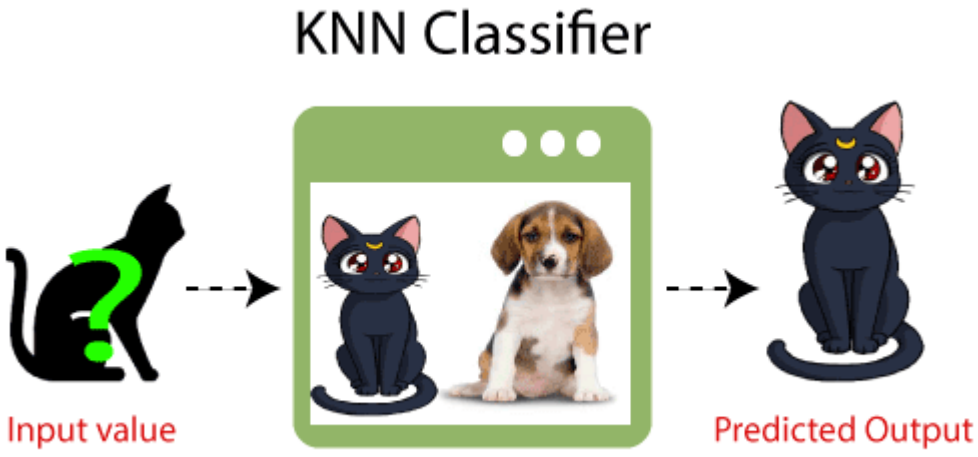
KNN

1) Explain the working of KNN algorithm.

K-Nearest Neighbor(KNN) Algorithm for Machine Learning

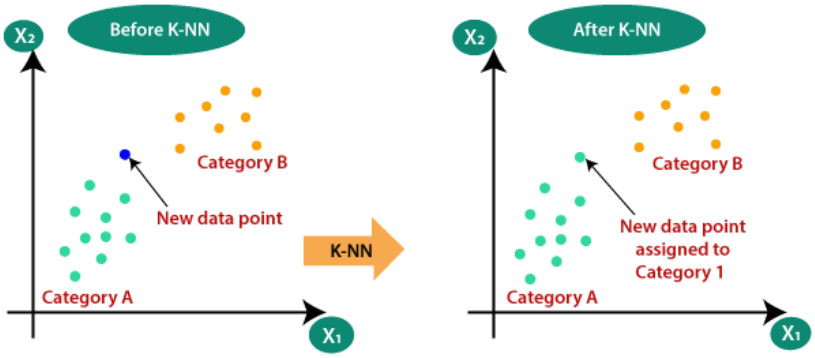
← prev next →

- K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.
- K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- **Example:** Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.



Why do we need a K-NN Algorithm?

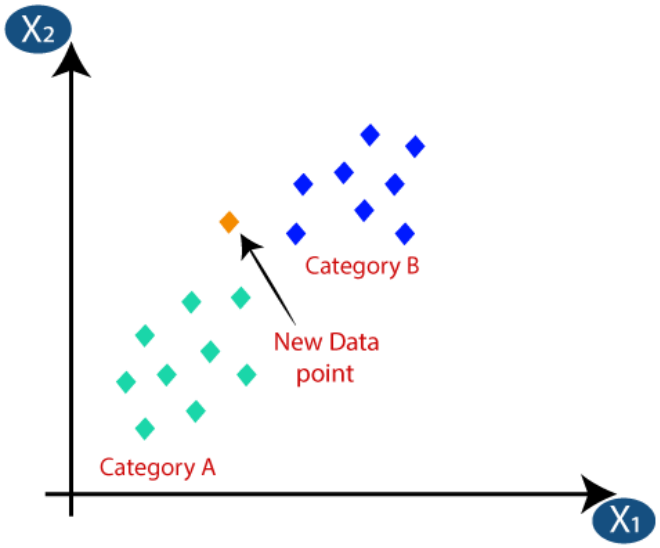
Suppose there are two categories, i.e., Category A and Category B, and we have a new data point x1, so this data point will lie in which of these categories. To solve this type of problem, we need a K-NN algorithm. With the help of K-NN, we can easily identify the category or class of a particular dataset. Consider the below diagram:



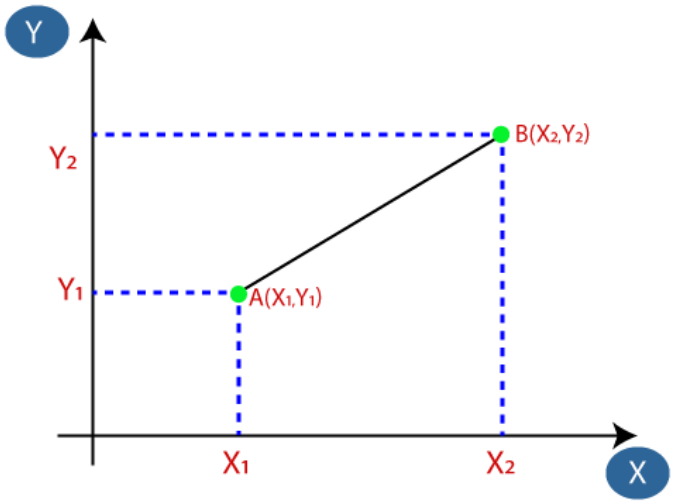
How does K-NN work?

The K-NN working can be explained on the basis of the below algorithm:

- **Step-1:** Select the number K of the neighbors
- **Step-2:** Calculate the Euclidean distance of **K number of neighbors**
- **Step-3:** Take the K nearest neighbors as per the calculated Euclidean distance.
- **Step-4:** Among these k neighbors, count the number of the data points in each category.
- **Step-5:** Assign the new data points to that category for which the number of the neighbor is maximum.
- **Step-6:** Our model is ready.

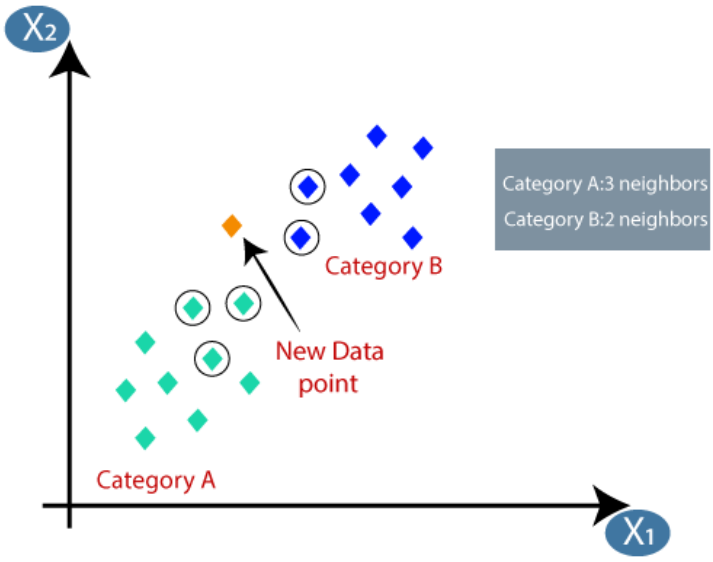


- Firstly, we will choose the number of neighbors, so we will choose the k=5.
- Next, we will calculate the **Euclidean distance** between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



Euclidean Distance between A_1 and B_2 = $\sqrt{(X_2-X_1)^2+(Y_2-Y_1)^2}$

- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to select the value of K in the K-NN Algorithm?

Below are some points to remember while selecting the value of K in the K-NN algorithm:

- There is no particular way to determine the best value for "K", so we need to try some values to find the best out of them. The most preferred value for K is 5.
- A very low value for K such as K=1 or K=2, can be noisy and lead to the effects of outliers in the model.
- Large values for K are good, but it may find some difficulties.

Advantages of KNN Algorithm:

- It is simple to implement.
- It is robust to the noisy training data
- It can be more effective if the training data is large.

Disadvantages of KNN Algorithm:

- Always needs to determine the value of K which may be complex some time.
- The computation cost is high because of calculating the distance between the data points for all the training samples.

2) How KNN predicts the output for regression and classification problems?

KNN regression tries to predict the value of the output variable by using a local average.

KNN classification attempts to predict the class to which the output variable belong by computing the local probability.

3) What are the different distances used in KNN? How are they calculated?

Minkowski Distance:

Manhattan Distance:

Euclidean Distance:

Cosine Distance:

Mahalanobis Distance:

Distance functions

Euclidean $$\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$$

Manhattan $$\sum_{i=1}^{k}|x_i - y_i|$$

Minkowski $$\left(\sum_{i=1}^{k}\left(|x_i - y_i|\right)^q\right)^{1/q}$$

d 1 when there is a mixture of numerical and categoric

Hamming Distance

$$D_H = \sum_{i=1}^{k}|x_i - y_i|$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

X	Y	Distance
Male	Male	0
Male	Female	1

s best done by first inspecting the data. In general, a la

4) What are Lazy Learners? Why KNN is called a lazy learner?

The nearest neighbor algorithms, for example, K-Nearest Neighbors (K-NN) are very "simple" algorithms, but that's not why they are called lazy ;). K-NN is a lazy learner because it doesn't learn a discriminative function from the training data but memorizes the training dataset instead. For example, the logistic regression algorithm learns the model weights during training time. In contrast, there is no training time in K-NN. On the other hand, the "prediction" step in K-NN is relatively expensive: Each time you want to make a prediction, you are searching for the nearest neighbor in the entire training set (note that there are tricks such as BallTrees and KDtrees to speed this up a bit.

To summarize: An eager learner has a model fitting or training step. A lazy learner does not have a training phase.

The nearest neighbour classifier works as follows. Given a new data point whose class label is unknown, we identify the k nearest neighbours of the new data point that exist in the labeled dataset (using some distance function). We then check the class labels of these k nearest neighbours and select the most frequently occurring one (just within the k nearest neighbours, not in the entire dataset). This is the prediction for the class label of the new data point.

For example, consider the following dataset, where income is a numeric feature variable and loan is a binary class label, denoting whether a particular customer with a particular income was able to pay off his/her loan:

income loan 30 n 32 n 35 y 35 n 40 y 42 n 43 y 44 y 50 y Question: Using Euclidean distance, what prediction will 1-nn and 3-nn give for a new data point with income=39.5?

Answer: 1nn = one nearest neighbour. The closest point to 39.5 is the one with income=40 and loan=yes, so predict loan=yes. 3nn = 3 nearest neighbours. The three closest points to 39.5 are those with income=40, 42 and 43. Two of them have loan=yes, one has loan=no, so take the majority vote and predict loan=yes.

Notes about the nearest neighbour classifier: - It is often called “lazy learning”. It does not build any “model”. Instead, all the work (i.e., finding the nearest neighbours) is done at prediction time. - How to choose a good value of k for a particular dataset/application? Try several different values and compute the error rate using cross-validation.

5) How do we select the value of k? How bias and variance varies with k?

The choice of k=10 is somewhat arbitrary. Here's how I decide k:

first of all, in order to lower the variance of the CV result, you can and should repeat/iterate the CV with new random splits. This makes the argument of high k => more computation time largely irrelevant, as you anyways want to calculate many models. I tend to think mainly of the total number of models calculated (in analogy to bootstrapping). So I may decide for 100 x 10-fold CV or 200 x 5-fold CV.

@ogrisel already explained that usually large k mean less (pessimistic) bias. (Some exceptions are known particularly for k=n, i.e. leave-one-out).

If possible, I use a k that is a divisor of the sample size, or the size of the groups in the sample that should be stratified.

Too large k mean that only a low number of sample combinations is possible, thus limiting the number of iterations that are different.

For leave-one-out: (n1)=n=k different model/test sample combinations are possible. Iterations don't make sense at all. E.g. n=20 and k=10: (n=202)=190=19·k different model/test sample combinations exist. You may consider going through all possible combinations here as 19 iterations of k-fold CV or a total of 190 models is not very much. These thoughts have more weight with small sample sizes. With more samples available k doesn't matter very much. The possible number of combinations soon becomes large enough so the (say) 100 iterations of 10-

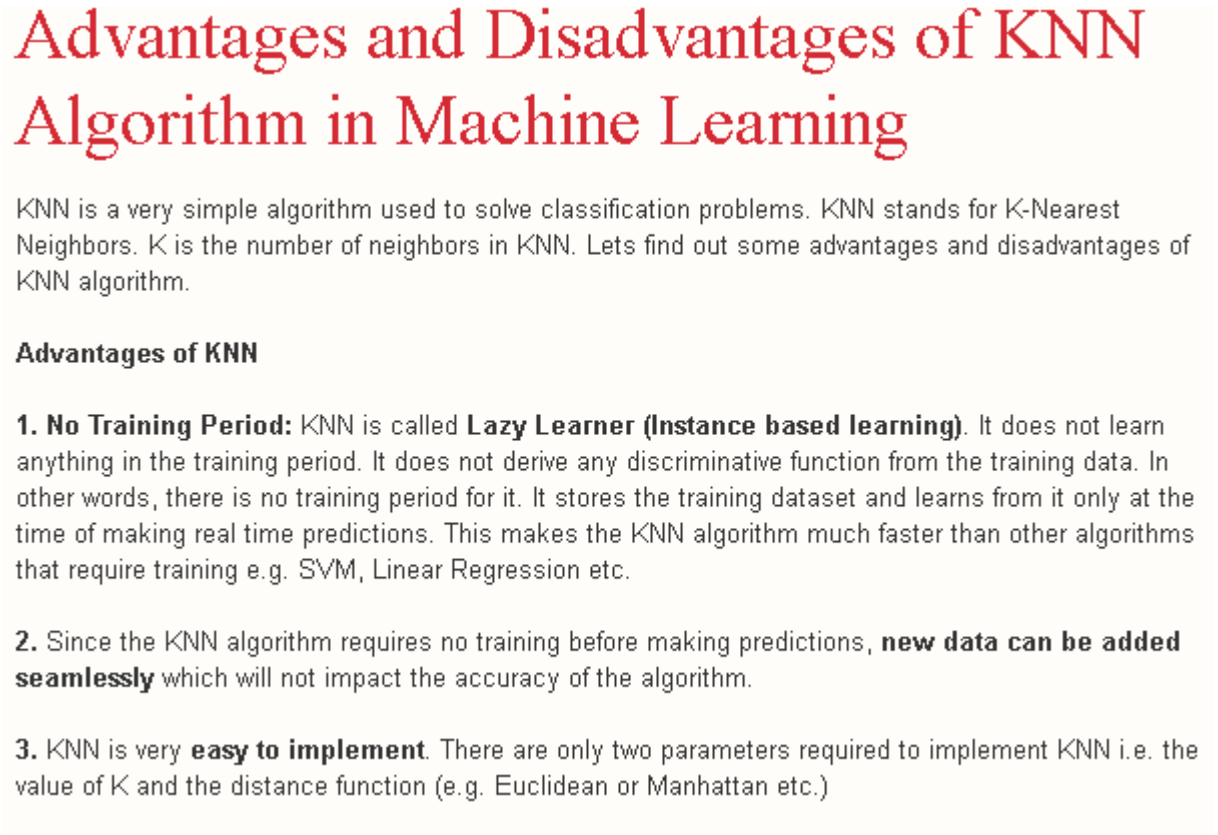
fold CV do not run a great risk of being duplicates. Also, more training samples usually means that you are at a flatter part of the learning curve, so the difference between the surrogate models and the "real" model trained on all n samples becomes negligible.

6) What are advantages and disadvantages of KNN?

k-NN classification: Given a data point in the test set and a training set for which class labels are given, find the k nearest data points in the training set and target label is computed as the mode of the class label of the k nearest neighbours.

Advantages : Simple to implement. Can learn non-linear boundary, robust to noise in the input data.

Disadvantages: Inefficient since the entire training data is processed for every prediction. Time complexity is O(dMNlog(k)) where d is the dimension of the data M the size of training data and N the size of test data.



Disadvantages of KNN

1. Does not work well with large dataset: In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.

2. Does not work well with high dimensions: The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.

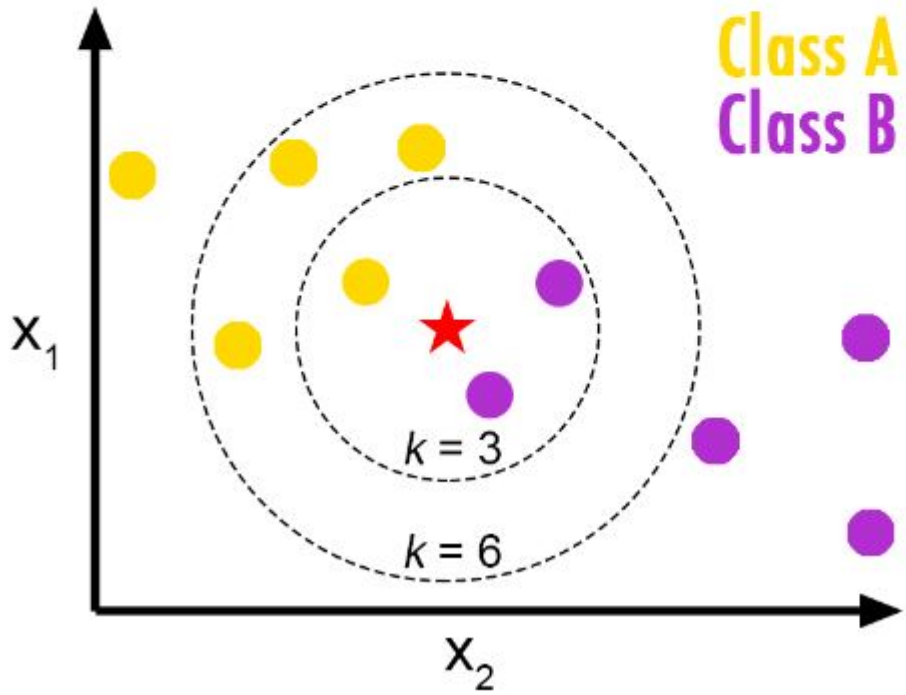
3. Need feature scaling: We need to do feature scaling (standardization and normalization) before applying KNN algorithm to any dataset. If we don't do so, KNN may generate wrong predictions.

4. Sensitive to noisy data, missing values and outliers: KNN is sensitive to noise in the dataset. We need to manually impute missing values and remove outliers.

Pros and Cons of K-Nearest Neighbors

By **Genesis** - September 25, 2018

Share



K- Nearest Neighbors or also known as K-NN belong to the family of supervised machine learning algorithms which means we use labeled (Target Variable) dataset to predict the class of new data point. The K-NN algorithm is a robust classifier which is often used as a benchmark for more complex classifiers such as Artificial Neural Network (ANN) or Support vector machine (SVM).

Below is the list of few of the reasons to choose K-NN machine learning algorithm:

1. **K-NN is pretty intuitive and simple**: K-NN algorithm is very simple to understand and equally easy to implement. To classify the new data point K-NN algorithm reads through whole dataset to find out K nearest neighbors.
2. **K-NN has no assumptions**: K-NN is a non-parametric algorithm which means there are assumptions to be met to implement K-NN. Parametric models like linear regression has lots of assumptions to be met by data before it can be implemented which is not the case with K-NN.
3. **No Training Step:** K-NN does not explicitly build any model, it simply tags the new data entry based learning from historical data. New data entry would be tagged with majority class in the nearest neighbor.
4. **It constantly evolves**: Given it's an instance-based learning; k-NN is a memory-based approach. The classifier immediately adapts as we collect new training data. It allows the algorithm to respond quickly to changes in the input during real-time use.
5. **Very easy to implement for multi-class problem:** Most of the classifier algorithms are easy to implement for binary problems and needs effort to implement for multi class whereas K-NN adjust to multi class without any extra efforts.
6. **Can be used both for Classification and Regression:** One of the biggest advantages of K-NN is that K-NN can be used both for classification and regression problems.
7. **One Hyper Parameter:** K-NN might take some time while selecting the first hyper parameter but after that rest of the parameters are aligned to it.
8. **Variety of distance criteria to be choose from:** K-NN algorithm gives user the flexibility to choose distance while building K-NN model.
    1. Euclidean Distance
    2. Hamming Distance
    3. Manhattan Distance
    4. Minkowski Distance

Even though K-NN has several advantages but there are certain very important disadvantages or constraints of K-NN. Below are listed few cons of K-NN.

1. **K-NN slow algorithm**: K-NN might be very easy to implement but as dataset grows efficiency or speed of algorithm declines very fast.
2. **Curse of Dimensionality:** KNN works well with small number of input variables but as the numbers of variables grow K-NN algorithm struggles to predict the output of new data point.

3. **K-NN needs homogeneous features**: If you decide to build k-NN using a common distance, like Euclidean or Manhattan distances, it is completely necessary that features have the same scale, since absolute differences in features weight the same, i.e., a given distance in feature 1 must means the same for feature 2.
4. **Optimal number of neighbors**: One of the biggest issues with K-NN is to choose the optimal number of neighbors to be consider while classifying the new data entry.
5. **Imbalanced data causes problems**: k-NN doesn't perform well on imbalanced data. If we consider two classes, A and B, and the majority of the training data is labeled as A, then the model will ultimately give a lot of preference to A. This might result in getting the less common class B wrongly classified.
6. **Outlier sensitivity:** K-NN algorithm is very sensitive to outliers as it simply chose the neighbors based on distance criteria.
7. **Missing Value treatment:** K-NN inherently has no capability of dealing with missing value problem.

Some pros and cons of KNN

Pros:

- No assumptions about data — useful, for example, for nonlinear data
- Simple algorithm — to explain and understand/interpret
- High accuracy (relatively) — it is pretty high but not competitive in comparison to better supervised learning models
- Versatile — useful for classification or regression

Cons:

- Computationally expensive — because the algorithm stores all of the training data
- High memory requirement
- Stores all (or almost all) of the training data
- Prediction stage might be slow (with big N)
- Sensitive to irrelevant features and the scale of the data

Pros:

- *Very easy to understand and implement.* A *k*-NN implementation does not require much code and can be a quick and simple way to begin machine learning datasets.
- *Does not assume any probability distributions on the input data.* This can come in handy for inputs where the probability distribution is unknown and is therefore robust.
- *Can quickly respond to changes in input.* *k*-NN employs lazy learning, which generalizes during testing--this allows it to change during real-time use.

Cons:

- *Sensitive to localized data.* Since *k*-NN gets all of its information from the input's neighbors, localized anomalies affect outcomes significantly, rather than for an algorithm that uses a generalized view of the data.
- *Computation time.* Lazy learning requires that most of *k*-NN's computation be done during testing, rather than during training. This can be an issue for large datasets.
- *Normalization.* If one type of category occurs much more than another, classifying an input will be more biased towards that one category (since it is more likely to be neighbors with the input). This can be mitigated by applying a lower weight to more common categories and a higher weight to less common categories; however, this can still cause errors near decision boundaries.
- *Dimensions.* In the case of many dimensions, inputs can commonly be "close" to many data points. This reduces the effectiveness of *k*-NN, since the algorithm relies on a correlation between closeness and similarity. One workaround for this issue is dimension reduction, which reduces the number of working variable dimensions (but can lose variable trends in the process).

7) Discuss kDTree algorithm used for KNN.

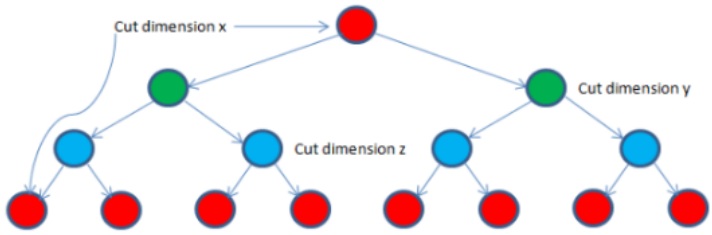
A Kd-tree is just like a normal b-tree but alternating the dimension to be used at each level of the tree. If you have 3 dimensions you can think as a B-tree in X then a B-tree in Y, then in Z, then again in X, etc.

You can and should use any KD-tree library for the language you are using. There's no big advantage coding your own KD-tree except learning of course. Kd-Trees usually perform well only in very small dimensional spaces, if you have many dimensions then a Kd-Tree will be similar to a brute-force linear search. In such cases if performance is critical approximations such as LSH can be used but you always have to compare LSH+KNN against another algorithm that doesn't need an approximation.

8) Discuss Ball Tree algorithm used for KNN.

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

In [ ]: