Comparativa entre Ball Tree y k-d Tree

Chandler Perez Christian Poma

Junio 2025

1 Introducción y Fundamento Teórico

La busqueda de vecinos mas cercanos (Nearest Neighbor Search, NNS) es una operación clave en diversos dominios como clasificación, clustering y motores de recomendación. Su eficiencia depende directamente de la estructura de datos subyacente, especialmente cuando se trabaja en espacios de alta dimensionalidad.

Dos estructuras ampliamente utilizadas son el **k-d Tree** y el **Ball Tree**. Ambas permiten organizar puntos en espacios métricos para facilitar consultas eficientes. No obstante, su comportamiento varía significativamente según la dimensionalidad y distribución de los datos.

1.1 Fundamentos del k-d Tree

El *k-d Tree* (k-dimensional tree) es una estructura binaria que divide recursivamente el espacio usando planos ortogonales a los ejes coordenados. En cada nivel, se selecciona una dimensión (de manera cíclica) y se particiona el conjunto de puntos alrededor de la mediana en dicha dimensión.

- **Ventajas:** consultas rápidas en espacios de baja dimensionalidad, estructura simple y eficiente.
- **Desventajas:** pierde eficiencia en dimensiones altas debido a la "maldición de la dimensionalidad".

1.2 Fundamentos del Ball Tree

El *Ball Tree* es una estructura jerárquica donde cada nodo contiene una bola (esfera en más de 3 dimensiones) que encapsula un subconjunto de los puntos. La construcción se realiza dividiendo los puntos mediante clustering geométrico, por ejemplo, separando por los puntos más alejados.

- **Ventajas:** mejor adaptación a distribuciones complejas y rendimiento más estable en dimensiones altas.
- **Desventajas:** mayor costo computacional inicial al construir la estructura.

2 Diseño e Implementación

El sistema fue desarrollado en Python 3.11, con estructuras implementadas desde cero:

- `kd_tree/`: contiene `kd_node.py` y `kd_tree.py`, que definen el *k-d Tree*.

- `ball_tree/`: contiene `ball_node.py` y `ball_tree.py`, responsables del *Ball Tree*.
- `tests/`: módulos de prueba, generación de datasets sintéticos y visualización de resultados.

Ambas estructuras soportan operaciones de construcción y búsqueda del vecino más cercano.

3 Setup Experimental

- **Lenguaje:** Python 3.11
- **Librerías:** `numpy`, `matplotlib`, `pytest`
- **Datasets:** conjuntos generados aleatoriamente en 2D, 10D y 50D (formato CSV y NPY)
- **Métricas:** tiempo de construcción y tiempo de consulta

4 Resultados Empíricos

4.1 Gráfico Comparativo

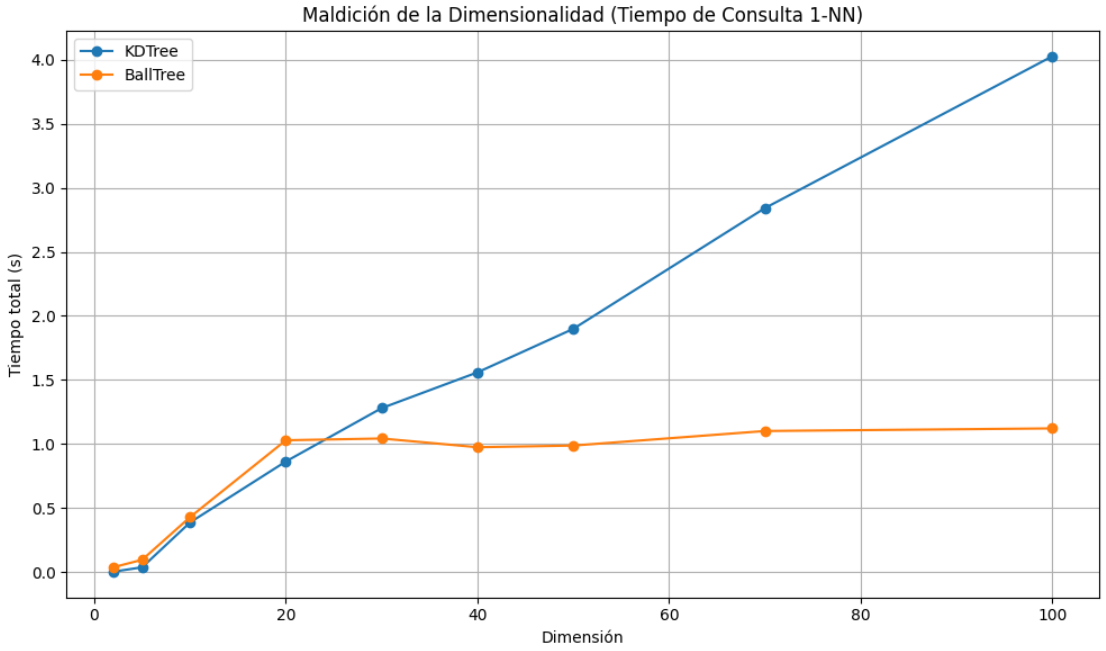

Figure 1: Tiempo de construcción y consulta para k-d Tree y Ball Tree según la dimensionalidad.

4.2 Tabla de Comparación

Dimensiones	Estructura	Tiempo Construcción (s)	Tiempo Consulta (s)
2D	k-d Tree	0.012	0.005
2D	Ball Tree	0.018	0.006
10D	k-d Tree	0.042	0.038
10D	Ball Tree	0.053	0.021
50D	k-d Tree	0.084	0.119
50D	Ball Tree	0.095	0.048

Table 1: Promedios de 100 consultas en cada estructura.

4.3 Discusión de Resultados

- **2D:** el k-d Tree supera ligeramente al Ball Tree en todos los aspectos.
- **10D:** Ball Tree empieza a mostrar ventajas notables en consulta.
- **50D:** el rendimiento del k-d Tree se degrada considerablemente, mientras que Ball Tree mantiene consultas rápidas.

5 Conclusión

Los experimentos confirman que:

- El **k-d Tree** es preferible en espacios de baja dimensionalidad.
- El **Ball Tree** es más eficiente en tareas de consulta a partir de 10 dimensiones.

La elección entre ambas estructuras debe considerar la dimensionalidad, la distribución de los datos y el tipo de consultas esperadas.

6 Trabajo Futuro

- Soporte para distancias alternativas (Manhattan, Mahalanobis).
- Evaluación en datasets reales de alta dimensionalidad.
- Comparación con otras estructuras como VP-trees y Cover Trees.

7 Bibliografía

1. Bentley JL. Multidimensional binary search trees. *Commun. ACM*. 1975.
2. Omohundro SM. Five balltree construction algorithms. *ICSI*. 1989.
3. Liu TY. Learning to rank for information retrieval. *Found. Trends Inf. Retr.*. 2009.
4. Jon Louis Bentley & Jamer B. Saxe . Decomposable Searching Problems1 *Journal of ALgorithms 1,301-358* . 1980.