

★ Get unlimited access to the best of Medium for less than \$1/week. [Become a member](#)



KD-TREE AND BALL TREE IN KNN ALGORITHM



Narasimharaodevisetti · [Follow](#)

7 min read · Jan 23, 2024



1



INTRODUCTION

In this blog, we will explore two algorithms for finding the nearest neighbors of a data point in a multidimensional space: kd-Tree and Ball Tree. These algorithms are based on tree data structures that partition the space into regions and allow for efficient search operations

What are kd — Tree and Ball Tree algorithms?

A kd — Tree (k-dimensional tree) is a data structure that recursively subdivides the space into regions associated with specific data points. The primary objective of a kd — Tree is to facilitate efficient multidimensional search operations, particularly nearest-neighbor searches. The algorithm constructs a binary tree in which each node represents a region in the multidimensional space, and the associated hyperplane is aligned with one of the coordinate axes. At each level of the tree, the algorithm selects a dimension to split the data, creating two child nodes. This process continues recursively until a termination condition is met, such as a predefined depth or a threshold number of points per node.

A Ball Tree is another data structure that organizes the data points into a tree structure by enclosing them within hyperspheres. The main goal of a Ball

Tree is to reduce the search complexity for finding the nearest neighbors of a data point in a high-dimensional space. The algorithm builds a binary tree in which each node represents a hypersphere that contains a subset of the data points. The root node contains all the data points, and the leaf nodes contain only a few or even one data point. The algorithm splits the data points at each level of the tree by finding the two hyperspheres that minimize the sum of their volumes, creating two child nodes. This process continues recursively until a termination condition is met, such as a predefined depth or a threshold number of points per node.

How can we understand these algorithms with real world examples?

To illustrate how these algorithms work, let us consider a simple example of finding the nearest neighbor of a query point in a two-dimensional space.

Suppose we have the following data points with two dimensions

$(x,y) = (1,9), (2,3), (4,1), (3,7), (5,4), (6,8), (7,2), (8,8), (7,9), (9,6)$

And our query point is $(6,5)$.

Explain the algorithm by drawing analogy from real world examples

KD-TREE ANALOGY

Real-World Analogy: Grocery Store Layout

create a KD-Tree to represent the layout of the store.

1. *Building the KD-Tree:*

- The KD-Tree divides the store into sections along different dimensions (e.g., aisles for fruits, vegetables, dairy).
- Each node in the KD-Tree represents a specific location in the store, and the splitting dimension alternates at each level (e.g., one level for x-coordinate, the next for y-coordinate).

2. *Searching for Neighbors:*

- When a shopper comes in looking for an item, you traverse the KD-Tree based on the dimensions that are most relevant to the shopper's needs.
- At each level, you decide whether to explore the left or right subtree, guiding the shopper toward the relevant section.

3. Finding Nearest Neighbors:

- As the shopper gets closer to the target section, you focus on a smaller region of the KD-Tree, narrowing down the search space efficiently.
- Once in the target section, the shopper easily finds the nearest neighbors (items) without having to explore the entire store.

In the KNN algorithm, the KD-Tree efficiently organizes the data points, making it faster to search for neighbors by narrowing down the relevant dimensions at each step.

BALL TREE ANALOGY

Real-World Analogy: Social Network Friendship Clusters

1. Building the Ball Tree:

- Each user in the social network is represented as a point in a high-dimensional space, with each dimension corresponding to a different interest.
- The Ball Tree constructs hyperspheres (balls) that encompass groups of users who share similar interests.

2. Searching for Neighbors:

- When a user wants to find friends with similar interests, you traverse the Ball Tree by moving toward the hyperspheres that are most relevant to the user's interests.

- At each level, you decide which hypersphere is most likely to contain users with similar interests.

3. *Finding Nearest Neighbors:*

- As the search narrows down, you focus on smaller and more specific hyperspheres, efficiently locating users who are closest in interest to the target user.
- Users within the same hypersphere are likely to be closer in interest, forming a natural cluster of friends.

In the KNN algorithm, the Ball Tree organizes users based on shared interests, making it faster to search for neighbors by narrowing down the relevant hyperspheres at each step.

MATHEMATICAL EXPLANATION OF ALGORITHM

KD-TREE:

1. *Building the KD-Tree:*

- The KD-Tree recursively partitions the data into regions along alternating dimensions.
- At each step, it selects a dimension d and a split value v to divide the data into two subsets: one with values less than v along dimension d , and the other with values greater than or equal to v along dimension d .

$$X_{\text{left}} = \{x \in X \mid x_d < v\}$$

$$X_{\text{right}} = \{x \in X \mid x_d \geq v\}$$

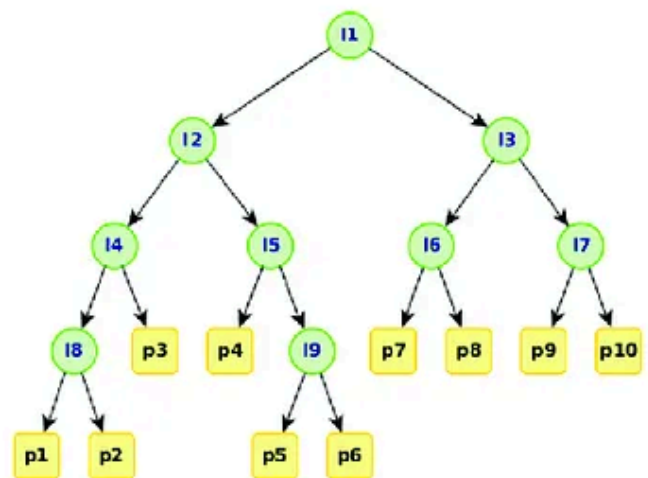
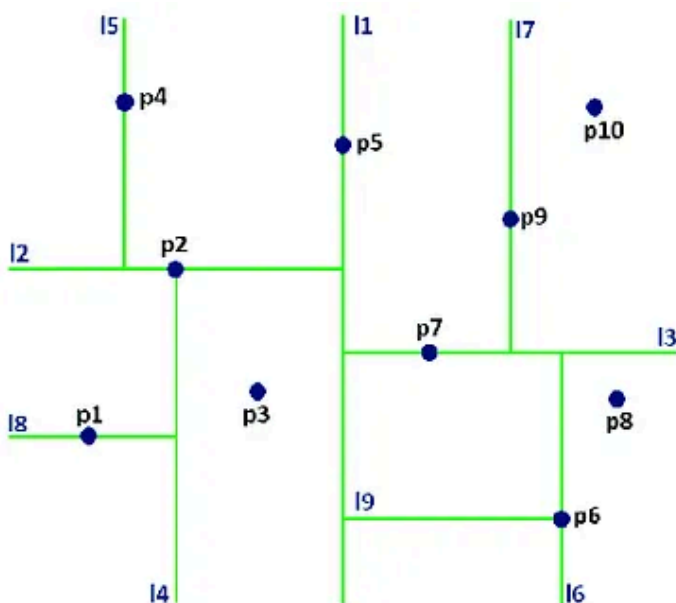
- This process is repeated recursively for each subset until a predefined condition (e.g., a maximum depth or a minimum number of points per leaf node) is met.

2. Searching in the KD-Tree:

- During the search phase, starting at the root, the algorithm traverses down the tree based on the splitting dimension at each level.
- It decides whether to move to the left or right child based on the value along the splitting dimension.

3. Nearest Neighbor Search:

- While traversing, the algorithm keeps track of the best-known distance to a point in the tree and checks whether the other side of the splitting plane could contain a closer point.
- This involves updating the best-known distance and recursively searching the other side of the splitting plane if necessary.



BALL TREE:

1. Building the Ball Tree:

- The Ball Tree represents data points as nodes in a tree, where each node is associated with a hypersphere (ball) that contains a subset of data points.

- In the construction phase, the algorithm selects a “center” and a “radius” for each hypersphere to cover a subset of points.

$$B(\text{center}, \text{radius}) = \{x \in X \mid \text{distance}(x, \text{center}) \leq \text{radius}\}$$

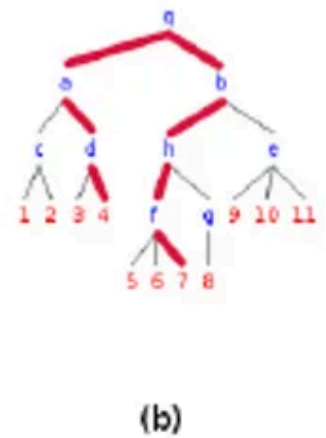
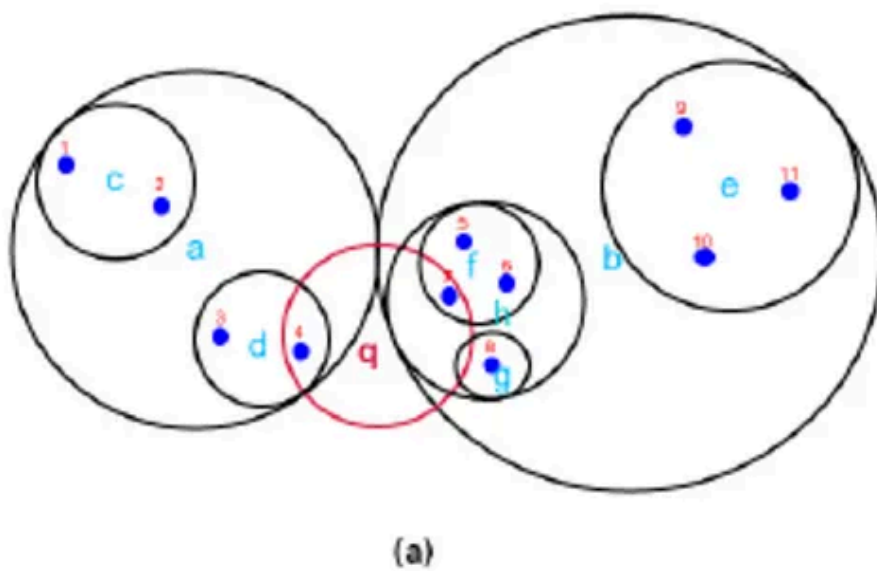
- The data points are divided into two subsets based on the choice of the center and radius.

2. Searching in the Ball Tree:

- During the search phase, starting at the root, the algorithm traverses down the tree based on the distance from the query point to the center of each hypersphere.
- It decides whether to move to the left or right child based on this distance.

3. Nearest Neighbor Search:

- While traversing, the algorithm keeps track of the best-known distance to a point in the tree and checks whether the other hyperspheres could contain a closer point.
- This involves updating the best-known distance and recursively searching the other side of the tree if necessary.



Both KD-Tree and Ball Tree aim to efficiently organize data for nearest neighbor searches in high-dimensional spaces. The choice between them often depends on the characteristics of the data and the specific requirements of the application. KD-Tree focuses on dividing data along dimensions, while Ball Tree organizes data based on proximity within hyperspheres.

KD-TREE

ADVANTAGES :

1. Efficient for Low-Dimensional Data:
 - KD-Trees are particularly effective when the dimensionality of the data is relatively low.
 - In low-dimensional spaces, the tree structure can be constructed and traversed efficiently.
2. Simple Structure:
 - The structure of a KD-Tree is straightforward, making it easy to implement and understand.
 - It is a binary tree where each level corresponds to a different dimension.

3. Balanced Tree:

- When constructed with a good strategy for choosing splitting dimensions, KD-Trees can be relatively balanced, leading to efficient search times.

DISADVANTAGES:

1. Sensitivity to Data Distribution:

- The performance of KD-Trees can degrade when the data is highly skewed or imbalanced.
- If the data has outliers or is unevenly distributed along certain dimensions, the tree may become unbalanced, affecting search efficiency.

2. Inefficient for High-Dimensional Data:

- KD-Trees become less efficient as the dimensionality of the data increases.
- In high-dimensional spaces, the tree structure may become overly deep, leading to slower search times.

3. Sensitive to Noise:

- KD-Trees can be sensitive to noise and outliers in the data, potentially affecting the quality of nearest neighbor searches.

BALL TREE

ADVANTAGES:

1. Effective for High-Dimensional Data:

- Ball Trees can perform well in high-dimensional spaces where KD-Trees may struggle.
- They are less affected by the curse of dimensionality due to the way hyperspheres organize the data.

2. Robust to Outliers:

- Ball Trees are more robust to outliers and noise compared to KD-Trees.
- The use of hyperspheres allows them to encapsulate groups of points more effectively.

3. Adaptable to Different Distance Metrics:

- Ball Trees can be adapted to different distance metrics beyond the Euclidean distance, making them versatile for various applications.

DISADVANTAGES:

1. Complex Structure:

- The structure of a Ball Tree is more complex than that of a KD-Tree, which can make it harder to implement and understand.

2. Slower Construction Time:

- Constructing a Ball Tree can be computationally more expensive than building a KD-Tree.
- The complexity of selecting optimal centers and radii contributes to the slower construction time.

3. Lower Performance in Low-Dimensional Spaces:

- In low-dimensional spaces, Ball Trees may not offer a significant advantage over KD-Trees and might even perform worse.

Conclusion:

In this blog, we learned about two algorithms for finding the nearest neighbors of a data point in a multidimensional space: kd-Tree and Ball Tree. We saw how these algorithms work with real world examples, and compared their advantages and disadvantages. We also discussed the best case scenarios where they should be used. We hope that this blog helped you understand these algorithms better and inspired you to apply them to your own problems.



Written by Narasimharaodevisetti

1 Follower

Follow



More from Narasimharaodevisetti



K-Nearest Neighbor (KNN)

Introduction

Dec 26, 2023



NAIVE BAYES ALGORITHMS

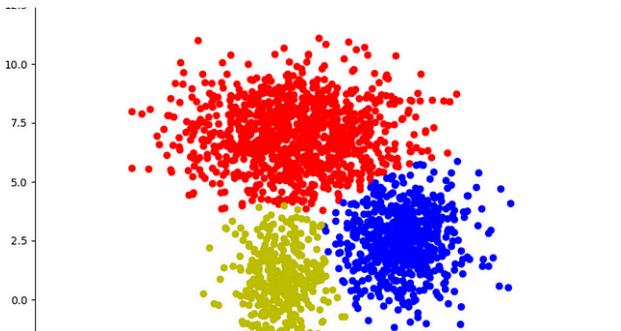
INTRODUCTION

Jan 23



See all from Narasimharaodevisetti


Recommended from Medium

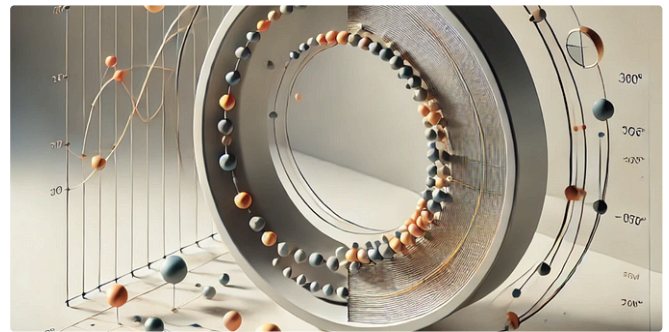



 PointCloud-Slam-I... in Point Cloud Python Matla...

Gaussian Mixture Model (GMM) clustering algorithm and Kmeans...

Target: To divide the sample set into clusters represented by K Gaussian distributions, ea...


★ May 13  1



 Martin Timms

Extending Scikit-Learn to Use Modulo Distance in K-Nearest...

A Guide for Engineers

★ 5d ago  1

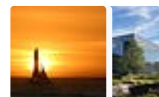


Lists



Staff Picks

755 stories · 1417 saves



Stories to Help You Level-Up at Work

19 stories · 852 saves



Self-Improvement 101

20 stories · 2962 saves



Productivity 101

20 stories · 2507 saves

Source Sentence

That is a happy person

Sentences to compare to

That is a happy dog

That is a very happy person

Today is a sunny day

Add Sentence

Compute

That is a happy dog	0.695
That is a very happy person	0.943
Today is a sunny day	0.257



Emad Dehnavi

Understanding Sentence Similarity in NLP: Top 3 Models You Should...

Sentence Similarity, is a specific task within field of Natural Language Processing (NLP)...



Aug 8



Hassaan Idrees

K-Means vs. DBSCAN: Clustering Algorithms for Grouping Data

A Comprehensive Comparison of Two Popular Clustering Techniques in Machine Learning

Oct 22



$$\mu_B = \frac{1}{m} \sum_{i=1}^m x_i$$
$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^m (x_i - \mu_B)^2$$

Here, x_i are the activations in the mini-batch, μ_B is the mean, σ_B^2 is the variance, and m is the mini-batch size.



Jo Wang

Deep Learning Part 5 -How to prevent overfitting

Techniques used to prevent overfitting in deep learning models:



Jun 29



1



1



Rishabh Singh

KNN (K-Nearest Neighbour)

In the world of machine learning, the K-Nearest Neighbors (KNN) algorithm stands...

Oct 17



1



See more recommendations

