solo son asignados, por lo que su complejidad computacional es de $O(1)$. Además no es necesaria la función de construcción ya que no se usa alguna estructura de datos compleja como un árbol.

Para el caso de KD-Tree, la función de inserción en k-d equilibrado requiere un tiempo $O(\log n)$ y para en el caso de la construcción esta tiene un complejidad de $O(n \log n)$.

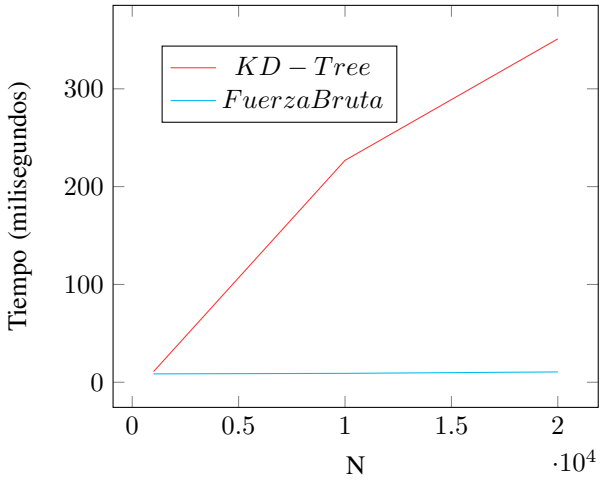
B. Análisis del tiempo de ejecución de la función de inserción

Para analizar y comparar el tiempo de ejecución de la función de inserción con el algoritmo KD-Tree se hizo usó de 3 bases de datos de tamaño 1000, 10000 y 20000, obteniéndose los siguientes resultados:

TABLA I: Tiempo de Inserción en KD-Tree

Tamaño de Datos	Tiempo de Inserción (ms)
1000	7.6160430908
10000	107.3389053345
20000	294.3329811096

1) Tiempo de ejecución de la función de inserción del KD-tree: Como sabemos la complejidad de inserción del algoritmo es de $O(\log n)$ en la siguiente gráfica se puede ver la función de inserción , ahora como se menciono también en el caso de la función de inserción del algoritmo de buerza bruta es de O(1) puesto que como no necesitamos de alguna estructura compleja para el algoritmo , solo se inserta en un arreglo , podemos notar que el tiempo de ejecución usando el KD-tree es mucho mayor que usando el algoritmo de fuerza bruta.



C. Análisis del costo computacional de la función de búsqueda de los k-vecinos más cercanos

La búsqueda de los k-vecinos más cercanos usando KNN de Fuerza Bruta se puede analizar a través de su pseudocódigo [3]:

1) Calcular todas las distancias entre el punto de consulta y los puntos de referencia.
2) Ordenar las distancias calculadas.
3) Seleccionar los k puntos de referencia con las distancias más pequeñas.
4) Repetir los pasos de 1 a 3 para todos los puntos de consulta.

Para el paso 1, sea *N* puntos de referencia y $O(d)$ el coste de calcular cada distancia para un punto se tiene una complejidad de $O(N*d)$. Para los paso 2 y 3, se ordena la lista de distancia en $O(logN)$ en promedio y luego seleccionar los k vecinos es $O(1)$ en la lista ordenada. Para un solo punto, este proceso costaría $O(N*d+logN)$. Como la búsqueda de los vecinos debe ser para todos los puntos de consulta *M*, entonces la complejidad de toda la búsqueda seria $O(M*N*d)$.

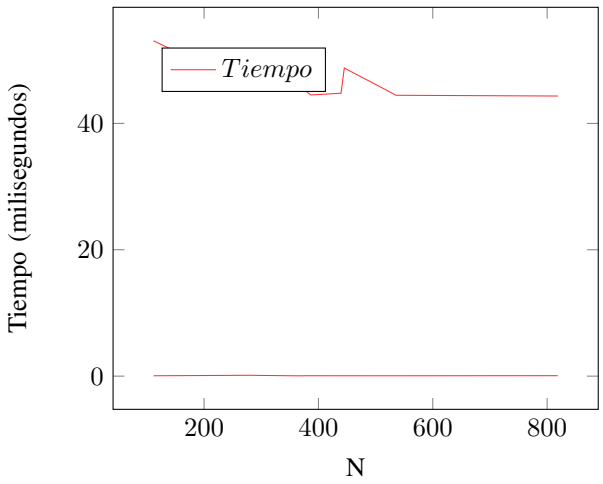
Para el caso de KD-Tree, la búsqueda de los k-vecinos más cercanos En cuanto a la fase de predicción, la estructura de árbol kd el enfoque simple es solo consultar k los tiempos, eliminando el punto encontrado cada vez; dado que la consulta toma $O(logN)$, es $k*O(logN)$ en total. Pero dado que el árbol kd ya corta espacio durante la construcción, después de una sola consulta, podemos saber aproximadamente dónde buscar solo podemos buscar alrededor de ese punto. Como hemos implementado un árbol KD-Tree equilibrado. se usa la mediana durante la construcción del árbol, por ello el costo computacional sería $O(logN)$

D. Análisis del tiempo de ejecución de la función de búsqueda de k-vecinos más próximos

Dado un conjunto de datos a consultar de tamaño 10, se comparó el tiempo que tomaba para cada algoritmo el cálculo de su vecino más cercano. Los resultados obtenidos al usar el KNN de Fuerza Bruta y el KD-Tree se muestran a continuación en la tabla II.

TABLA II: Comparación de Tiempo de Búsqueda para KNN y KD-Tree

Distancia	Tiempo Fuerza Bruta	Tiempo KD-Tree
279.7659	45.8443(ms)	0.1345(ms)
370.5792	45.6598(ms)	0.0491(ms)
535.9645	44.4524(ms)	0.0576(ms)
445.2381	48.7687(ms)	0.0594(ms)
348.8510	47.4420(ms)	0.0644(ms)
386.4829	44.4984(ms)	0.0643(ms)
374.2004	45.4103(ms)	0.0625(ms)
439.5065	44.7812(ms)	0.0594(ms)
111.5750	53.0724(ms)	0.0582(ms)
818.6562	44.3329(ms)	0.0713(ms)



1) Tiempo de búsqueda de los k-vecinos más cercanos del KNN de Fuerza Bruta: En el caso de KNN de Fuerza Bruta, el tiempo de búsqueda obtenido fue bastante bajo, menor a 100 milisegundos. El vecino más cercano de cada uno de los 10 puntos y el tiempo que tomó encontrarlo se grafíca en la Figura 3.

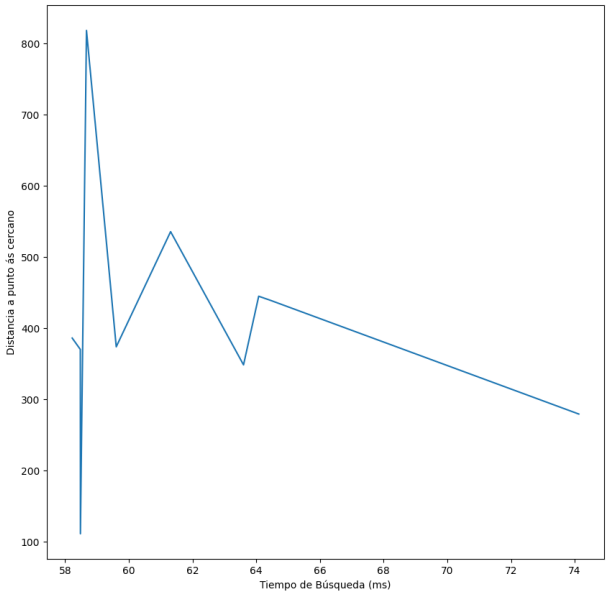


Fig. 3: Tiempo de búsqueda y distancia de KNN

2) Tiempo de búsqueda de los k-vecinos más cercanos del KD-Tree: Se puede observar en la tabla II que el tiempo de búsqueda del primer vecino más cercano a un punto dado es mucho menor llegando a ser menos de 1 milisegundo usando el algoritmo que usa la estructura del KD-Tree a comparación del algoritmo cuyo enfoque es la Fuerza Bruta , esto también podemos notarlo al comparar sus complejidades en el peor de

los casos siento la complejidad del algoritmo de fuerza Bruta $O(M * N * d)$ mucho mayor que el de la complejidad del algoritmo que usa el KD-Tree $O(logN)$.

E. Relación del incremento del parámetro k en KNN de Fuerza Bruta

Al momento de incrementar **k**, el número de vecinos a buscar para cada punto del conjunto de puntos a consultar, el tiempo de búsqueda usando KNN de Fuerza Bruta no sufre variaciones significativas, al menos para el rango de 1 a 100 vecinos como se observa en la siguiente figura 4.

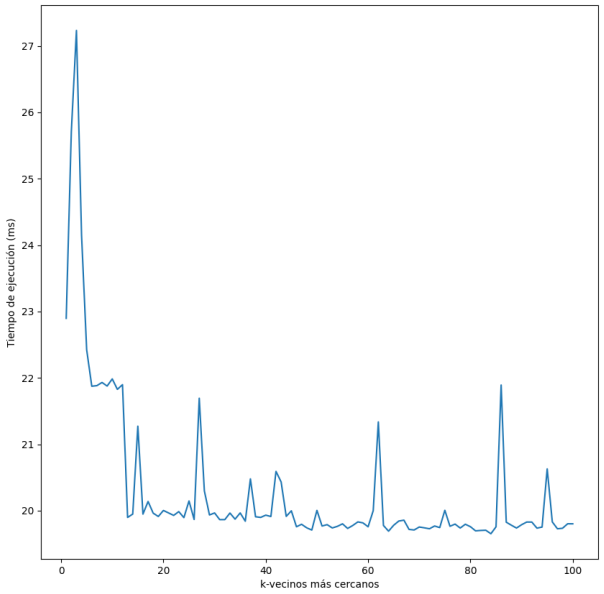


Fig. 4: Tiempo de ejecución de KNN para búsqueda de k-vecinos

En cuanto a a relación del incremento del parámetro *k*, respecto al tiempo que toma hacer la predicción de la clase de los puntos consultados, se obtuvieron resultados similares a los tiempos de búsqueda (Figura 5). Las variaciones fueron de unas decenas de milisegundos, pero no guardaron relación con la cantidad de vecinos a tomar en cuenta para la votación de clases.

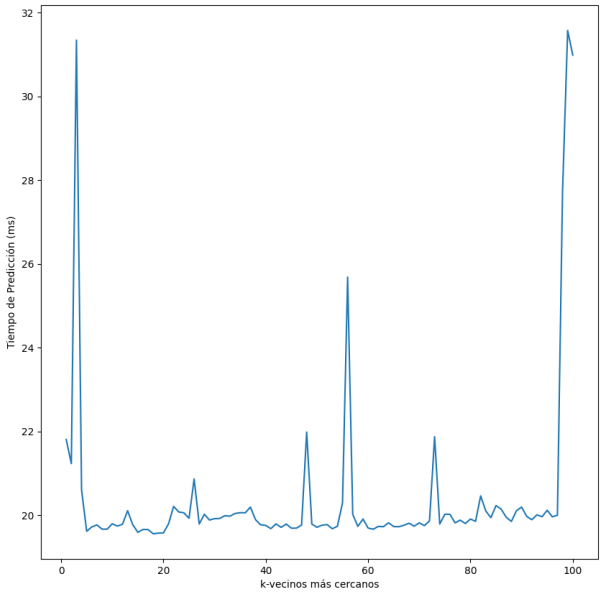


Fig. 5: Tiempo de predicción de clase con KNN usando k-vecinos

F. Relación del incremento del tamaño de las hojas en KD-Tree

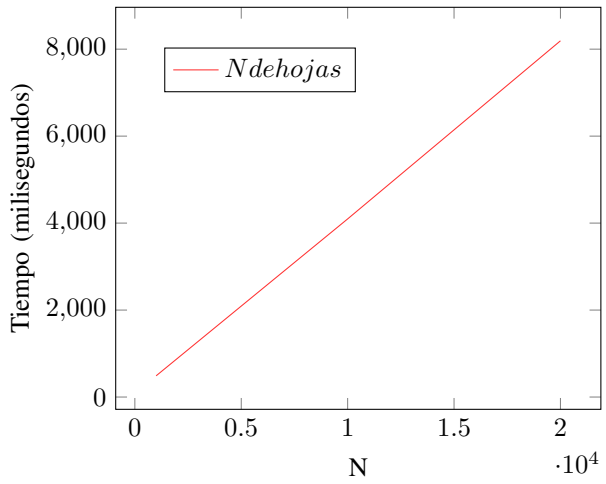
TABLA III: Relación del incremento del tamaño de hojas del KD-Tree

N	**N° de hojas**
1000	488
10000	4096
20000	8192

En la tabla III se puede observar que la relación de incremento es directamente proporcional con la cantidad de datos , como sabemos en un árbol binario la cantidad de datos es de $\frac{n+1}{2}$ donde n es la cantidad de nodos en el árbol , con la data de 1000 nodos y 10000 nodos , sabemos que son directamente proporcionales con una razón de 10 por lo que el número de sus nodos hojas también están en esa razón de 10 , por lo tanto podemos decir que la relación de incremento es igual a $K = \frac{NA}{NB}$ donde NA y NB son el número de data.

V. Discusión

Respecto al incremento del parámetro *k* al momento de hacer la consulta de los vecinos más próximos y la predicción de clases, el tiempo de ejecución para valores *k* de 1 a 100 no muestró diferencias significativas. La elección de un parámetro *k* idóneo que garantice un tiempo adecuado de ejecución y una buena elección de clase es un problema que ha sido estudiado y se han dado algunas reglas para fijar este valor. Observamos



que la elección de *k* está determinado por la densidad de los puntos y debería hacerse en relación a N [4]. Así, una elección adecuada de k en función de N es: $cte\sqrt{N}$.

De forma teórica se sabe que en un espacio de representación bidimensional dado un patrón cualquiera *X* (representado como un punto bidimensional) en el que se consideran los *k* vecinos más próximos a *X*, éstos estarán localizados en un círculo centrado en *X*. El área del círculo que encierra un número fijo de tales puntos *k*, es menor en regiones densamente pobladas que en regiones donde los puntos están más dispersos. Lo mismo ocurre en espacios multidimensionales, donde el círculo se convierte en una hiperesfera.

VI. Conclusiones

El trabajo presentado analizó el rendimiento del uso de KD-Tree y KNN de Fuerza bruta para encontrar los vecinos más cercanos con el fin de conocer cuál de los dos tiene un mejor rendimiento. En general, se observó que KD-TRee tiene un tiempo de ejecución menor en las tareas de inserción y búsqueda para tamaños no muy grandes de datos, mientras que KNN de Fuerza Bruta demora más en la búsqueda, pero su inserción de datos es mucho más sencilla. En conclusión, En la mayoría de los casos, sobretodo cuando se busca un mejor rendimiento y los datos no es tan distribuidos en áreas densas, el KD-Tree en la mejor opción.

Referencias

[1] M. A. Weiss and Y. Chen, *Data structures and algorithm analysis in C*. Benjamin/Cummings California, 1993.

[2] J. M. Keller, M. R. Gray, and J. A. Givens, "A fuzzy k-nearest neighbor algorithm," *IEEE transactions on systems, man, and cybernetics*, no. 4, pp. 580–585, 1985.

[3] D. Verma, N. Kakkar, and N. Mehan, "Comparison of brute-force and k-d tree algorithm," *International Journal of Advanced Research in Computer and Communication Engineering*, no. 3, pp. 291–297, 2014.

[4] C. Cambronero García and I. Moreno Gómez, "Algoritmos de aprendizaje: Knn & kmeans [inteligencia en redes de telecomuncicación]." Last accessed: 12-1-2022.