K-Nearest Neighbours (KNN)

KNN is a supervised machine learning algorithm that can be used for both classification and regression tasks. It is a simple yet effective algorithm based on the principle that similar things are grouped together.

→ **How it works:**

1. Training phase → The algorithm stores all the training data points without any learning process.
2. Prediction phase → When a new data point (query point) needs to be classified or predicted →
   (a) It calculates distance between all the training point and query point.
   (b) Find the k-nearest neighbours to the query point.
   (c) If the task is classification, assigns the query point to the most frequent class among the k neighbours.
   (d) If the task is regression, calculate the average of the target values of the k neighbours and assign it to the query point.

* **Distance Metric** : The closeness between data points is measured using a distance metric. The most common metric used is **euclidean distance** which is given by the formula shown below and calculates straight line distance.

$$d(x,y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

Suppose two points in space are (3,4) and (6,3) then distance between them using euclidean distance is—

$$d = \sqrt{(3-6)^2 + (4-3)^2} = \sqrt{9+1} = \sqrt{10} \approx 3.16$$

Manhattan distance → The manhattan distance also known as (L1 distance or taxicab distance) between two points is the sum of the absolute differences of their coordinates. It is calculated using the formula—

$$d_{Manhattan}(p,q) = \sum_{i=1}^{n} |p_i - q_i|$$

where p & q are two points in an n-dimensional space, and p_i and q_i are the coordinates of these points in the i^{th} dimension.

The Manhattan distance measures how far two points are by moving along the gridlines (like a taxi navigating the streets of a city). The distance metric is more appropriate in situations where we can only move along the grid lines rather than in a straight line.

The Manhattan distance between two points (3,4) & (6,3) can be calculated as —

$$d = |6-3| + |3-4| = 3+1 = 4$$

→ It can be used in places where we require grid-based movement (e.g. city blocks, pixel grids in images). In case of high dimensional spaces and sparse data.

Minkowski Distance: Minkowski is a generalization of both Euclidean and Manhattan distances. It is calculated using the formula:

$$d_{Minkowski}(p,q) = \left(\sum_{i=1}^{n} |p_i - q_i|^p\right)^{1/p}$$

where p is the parameter that defines the type of distance. When p=1, Minkowski distance becomes Manhattan distance and when p=2, Minkowski distance becomes Euclidean distance. When p increases, the distance metric becomes more sensitive to large differences between in any single dimension.

→ If we suspect that some features might disproportionately influence the distance, we can choose a higher p to emphasize these differences.

The Minkowski distance between two points (3,4) & (6,3) taking p=3 & p=4, can be calculated as —

For p=3, $d = (|6-3|^3 + |3-4|^3)^{1/3} = (3^3 + 1^3)^{1/3} = 28^{1/3}$

$d \approx 3.03$

For p=4, $d = (|6-3|^4 + |3-4|^4)^{1/4} = (3^4 + 1^4)^{1/4} = (82)^{1/4}$

$d \approx 3$

For p=10, $d = (|6-3|^{10} + |3-4|^{10})^{1/10} = (3^{10} + 1^{10})^{1/10} = (59050)^{1/10}$

≈ 3

* **Choosing the correct value of k:** The choice of k in KNN significantly impacts the model's performance. A small k might lead to overfitting, while a large k can result in underfitting (ignoring local patterns).
(a) Using cross-validation can be a suitable valid method for choosing k.
(b) We can even do a hyperparameter search using different values of k to find the best k.
(c) We can consider square root of the dataset size as a starting point and start by choosing an odd value.
(d) For imbalanced datasets, a smaller k might be more appropriate to avoid the majority class dominating the predictions.
(e) If the data is noisy and contains many outliers, a larger k can help smooth out the noise and make the model more robust.

Pros and Cons of KNN algorithm:

→ Pros: (a) It is simple and easy to understand.
(b) No training phase is required (a lazy learning algorithm).
(c) It can handle high-dimensional data effectively.
→ Cons: (a) Can be computationally expensive for large datasets (because it needs to calculate distance to every data point), especially during prediction.
(b) It is sensitive to the choice of both 'k' and distance metric.
(c) It can be affected by presence of noisy data and outliers.

Applications:

(a) Recommendation Systems: Suggesting similar items based on user preferences.
(b) Image Recognition: For identifying and classifying objects in images
(c) Anomaly detection: Identifying unusual patterns that do not confirm to expected behaviours.

Choosing distance calculation methods: In the sklearn library for KNN, the distance between data points can be calculated using three different methods-
(a) brute-force (b) kd-tree (c) ball-tree.

(a) **Brute-force:** In the brute-force method, distance of one point is calculated with every other point. Suppose our dataset has 'n' samples and 'd' dimensions. Then we have a time complexity of **O(nd)** to query a data point to find its nearest neighbour.

For Eg.

(x_1, y_1)	(x_2, y_2)	(x_3, y_3)	---	---	---	(x_n, y_n)

In this we have 'n' samples with two dimensions (x & y). For point (x_1, y_1) we will first find out distance of (x_1, y_1) to (x_2, y_2), (x_1, y_1) to (x_3, y_3) --- to (x_n, y_n) and thus time to find one nearest neighbour is quite large. If we have large dimensional space this will be a slow process and it will be computationally expensive.

(b) **KD Tree:** KD-tree is a binary-tree based data structure used for efficient querying of k-dimensional points. The basic idea is to recursively split the data into two half-spaces based on the median of one of the dimensions at each step. It should be used for low-dimension -al data (lower than 20).

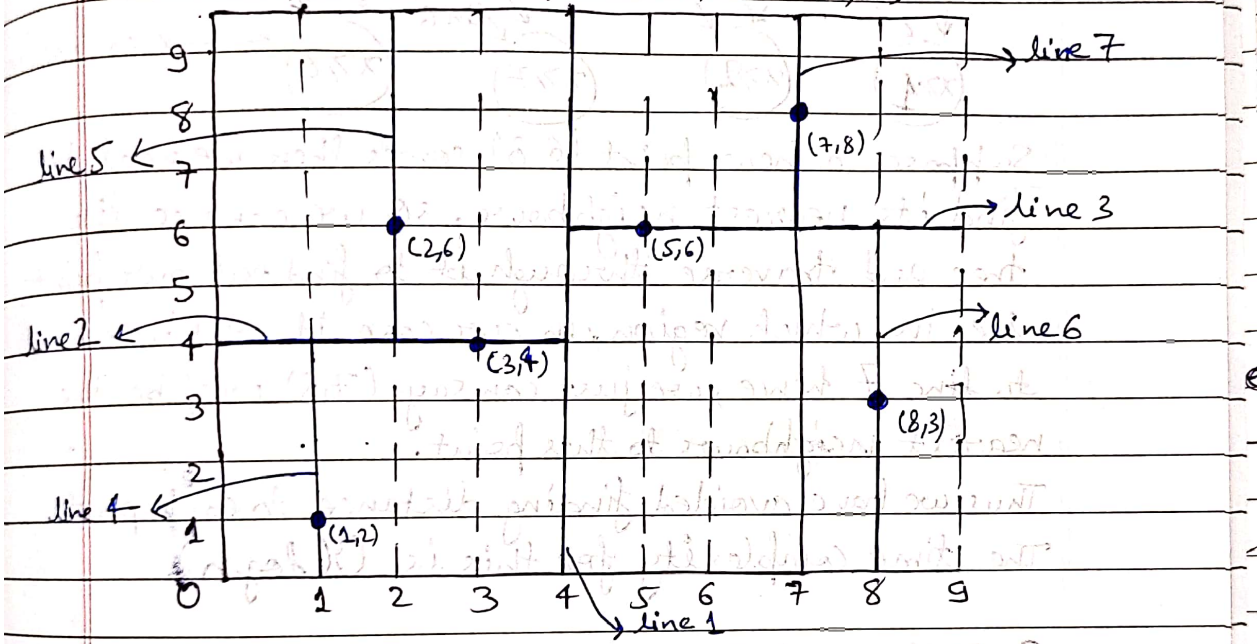
→ How trees are constructed:

1. At each node data is split based on median value along one dimension (starting with first dimension and so on).
2. This splitting creates two sub-trees, with points in left sub-tree having smaller values in that dimension and points in the right subtree having larger values.
3. This process continues recursively until each node contains a small number of points or a single point.

Querying a new point :

1. To find k-nearest neighbours, KD Tree recursively traverses the tree.
2. The search starts at the root, and at each node, it checks which side of the node the query point lies on.
3. The search first explores the subtree containing the query point, but it also checks other subtree if there is a chance that points from that subtree could be closer.

Example: Suppose, we have two dimensional datapoints as —
(1,2) (2,6) (3,4) (5,6) (7,8) (8,3)



Step 1: Using the x-dimension, we find median of all the points on the x-axis i.e., median of
[1, 2, 3, 5, 7, 8] , median = $\frac{3+5}{2} = 4$.
So we can draw a line parallel to y-axis on x=4 (line 1).
This line 1 divides the space into two parts.

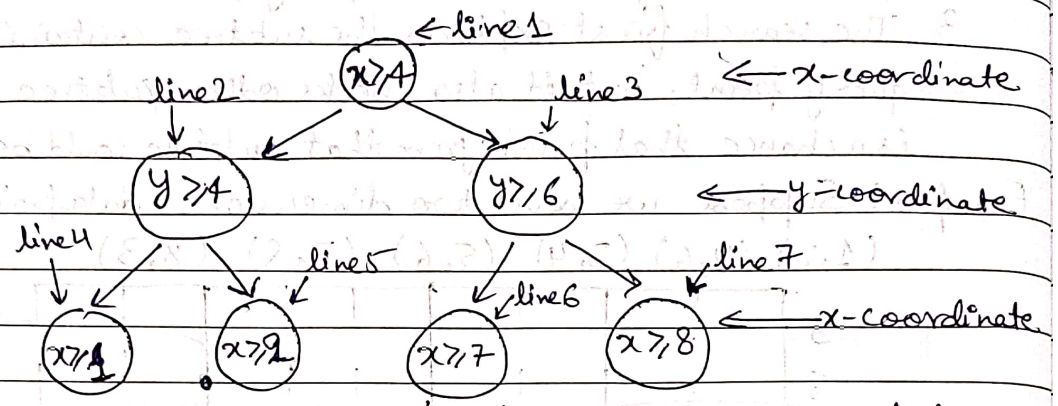
Step 2: In the next step, we take y-dimension points from both left half and right half and find their median.

Left half (1,2) (2,6) (3,4)
median of [2, 4, 6] = 4
so now we draw a line parallel to x-axis at y = 4. (line 2)

Right half (5,6) (7,8) (8,3)
median of [3, 6, 8] = 6
so we draw a line parallel to x-axis at y = 6. (line 3)

These two lines further sub-divide the plane in two halfs.

Step3: We again take points in each half and now take their x-coordinates and find their median to split plane into more such halves.
The KD-tree looks like —



Suppose a new point (6,8) comes then we need to find its nearest neighbour, so we can use the tree and traverse through it to find out our point lies in which region, in our case it will be closest to line 7 hence, we just can say (7,8) will be the nearest neighbour to this point.
Thus, we have avoided finding distance to each point.
The time complexity for this is O(log n).

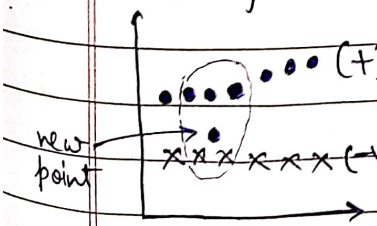
(c) **Ball-tree**: Balltree is another tree-based data structure that improves the efficiency of nearest neighbour search. It's particularly useful for higher dimension data where KD-Tree becomes less efficient.
Instead of splitting the data by planes (as in KD-Tree), BallTree splits the data by hyperspheres.
How it works:

1. At each node, a ball (hypersphere) is created that encompasses all the points assigned to that node.
2. The points are then divided into two subsets, and new balls are created for each subsets. The center of the ball/hypersphere is usually the centroid of the points it has.
3. This process continues recursively until the dataset is partitioned into small enough groups.

Querying a new point:
For querying, the ball tree recursively traverses the tree, and at each node, it checks whether the hypersphere at that node can contain points closer than the currently found nearest point(s).
If the ball can't contain closer points, the search skips that subtree entirely.
The time complexity for using Ball Tree is also $O(\log n)$.

Conclusion:
Both KD-Tree and Ball Tree are powerful data structures that significantly speed up KNN queries by reducing the number of distance calculations required. KD Tree is ideal for low-dimensional data, while ball tree is more effective in higher dimensional spaces.

Weighted KNN: In the traditional KNN algorithm, each of the K nearest neighbours of a query point has an equal vote in determining the predicted label or regression output. However, this can sometimes lead to suboptimal results, especially when some neighbours are closer to the query point than others. Weighted KNN addresses this by assigning different weights to the neighbours based on their distance to the query point, giving closer points more influence.



suppose K=5, then new point will be assigned to (+ve) class even though it is closer to (-ve), so at these places weighted KNN can be helpful.

→ **When to use weighted KNN:**
1) Non-uniform distribution of neighbours.
2) High variance in distances.
3) Imbalanced classes: In cases where classes are imbalanced, the decision boundary between classes can be skewed. Weighted KNN can help mitigate this by allowing closer examples (which are likely to be more relevant) to influence the decision more.