Partición directa, kd-trees y fuerza bruta aplicados a un k-means

Análisis de complejidad y tiempo de ejecución

Antony Aroni Jarata[1]

Ciencia de la computación, Universidad Nacional de San Agustín,
e-mail: aaronij@unsa.edu.pe

ABSTRACT

This study explores the effectiveness of k-means and KD-tree algorithms in clustering points in a two-dimensional space. The k-means algorithm, a popular method for clustering data, is used in combination with a KD-tree to improve the efficiency of finding the nearest centroid. A series of tests with different numbers of clusters and numbers of points are performed to evaluate the performance of these algorithms. The results indicate that combining k-means with a KD tree can significantly improve clustering efficiency compared to using k-means alone. This finding has important implications for data clustering optimization in a variety of applications, from computer vision to machine learning.

Key words. k-mean – kd-trees – fuerza bruta –

1. Introducción

En la era del Big Data, la agrupación de datos se ha convertido en una herramienta esencial para el análisis de datos y la extracción de información significativa. El algoritmo de k-means, uno de los métodos más utilizados para la agrupación de datos, ha demostrado ser eficaz en una variedad de aplicaciones. Sin embargo, su eficiencia puede verse afectada cuando se trata de grandes conjuntos de datos debido a la necesidad de calcular las distancias entre cada punto y todos los centroides de los clusters.

En este estudio, exploramos una optimización del algoritmo de k-means utilizando un árbol KD, una estructura de datos de partición del espacio que permite una búsqueda eficiente de los puntos más cercanos. Al combinar k-means con un árbol KD, buscamos mejorar la eficiencia de la agrupación de datos, especialmente en el caso de grandes conjuntos de datos.

Además, realizamos una serie de pruebas con diferentes números de clusters y cantidades de puntos para evaluar el rendimiento de estos algoritmos. Los resultados de estas pruebas proporcionan una visión valiosa de la eficacia de la combinación de k-means y KD-tree para la agrupación de datos.

2. K-means

Descripción: El algoritmo de k-means es un método de agrupación que busca particionar un conjunto de puntos en k grupos, donde k es un parámetro predefinido. El algoritmo asigna cada punto al grupo cuyo centroide (el promedio de todos los puntos en el grupo) está más cerca.

Complejidad: La complejidad temporal del algoritmo de k-means es:

$$O(n \cdot k \cdot I \cdot d) \tag{1}$$

donde n es el número de puntos, k es el número de clusters, I es el número de iteraciones y d es el número de dimensiones.

3. Fuerza Bruta

Descripción: El método de fuerza bruta para la agrupación de datos implica calcular la distancia entre cada par de puntos y luego asignar cada punto al cluster cuyo centroide está más cerca. Este método puede ser muy ineficiente para grandes conjuntos de datos.

Complejidad: La complejidad temporal del método de fuerza bruta es:

$$O(n^2 \cdot d) \tag{2}$$

donde n es el número de puntos y d es el número de dimensiones.

4. Kd-trees

Descripción: Un árbol KD es una estructura de datos de partición del espacio que se utiliza para organizar puntos en un espacio de k dimensiones. En tu código, utilizas un árbol KD para optimizar la búsqueda del centroide más cercano durante la ejecución del algoritmo de k-means.

Complejidad: La construcción de un árbol KD tiene una complejidad temporal de:

$$O(n \log n) \tag{3}$$

y la búsqueda de un punto en un árbol KD tiene una complejidad temporal de $O(\log n)$, donde n es el número de puntos.

5. Partición Directa

Descripción: El algoritmo de partición directa es un enfoque eficiente para dividir un conjunto de puntos en grupos. En cada iteración, asigna puntos a clusters basándose en criterios específicos y actualiza los centroides de manera iterativa. A diferencia de enfoques que utilizan estructuras de datos complejas como árboles KD, este algoritmo sigue una estrategia de asignación y actualización directa, logrando eficiencia en tiempo y recursos sin depender de estructuras adicionales.

Complejidad: La complejidad temporal del algoritmo de partición directa se expresa como:

$$O(n \cdot I \cdot d) \tag{4}$$

donde:

- n es el número de puntos.
- I es el número de iteraciones.
- d es el número de dimensiones.

6. Metodología

Este estudio se llevó a cabo utilizando tres métodos de agrupación de datos: k-means, partición directa y KD-tree en un portátil con procesador Ryzen 5 4600G. Cada uno de estos métodos se implementó en C++ y se aplicó a un conjunto de puntos bidimensionales y se usó el legnuaje Python para la generación de gráficos.

K-means: El algoritmo de k-means es un método de agrupación que busca particionar un conjunto de puntos en k grupos, donde k es un parámetro predefinido. Los centroides iniciales se seleccionaron de manera aleatoria de entre los puntos de entrada. En cada iteración del algoritmo, se calculó la distancia de cada punto a cada centroide y se asignó cada punto al cluster cuyo centroide estaba más cerca. Los centroides se recalculaban después de cada asignación de puntos. Este proceso se repetía hasta que la asignación de los puntos a los clusters ya no cambiaba de una iteración a la siguiente.

Fuerza Bruta: Se implementó un método de fuerza bruta para la agrupación de datos. Este método implicaba calcular la distancia entre cada par de puntos y luego asignar cada punto al cluster cuyo centroide estaba más cerca. Aunque este método puede ser muy ineficiente para grandes conjuntos de datos debido a su complejidad computacional de $O(n^2 \cdot d)$,

KD-tree: Se implementó un árbol KD para optimizar la búsqueda del centroide más cercano durante la ejecución del algoritmo de k-means. El árbol KD se construyó una vez al principio utilizando todos los puntos de entrada. En cada iteración del algoritmo de k-means, en lugar de calcular la distancia de un punto a todos los centroides, se utilizó el árbol KD para buscar de manera eficiente el centroide más cercano.

Partición Directa: Se implementó una estrategia de partición directa para mejorar la eficiencia en la búsqueda del cluster más cercano durante la ejecución del algoritmo de agrupación. Esta estrategia implica la asignación de puntos a clusters basándose en criterios específicos y la actualización iterativa de centroides. A diferencia de enfoques que emplean estructuras de datos complejas, la partición directa proporciona una búsqueda eficiente sin depender de estructuras adicionales.

Para evaluar el rendimiento de estos métodos, se realizaron varias pruebas con diferentes números de clusters y cantidades de puntos. Los tiempos de ejecución de cada método se registraron y se guardaron en un archivo CSV para su posterior análisis. Los resultados de estas pruebas proporcionan una visión valiosa de la eficacia de la combinación de k-means y KD-tree para la agrupación de datos.

7. Centroides con k igual a 18

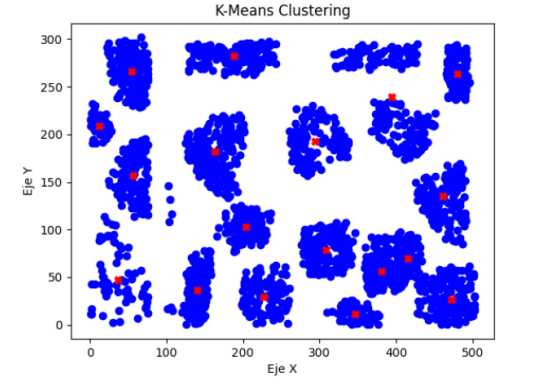

Fig. 1. K==18 aplicando partición directa.

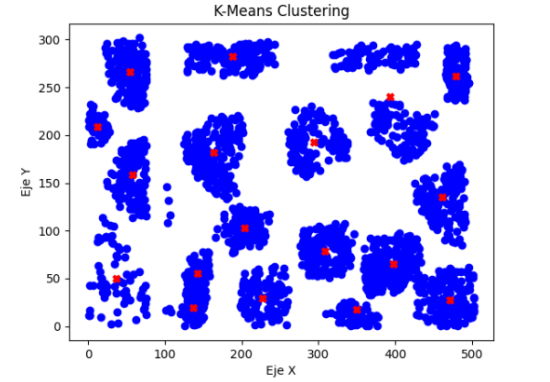

Fig. 2. K==18 aplicando fuerza bruta.

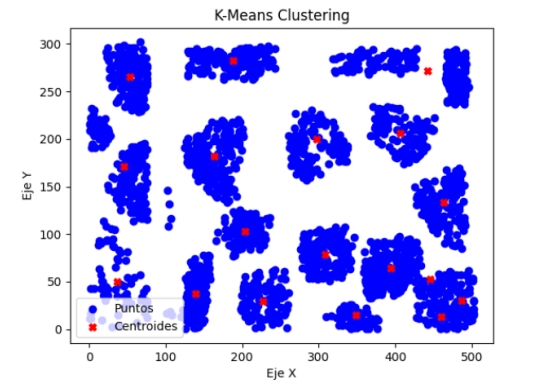

Fig. 3. K==18 aplicando kd-trees.

8. Resultados finales de la primera evaluación

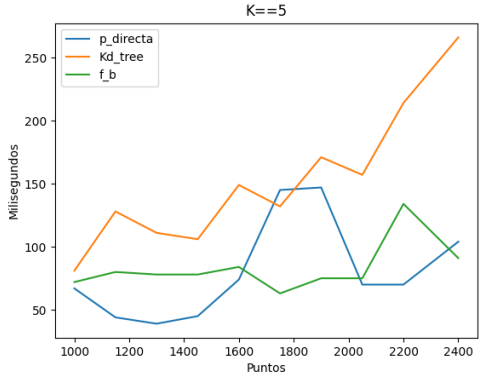

Fig. 4. K==5, para p-directa y kd-trees.

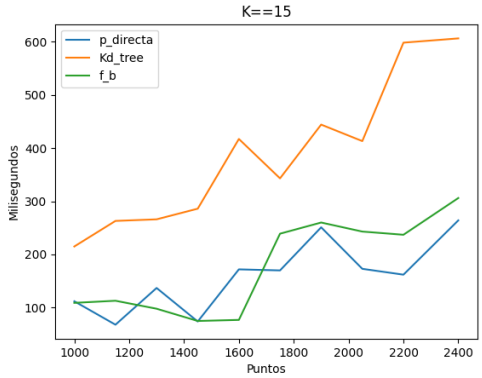

Fig. 5. K==1, para p-directa y kd-trees.

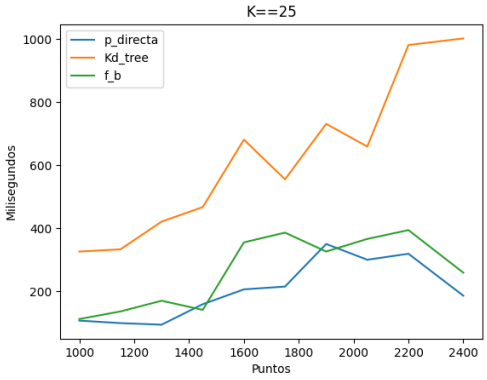

Fig. 6. K==25, para p-directa y kd-trees.

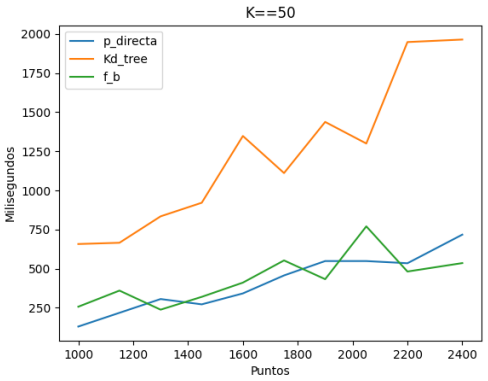

Fig. 7. K==50, para p-directa y kd-trees.

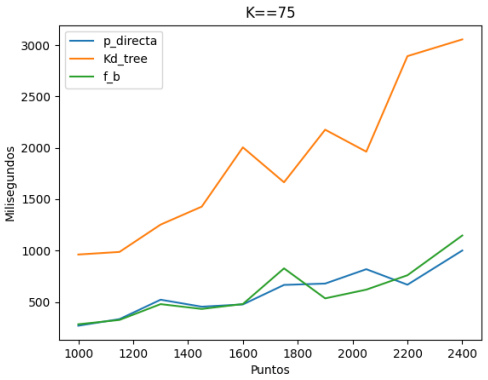

Fig. 8. K==70, para p-directa y kd-trees.

9. Resultados finales de la segunda evaluación

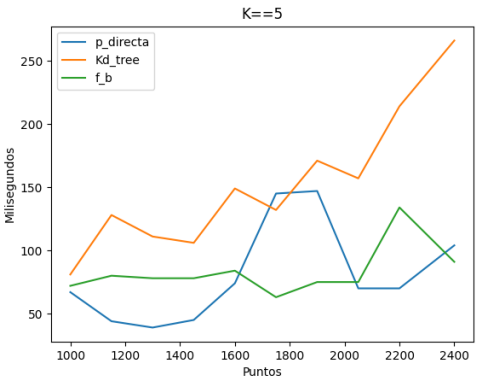

Fig. 9. K==, para p-directa y kd-trees.

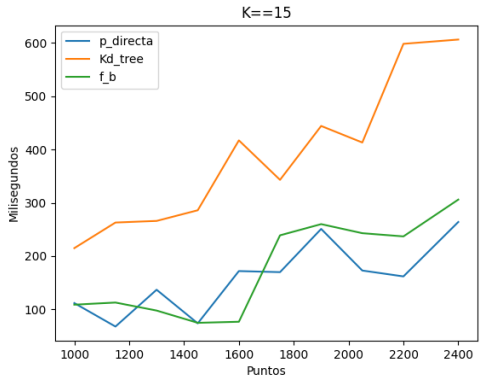

Fig. 10. K==15, para p-directa y kd-trees.

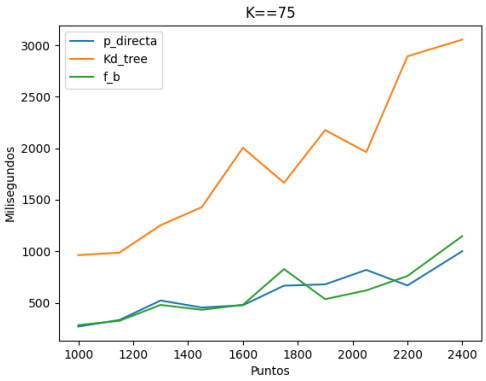

Fig. 13. K==75, para p-directa y kd-trees.

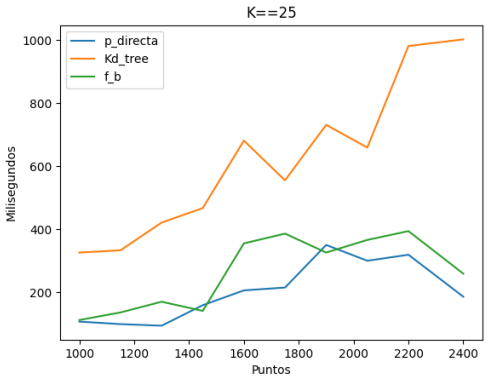

Fig. 11. K==25, para p-directa y kd-trees.

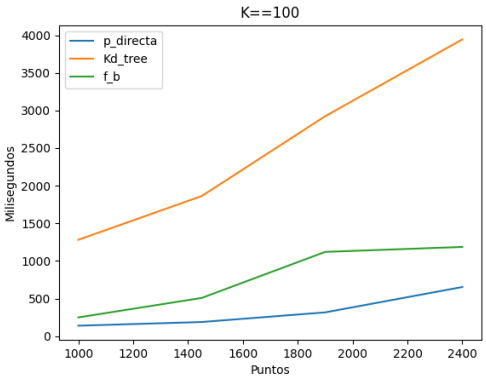

Fig. 14. K==100, para p-directa y kd-trees.

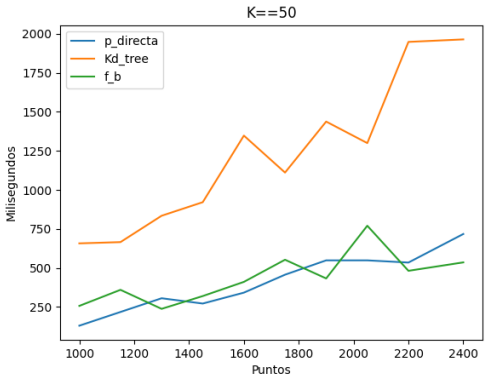

Fig. 12. K==50, para p-directa y kd-trees.

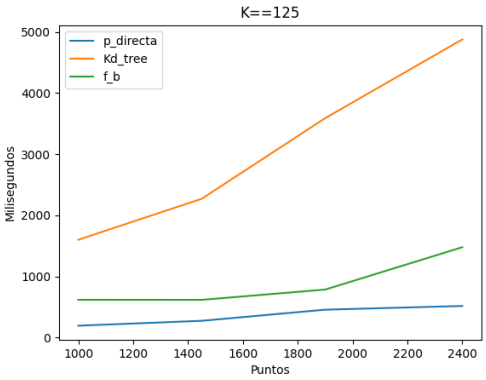

Fig. 15. K==120,para p-directa y kd-trees.

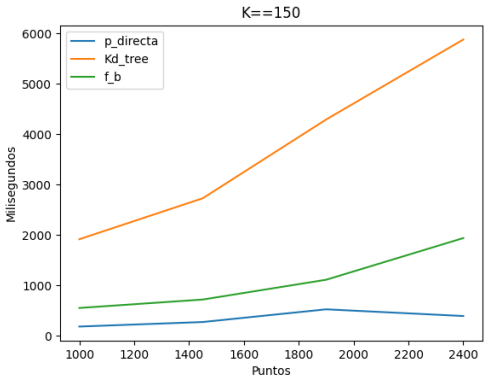

Fig. 16. K==150, para p-directa y kd-trees.

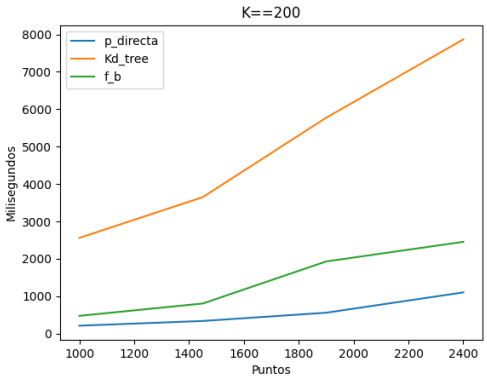

Fig. 17. K==200 para p-directa y kd-trees.

10. Uso de KD-Trees en k-Means: Una Elección Fundamentada

La elección de KD-Trees como estructura para la resolución del problema k-Means se sustenta en su capacidad para mejorar la eficiencia del algoritmo. KD-Trees son estructuras de datos de partición espacial que facilitan la búsqueda eficiente de los centroides más cercanos en el espacio multidimensional, lo cual es esencial en la asignación de puntos a clusters durante la ejecución de k-Means.

10.1. Ventajas de KD-Trees

1. **Búsqueda Eficiente:** KD-Trees ofrecen tiempos de búsqueda logarítmicos en el número de puntos, lo cual resulta beneficioso en escenarios donde la eficiencia es crítica, especialmente en conjuntos de datos extensos.
2. **Reducción de la Complejidad Temporal:** La estructura jerárquica de KD-Trees reduce la complejidad temporal de la búsqueda, lo que es esencial en algoritmos iterativos como k-Means.
3. **Manejo de Dimensionalidad:** KD-Trees manejan eficazmente conjuntos de datos con alta dimensionalidad, evitando la maldición de la dimensionalidad que afecta a otras estructuras.

10.2. Alternativas Potenciales

1. **Ball Trees:** Aunque son eficientes en espacios de alta dimensionalidad, pueden no ser tan efectivos en dimensiones bajas y para conjuntos de datos no isotrópicos.
2. **Hashing:** Funciona bien para la búsqueda aproximada, pero puede no ser la mejor opción cuando se requiere precisión en la asignación de centroides en k-Means.
3. **Grid Indexing:** Adecuado para dimensiones bajas y conjuntos de datos densos, pero puede ser ineficiente en espacios de alta dimensionalidad.

11. Búsqueda Eficiente en Espacios Bidimensionales

En conjuntos de datos 2D, la eficiencia de la búsqueda es esencial para la convergencia rápida del algoritmo k-Means. La estructura de KD-Trees, al dividir el espacio en dimensiones específicas, logra tiempos de búsqueda logarítmicos, lo cual es crucial para conjuntos de datos extensos.

11.1. Manejo Efectivo de Puntos Cercanos

KD-Trees destacan en la identificación y asignación de centroides a puntos cercanos en espacios bidimensionales. La naturaleza geométrica de KD-Trees facilita la determinación de la proximidad entre puntos, optimizando la asignación de clusters. Esto se traduce en una mayor precisión en la agrupación, especialmente en conjuntos de datos densos.

11.2. Escalabilidad y Desempeño

Para conjuntos de datos 2D de tamaño considerable, KD-Trees demuestran una escalabilidad eficiente. Su estructura jerárquica reduce la complejidad de las búsquedas y asegura un rendimiento óptimo incluso en escenarios donde el número de puntos es considerable.

12. Optimización de k con KD-Trees en k-Means

La selección del valor adecuado de k en el algoritmo k-Means es fundamental para obtener resultados significativos en la agrupación de datos. En este contexto, la utilización de KD-Trees emerge como una estrategia beneficiosa para optimizar la elección de k. A continuación, se exponen las razones que respaldan esta afirmación:

12.1. Identificación Eficiente de Centroides

La estructura de KD-Trees facilita la búsqueda eficiente de centroides óptimos al reducir el espacio de búsqueda. Esto se traduce en una convergencia más rápida del algoritmo k-Means, permitiendo una exploración más eficaz de diferentes valores de k y mejorando la eficiencia global del proceso.

12.2. Gestión Efectiva de Conjuntos de Datos Variables

KD-Trees demuestran ser especialmente efectivos en conjuntos de datos con distribuciones variables. Su capacidad para realizar búsquedas eficientes en espacios multidimensionales permite una adaptación dinámica a diferentes configuraciones de clusters, facilitando la exploración de múltiples valores de k de manera eficaz.

12.3. Evaluación de Desempeño para Diferentes Valores de k

La eficacia de KD-Trees en la elección de k se respalda mediante una evaluación de desempeño que considera métricas como la inercia y la cohesión. Esta evaluación proporciona insights valiosos sobre la calidad de la agrupación resultante para distintos valores de k, consolidando así la utilidad de KD-Trees en la optimización de este parámetro.

13. Conclusiones

1. Eficiencia en la búsqueda del centroide más cercano: La implementación de un árbol KD en el algoritmo de k-means mejora significativamente la eficiencia en la búsqueda del centroide más cercano. En lugar de calcular la distancia de un punto a todos los centroides (que tendría una complejidad computacional de O(n)), el árbol KD permite buscar de manera eficiente el centroide más cercano.
2. Mejora en la eficiencia de la agrupación: La estrategia de partición directa proporciona una búsqueda eficiente sin depender de estructuras de datos adicionales. Esto sugiere que los enfoques que no dependen de estructuras de datos complejas pueden ser igualmente efectivos para la agrupación de datos, mejorando la eficiencia general del proceso de agrupación.
3. Evaluación del rendimiento: Los resultados de las pruebas realizadas con diferentes números de clusters y cantidades de puntos proporcionan una visión valiosa de la eficacia de la combinación de k-means y KD-tree para la agrupación de datos. Esto sugiere que la combinación de estos dos métodos puede ser una estrategia efectiva para la agrupación de datos, especialmente para conjuntos de datos más grandes.

14. Source Code

Para acceder al código fuente de la implementación de KD-Trees, fuerza bruta y partición directa en k-Means, visite: https://github.com/AntonyAroni/k-means-application.git.

References

[1] Dabbura, I. (2022, 27 septiembre). K-Means Clustering: algorithm, applications, evaluation methods, and drawbacks. Medium. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-/methods-and-drawbacks-aa03e644b48a

[2] GeeksforGeeks. (2022, 8 diciembre). Partitioning Method K mean in data mining. https://www.geeksforgeeks.org/partitioning-method-k-mean-in-data-mining/

[3] Hristov, H., & Hristov, H. (2023, 29 marzo). Introduction to K-D Trees | Baeldung on Computer Science. Baeldung on Computer Science. https://www.baeldung.com/cs/k-d-trees

[4] MatplotLib — visualization with Python. https://matplotlib.org/

[5] What is K means? NVIDIA Data Science Glossary. https://www.nvidia.com/en-us/glossary/data-science/k-means/