k-Nearest Neighbors

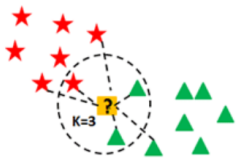
K-nearest neighbors (KNN) is a type of supervised learning algorithm which is used for both regression and classification purposes, but mostly it is used for the later. Given a dataset with different classes, KNN tries to predict the correct class of test data by calculating the distance between the test data and all the training points. It then selects the k points which are closest to the test data. Once the points are selected, the algorithm calculates the probability (in case of classification) of the test point belonging to the classes of the k training points and the class with the highest probability is selected. In the case of a regression problem, the predicted value is the mean of the k selected training points.

Let's understand this with an illustration:

1) Given a training dataset as given below. We have a new test data that we need to assign to one of the two classes.

2) Now, the k-NN algorithm calculates the distance between the test data and the given training data.

3) After calculating the distance, it will select the k training points which are nearest to the test data. Let's assume the value of k is 3 for our example.



4) Now, 3 nearest neighbors are selected, as shown in the figure above. Let's see in which class our test data will be assigned :

Number of Green class values = 2 Number of Red class values = 1 Probability(Green) = 2/3 Probability(Red) = 1/3

Since the probability for Green class is higher than Red, the k-NN algorithm will assign the test data to the Green class.

Similarly, if this were the case of a regression problem, the predicted value for the test data will simply be the mean of all the 3 nearest values.

This is the basic working algorithm for k-NN. Let's understand how the distance is calculated :

Euclidean Distance:

It is the most commonly used method to calculate the distance between two points. The Euclidean distance between two points 'p(p1,p2)' and 'q(q1,q2)' is calculated as :

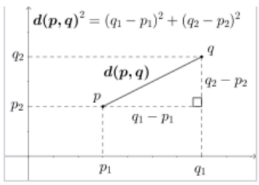


image source : Wikipedia

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

Similarly,for n-dimensional space, the Euclidean distance is given as :

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}.$$

Lazy Learners

k-NN algorithms are often termed as Lazy learners. Let's understand why is that. Most of the algorithms like Bayesian classification, logistic regression, SVM etc., are called Eager learners. These algorithms generalize over the training set before receiving the test data i.e. they create a model based on the training data before receiving the test data and then do the prediction/classification on the test data. But this is not the case with the k-NN algorithm. It doesn't create a generalized model for the training set but waits for the test data. Once test data is provided then only it starts generalizing the training data to classify the test data. So, a lazy learner just stores the training data and waits for the test set. Such algorithms work less while training and more while classifying a given test dataset.

Pros and Cons of k-NN Algorithm

Pros:

- It can be used for both regression and classification problems.
- It is very simple and easy to implement.
- Mathematics behind the algorithm is easy to understand.
- There is no need to create model or do hyperparameter tuning.
- KNN doesn't make any assumption for the distribution of the given data.
- There is not much time cost in training phase.

Cons:

- Finding the optimum value of 'k'
- It takes a lot of time to compute the distance between each test sample and all training samples.
- Since the model is not saved beforehand in this algorithm (lazy learner), so every time one predicts a test value, it follows the same steps again and again.
- Since, we need to store the whole training set for every test set, it requires a lot of space.
- It is not suitable for high dimensional data.
- Expensive in testing phase

Different ways to perform k-NN

Above we studied 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 also known as "Brute Force k-NN". This is computionally very expensive. So, there are other ways as well to perfrom k-NN which are comparatively less expensive than Brute force approach. The idea behind using other algorithms for k-NN 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.

Let's discuss and understand the two most famous algorithms:

k-Dimensional Tree (kd tree)

k-d tree is a hierarchical binary tree. When this algorithm is used for k-NN classficaition, it rearranges the whole dataset in a binary tree structure, so that when test data is provided, it would give out the result by traversing through the tree, which takes less time than brute search.

The dataset is divided like a tree as shown in the above figure. Say we have 3 dimensional data i.e. (x,y,z) then the tree is formed with root node being one of the dimensions, here we start with 'x'. Then on the next level the split is done on basis of the second dimension, 'y' in our case. Similarly, third level with 3rd dimension and so on. And in case of 'k' dimensions, each split is made on basis of 'k' dimensions. Let's understand how k-d trees are formed with an example:

Training Data ⇒ {(1,2), (2,3), (2,4), (3,6), (4,2), (5,7), (6,8), (7,5), (8,5), (9,1), (9,3)}

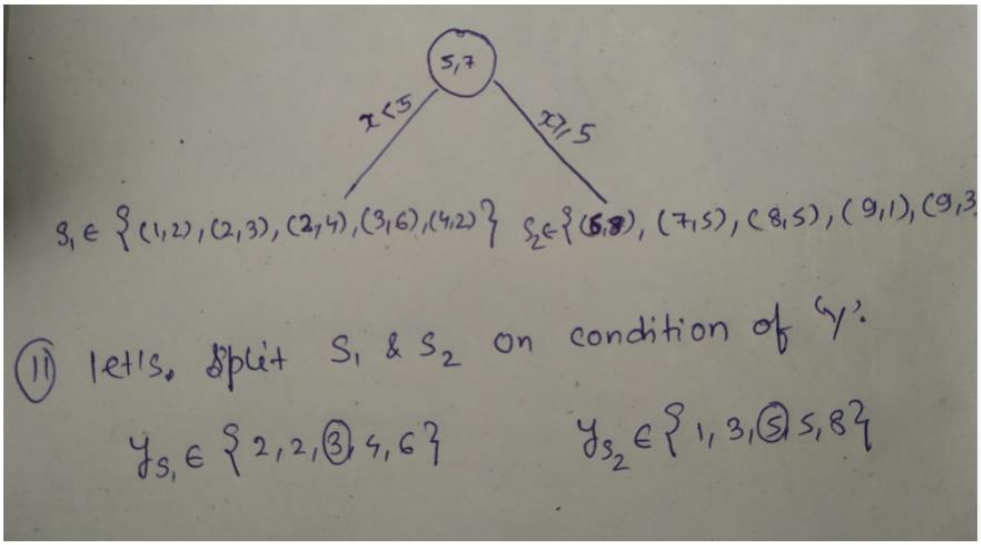
here, K = 2

let's build our 2-d tree

let's sort our data and choose the median to be the split point :-

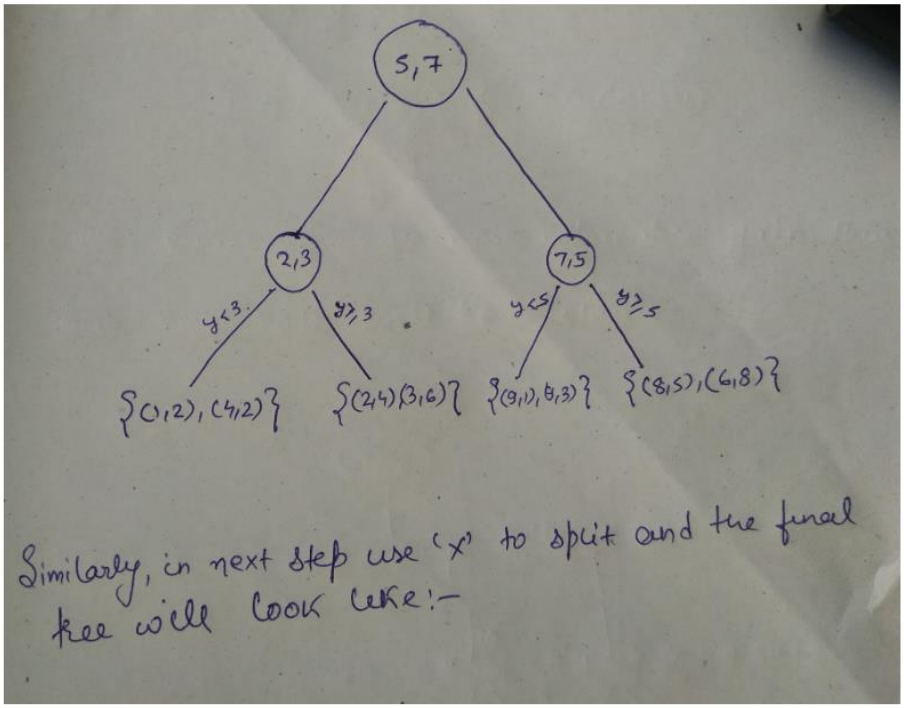
$x \in \{1, 2, 2, 3, 4, (5), 6, 7, 8, 9, 9\}$
↳ median

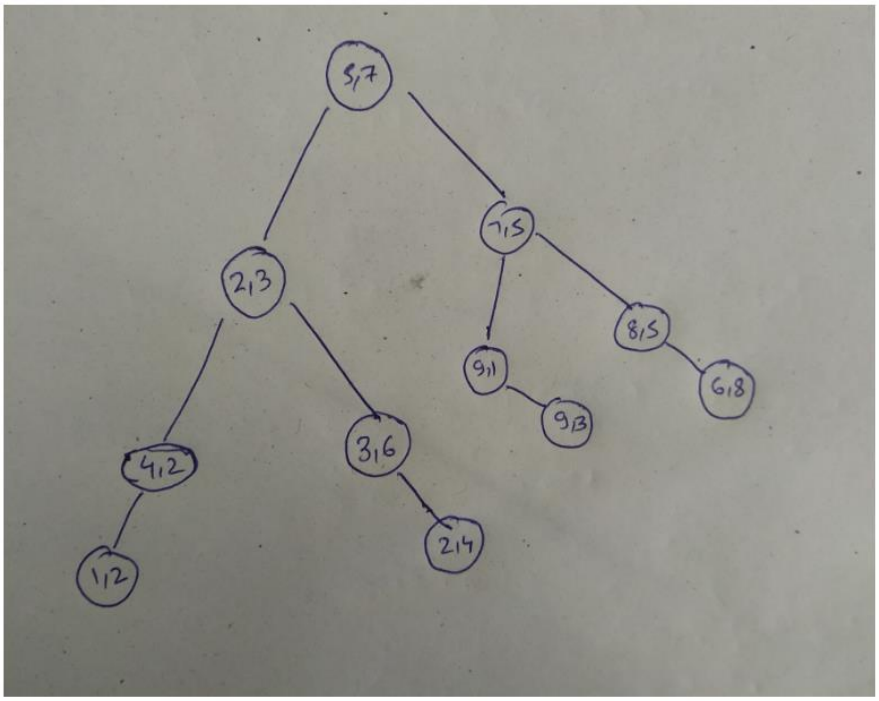
(i) our first node will be, (5,7), $x > 5$ → split condition



(ii) let's split S_1 & S_2 on condition of 'y':

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





Once the tree is formed , it is easy for algorithm to search for the probable nearest neighbor just by traversing the tree. The main problem k-d trees is that it gives probable nearest neighbors but can miss out actual nearest neighbors.