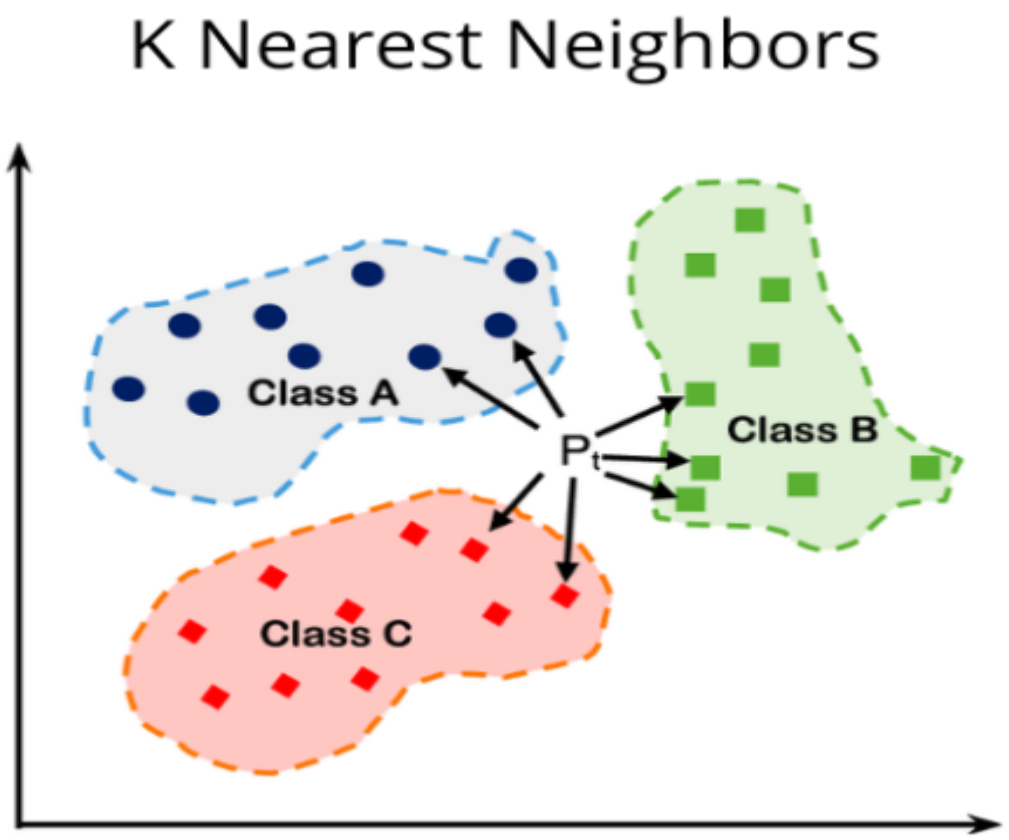


Table Of Contents

(9) K-Nearest neighbour (KNN)

→ KNN is a type of supervised learning algorithm which is used for both Regression & classification, mostly for classification.

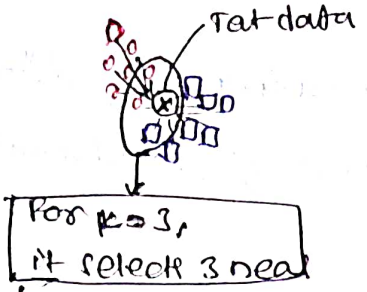
→ Given a dataset with different class, KNN tries to predict the correct class for test data by calculating the distance between the test data and ALL THE TRAINING POINTS!! (ALL)
Each element in test is measured with all the training points.

→ And then select the k points which are closest to the test data.
Hence, 'K' in K-NN is,
"How many closest point are you selecting to predict data?"

→ once the points are selected, the algorithm calculates the probability (for classification) of the test point belonging to the class of the k-training points and the class with the highest probability is selected.

→ In case of Regression Problem,
the predicted value is the mean of the k-selected training points.

Algorithm / visual illustration



KNN algorithm calculates distance b/w the test data and the given training data. And then it will select k points which are nearest.

↳ And it's 2 red, 1 blue. Hence, it is Red'O

↳ If it is Regr probl, the predicted value is mean of those three.

How distance calculated?

(*) Euclidean distance:-

→ It is most commonly used method to calculate the distance b/w two points.

→ The euclidean distance b/w two points, $P(p_1, p_2)$ & $Q(q_1, q_2)$ is calculated as:-

$$d(P,Q) = \sqrt{(q_1-p_1)^2+(q_2-p_2)^2}$$

(*) Hamming distance:-

→ Hamming distance is the distance metric that measures the no. of mismatches b/w two vectors. It is most widely used on cat or categorical data.

→ Generally, if we have features as categorical data then we consider the difference to be 0 if both values are same, difference is 1 if both are different.

Eg: $(A, B, C, D) - (A, D, C, B) = (0, 1, 0, 1) = 2$

(*) manhattan distance

→ Also known as L1 norm, Taxicab norm, Rectilineal dist, city block distance.

→ this distance represents the sum of absolute differences b/w the opposite values of vector.

$$MD(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

→ manhattan distance is less influenced by outliers than the Euclidean distance, worth very high dimensional data is more preferred.

(*) Lazy learners

→ KNN algorithm is often termed as lazy learner.

→ And most of the algorithms like Bayesian classific, logistic Regression & SVM etc. are called Eager learners. these algorithms generalize over the training set before receiving the test data.

i.e, they create the model based on the training data before receiving the test data, and then do the prediction of the data.

→ But this is not the case with KNN algorithm, it doesn't create a generalized model (equation) for the training set but waits for the test data. once the test data is provided then only it starts generalizing the training data to classify the test data. Here, entire training data is the model, waits for test set.

→ So, a lazy learner just stores training data &

(*) weighted nearest neighbour

→ As name suggests we assign weight to the k-nearest neighbour. The weights are typically assigned on the basis of distance.
Sometimes rest of data are assigned '0' also.

→ the main intuition is that the points in neighbor should have more weight than farther points.

(*) choosing the value of k

→ The value of k- affects the KNN classifier drastically

→ the flexibility of model decreases with the increase of k.

→ If 'k' value is lower, then model has high variance & low bias.
As 'k' value is increased, then variance decreases & bias increases.

→ with very low value of k, there is a chance of algorithm overfitting the data,
where as
with very high value of k, there is a chance of 'underfitting'

⊛ Different ways to Perform K-NN

→ the way K-NN classifies the data by calculating the distance of test data from each of the observations and selecting K-values, this approach is known as "Brute Force KNN". (which is computationally very expensive (time taking))

→ So, the idea behind using other algorithms for KNN classifier is to reduce the time during test-period by preprocessing the training data in such a way that the test data can be easily classified in the "appropriate clusters"

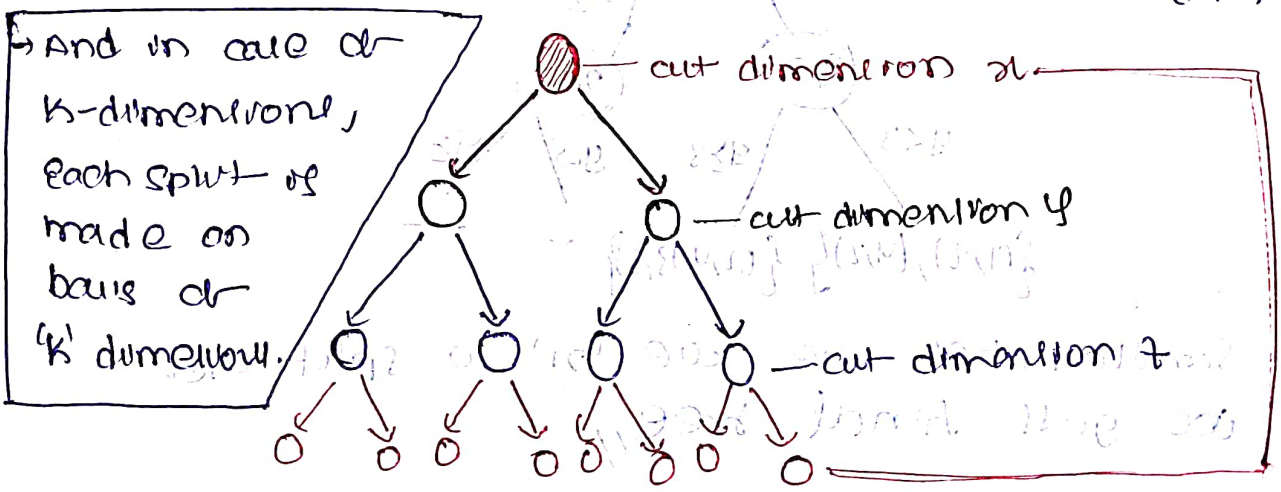
⊛ K-dimensional Tree (Kd Tree)

→ K-d Tree is a hierarchical binary tree.

→ It rearranges the whole dataset in a binary tree structure, so that when test data is provided, it could give out the result by traversing through the tree (every split eliminates half data), which takes less time than brute search.

Eg
Let's say we have 3 dimensional data (x, y, z).

→ And in case of K-dimensions, each split is made on basis of 'K' dimensions.



Eg)

Q) training data = { (1,2), (2,3), (2,4), (3,6), (4,2), (5,7), (6,8), (8,5), (9,1), (9,3) }

Ans

Here, k=2

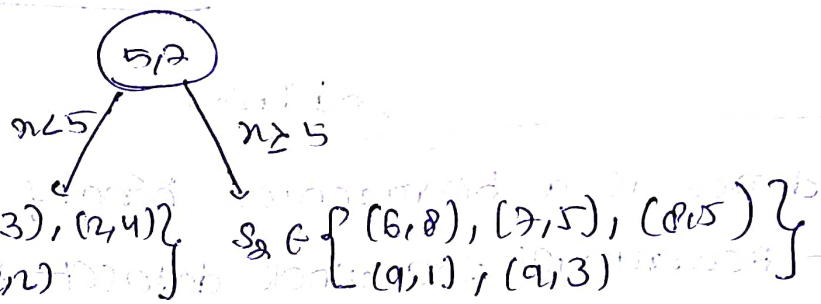
Let's build our kd-tree

(i) Sort data & choose median as split point.

$x \in \{1, 2, 2, 3, 4, (5), 6, 7, 8, 9, 9\}$

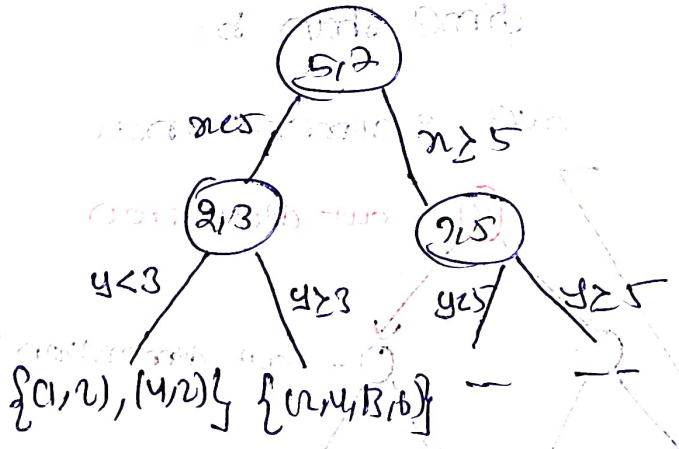
↳ median.

∴ our first node will be (5,7); $x \geq 5$ (split cond.)



(ii) Split S_1, S_2 on coordinate of y.

$y_{S_1} \in \{1, 2, (3), 4, 6\}$ $y_{S_2} \in \{1, 3, (5), 5, 8\}$



Similarly, here we see can't to split and we get final tree.

note (kd-tree) :-

→ once the tree is formed, it is easy for algorithm to search for the probable nearest neighbor just by traveling tree.

→ the main problem of kd tree is that it gives probable nearest neighbors but can miss out actual nearest neighbors.

ii) Ball-Tree

→ similar to kd trees, Ball trees are also hierarchical data structures. these are very efficient specially in case of higher dimensions.

formed by following steps

→ two clusters are created initially.
→ all data points must belong to atleast one of the cluster.
→ one point cannot be in both clusters.
→ distance of point is calculated from centeroid of each cluster. the point closer to centeroid goes into that particular cluster.
→ each cluster is then divided into subclusters again, and then the points are classified into each cluster on the basis of distance from centroid.
→ this is how clusters are kept divided till a certain depth.