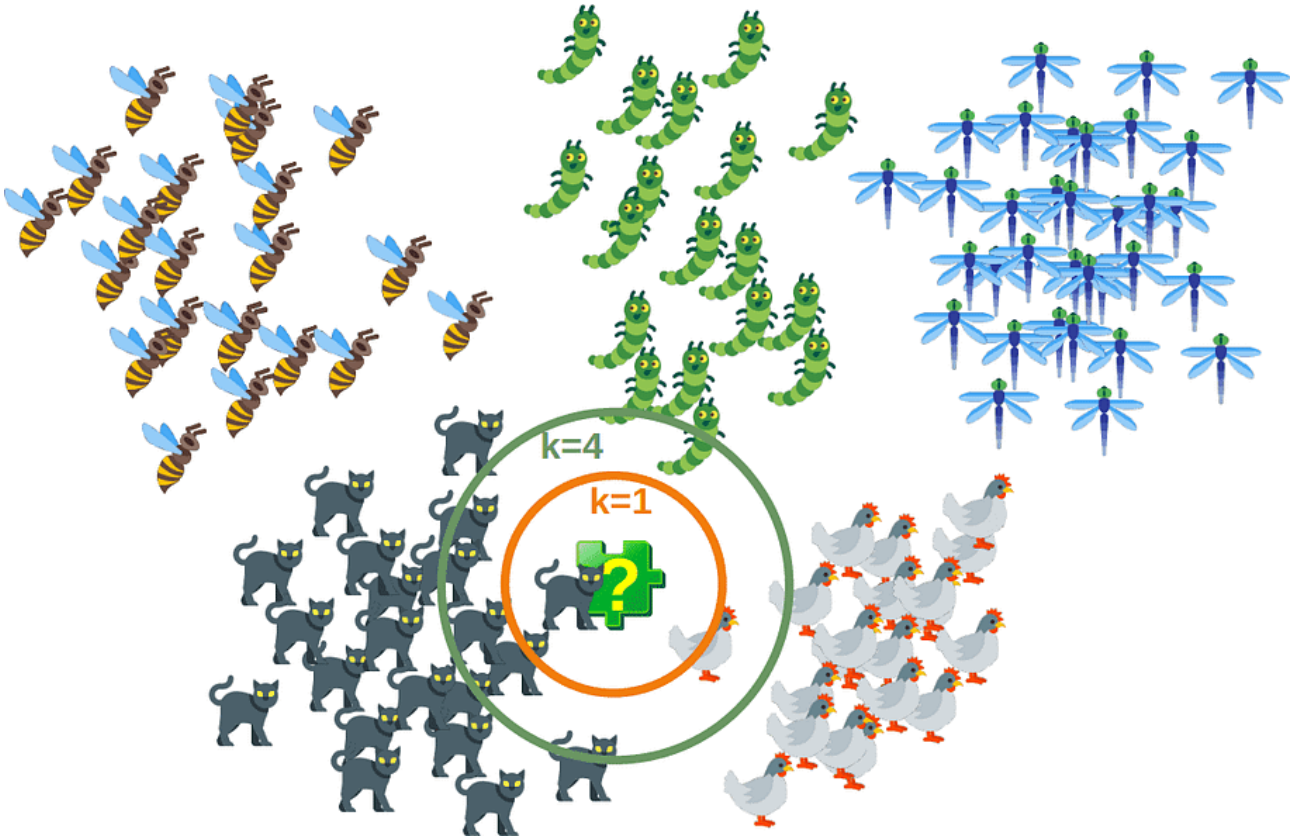
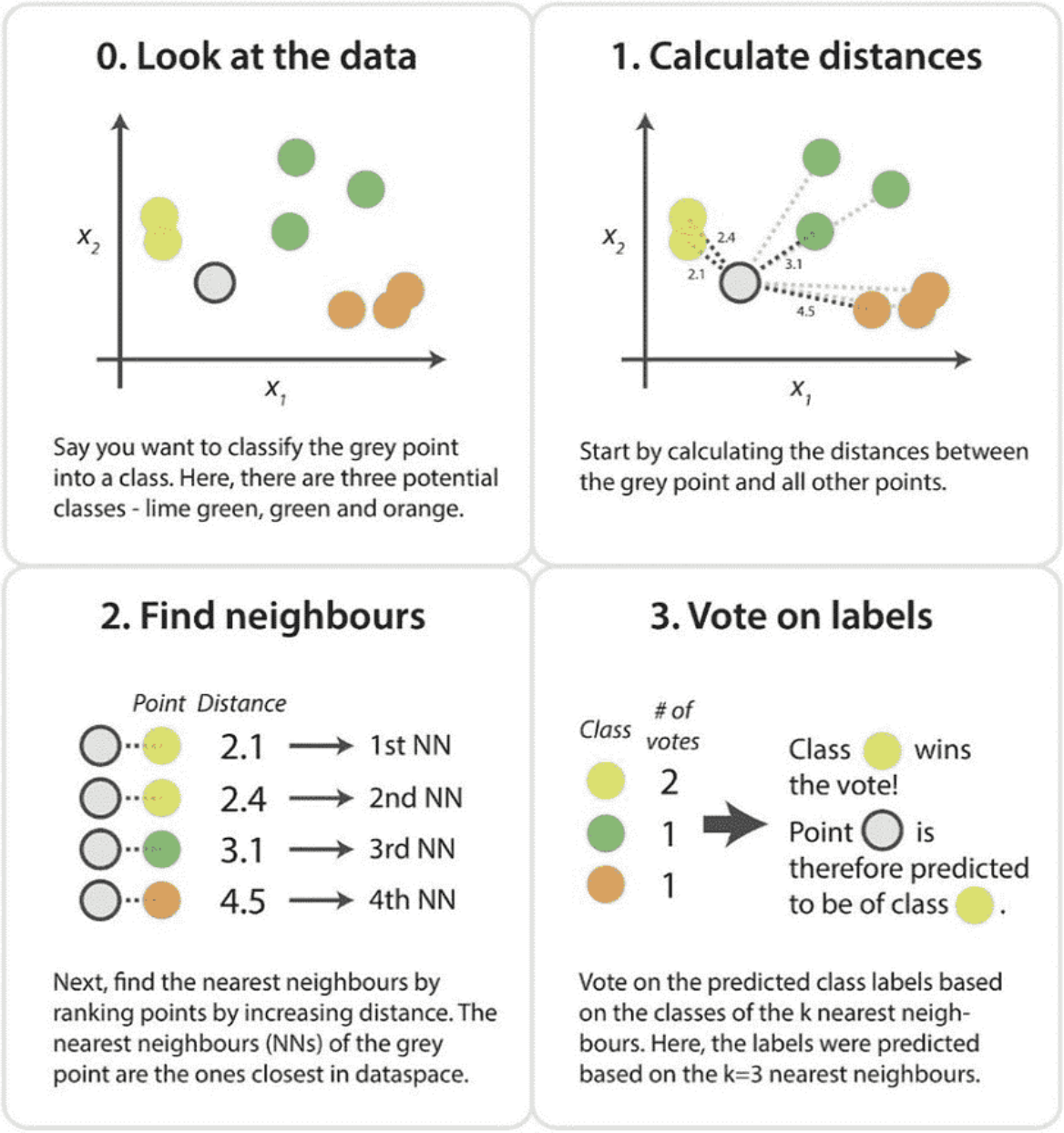
K-Nearest Neighbor



Introduction

K-nearest neighbors (KNN) is a type of supervised learning algorithm used for both regression and classification. KNN tries to predict the correct class for the test data by calculating the distance between the test data and all the training points. Then select the K number of points which is closet to the test data. The KNN algorithm calculates the probability of the test data belonging to the classes of 'K' training data and class holds the highest probability will be selected. In the case of regression, the value is the mean of the 'K' selected training points.

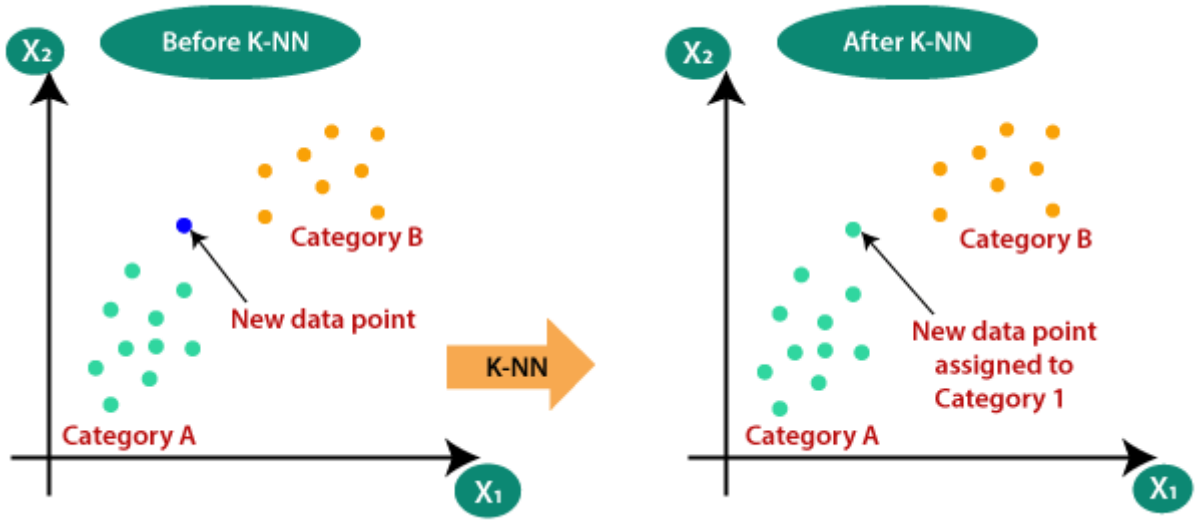
Let see the below example to make it a better understanding



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:

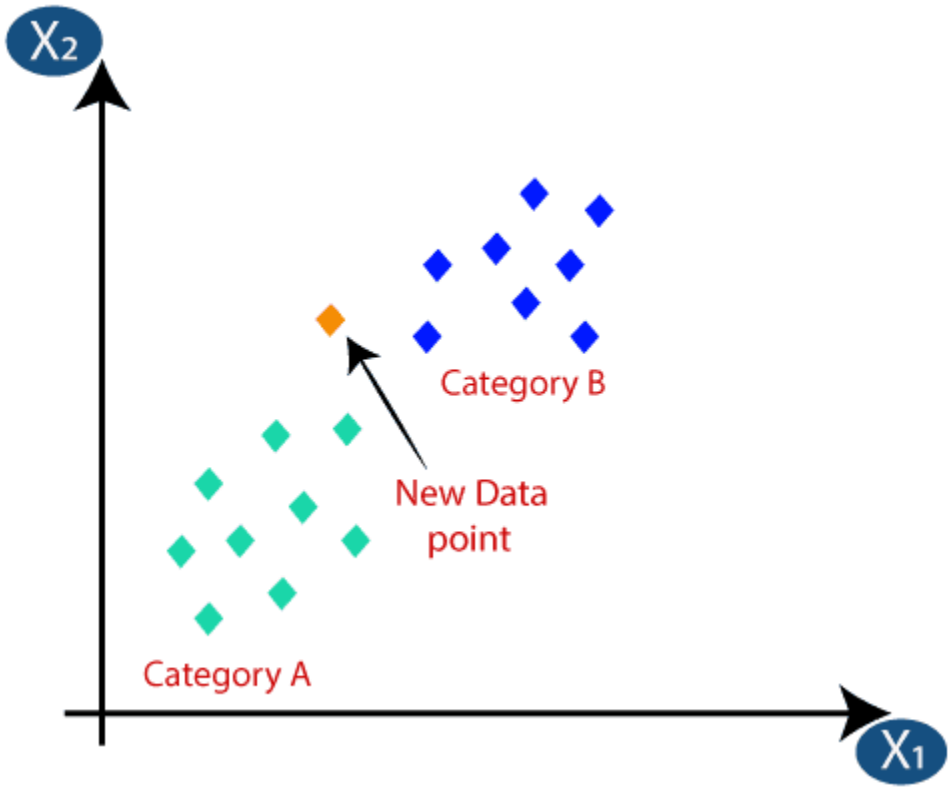


How does K-NN work?

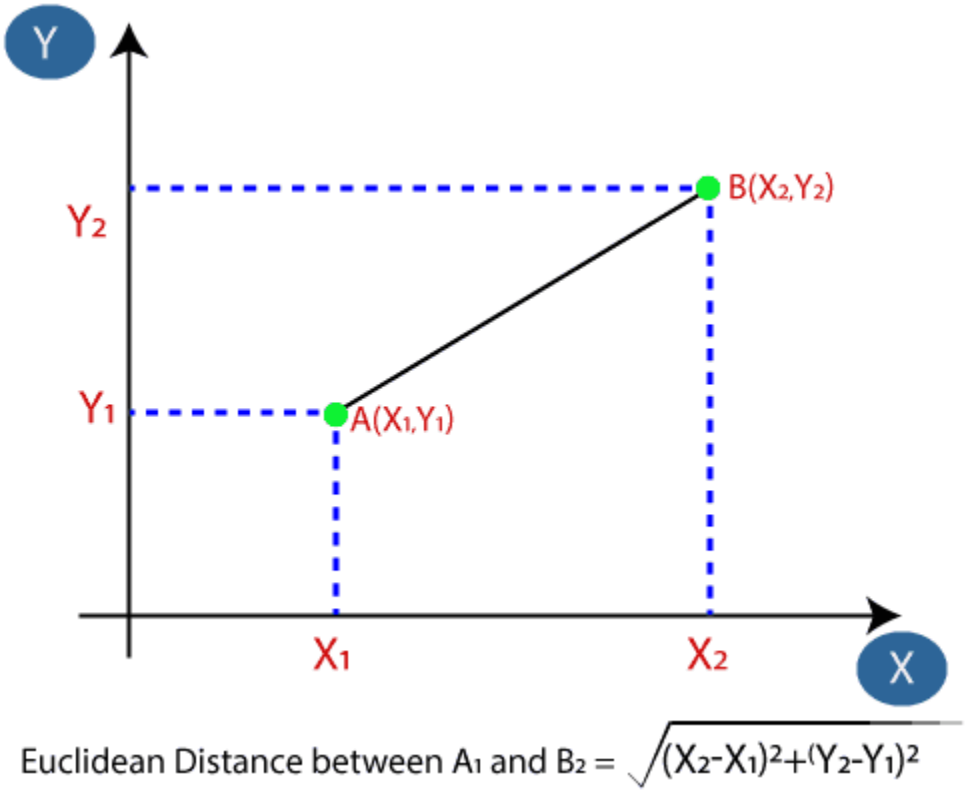
The K-NN working can be explained on the basis of the below algorithm:

- 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.

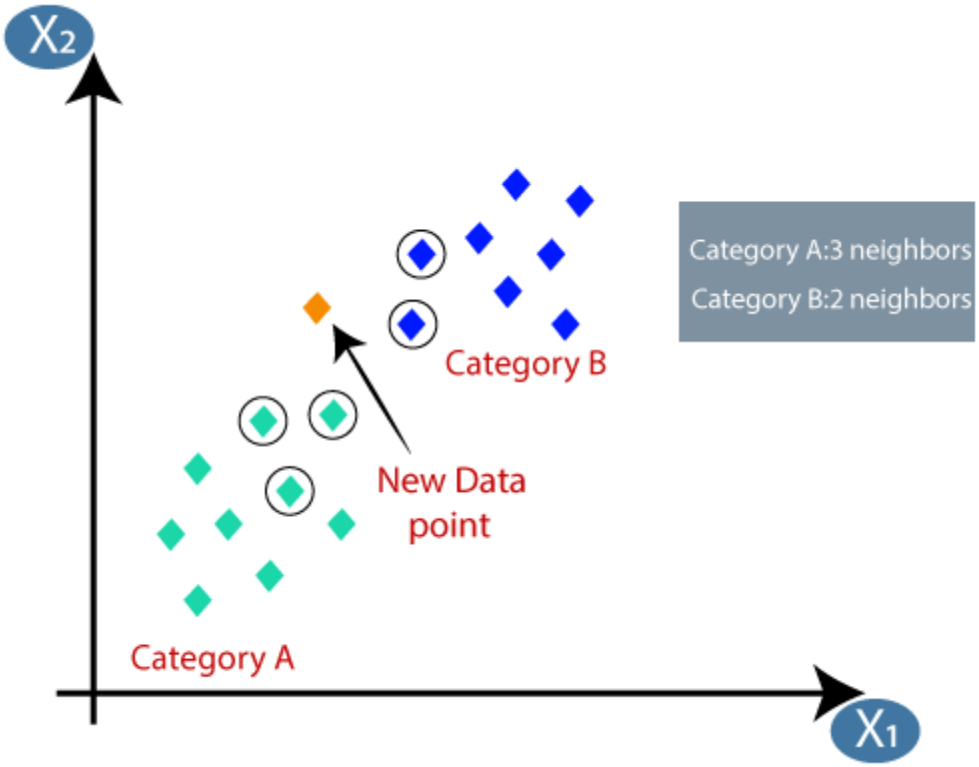
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- 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:

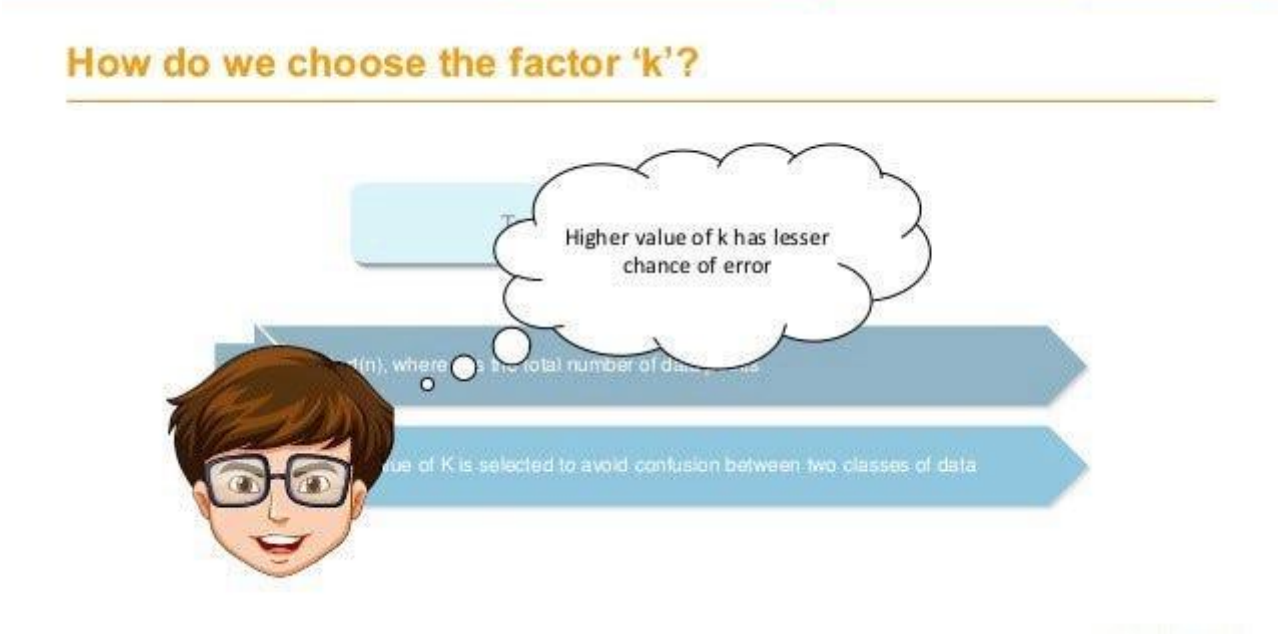


- By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:

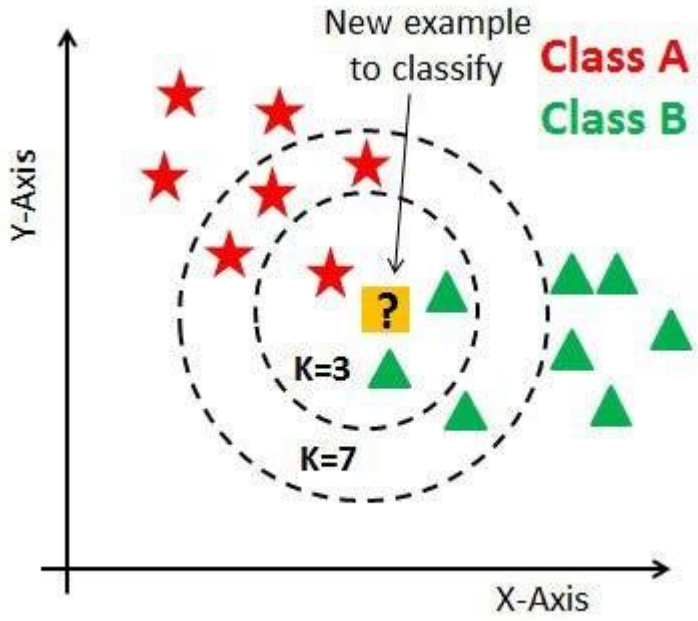


- As we can see the 3 nearest neighbors are from category A, hence this new data point must belong to category A.

How to choose a K value?



Kvalue indicates the count of the nearest neighbors. We have to compute distances between test points and trained labels points. Updating distance metrics with every iteration is computationally expensive, and that's why KNN is a lazy learning algorithm.



- As you can verify from the above image, if we proceed with K=3, then we predict that test input belongs to class B, and if we continue with K=7, then we predict that test input belongs to class A.
- That's how you can imagine that the K value has a powerful effect on KNN performance.

Then how to select the optimal K value?

- There are no pre-defined statistical methods to find the most favorable value of K.
- Initialize a random K value and start computing.
- Choosing a small value of K leads to unstable decision boundaries.

- The substantial K value is better for classification as it leads to smoothening the decision boundaries.
- Derive a plot between error rate and K denoting values in a defined range. Then choose the K value as having a minimum error rate.

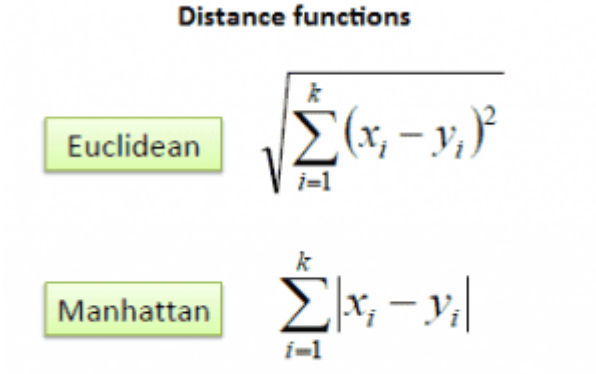
Now you will get the idea of choosing the optimal K value by implementing the model.

Calculating distance:

The first step is to calculate the distance between the new point and each training point. There are various methods for calculating this distance, of which the most commonly known methods are — Euclidian, Manhattan (for continuous) and Hamming distance (for categorical).

Euclidean Distance: Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (y).

Manhattan Distance: This is the distance between real vectors using the sum of their absolute difference.



Hamming Distance: It is used for categorical variables. If the value (x) and the value (y) are the same, the distance D will be equal to 0 . Otherwise D=1.

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

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

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

Ways to perform K-NN

*KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)*
algorithm : {'auto', 'ball_tree', 'kd_tree', 'brute'}, default='auto'

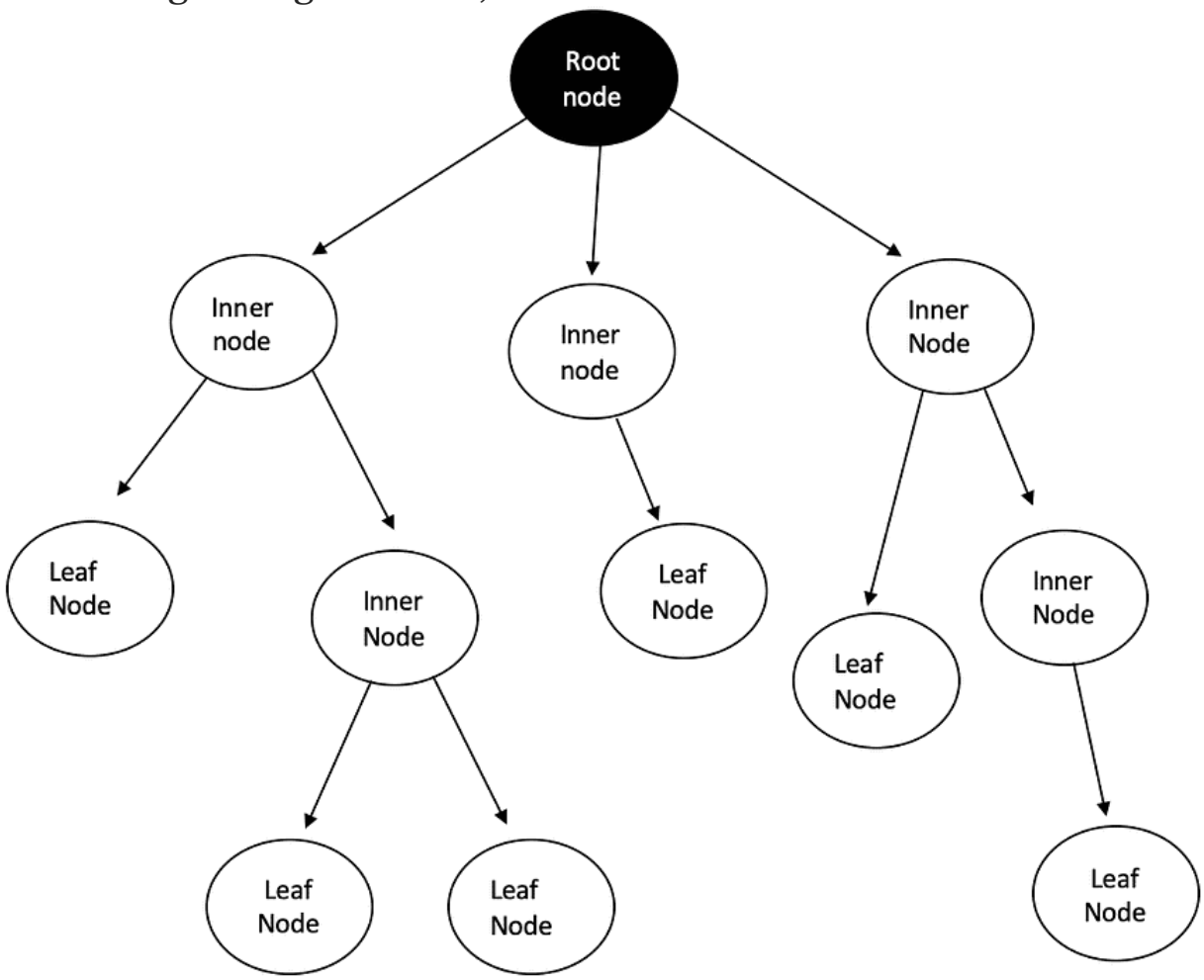
Brute Force

Lets consider for simple case with two dimension plot. If we look mathematically, the simple intuition is to calculate the euclidean distance from point of interest (of whose class we need to determine) to all the points in training set. Then we take class with majority points. This is called brute force method.

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

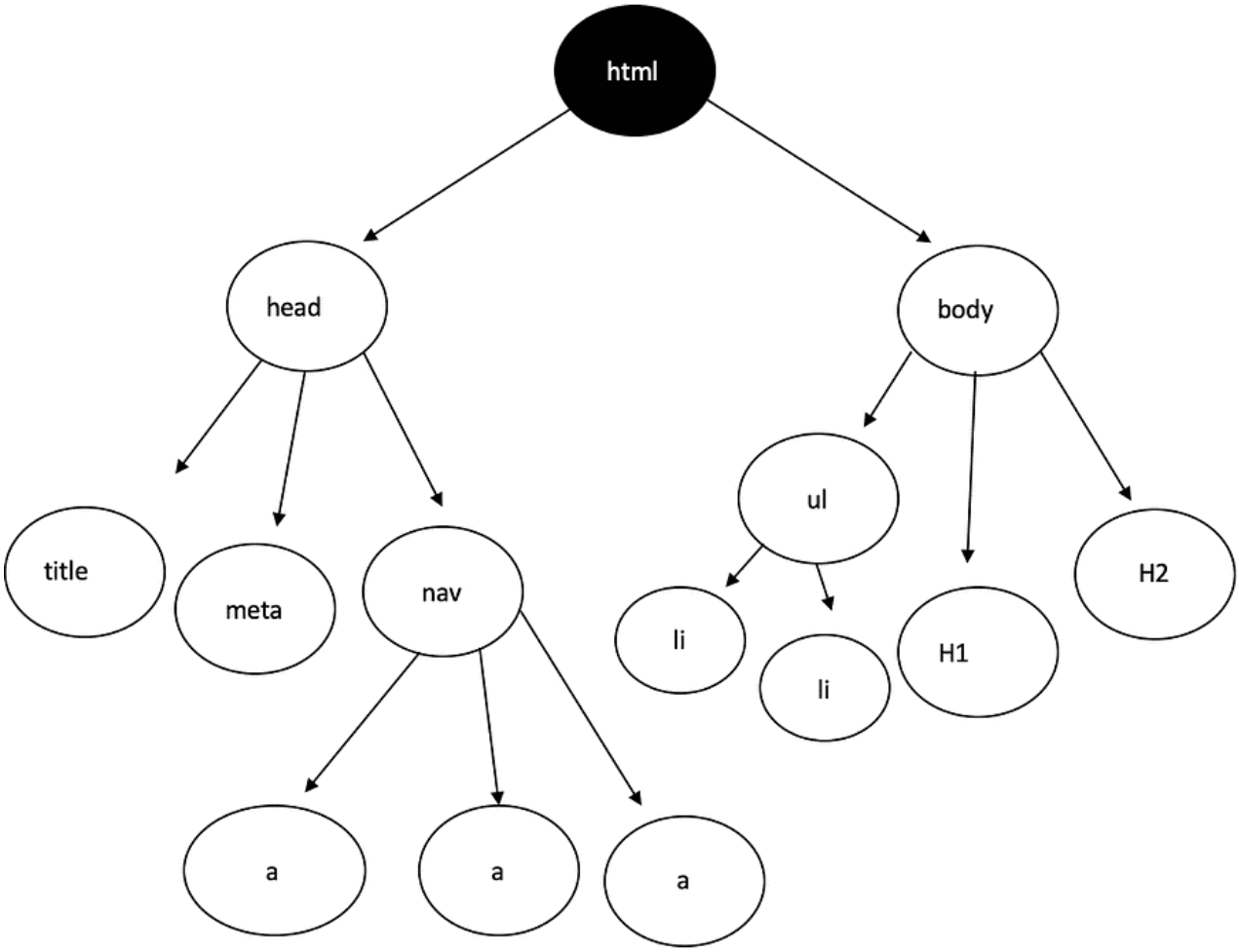
structure, so that when test data is provided, it would give out the result by traversing through the tree, which takes less time than brute search.



For a better understanding of how this can look like in a computer science topic, you can find below an HTML-code. A tree helps to structure a website and websites can normally be depicted using a tree.

```
<html>
<head>
    <meta charset=utf-8" />
    <title>Ball Tree vs. KD Tree</title>
    <nav>
    <a href="/r/">R</a>
    <a href="/js/">JavaScript</a>
    <a href="/python/">Python</a>
    </nav>
</head>
<body>
    <h1>What is a tree?</h1>
    <ul>
```

```
        <li>List item one</li>
        <li>List item two</li>
    </ul>
    <h2>How does a tree look like?</h2>
</body>
</html>
```

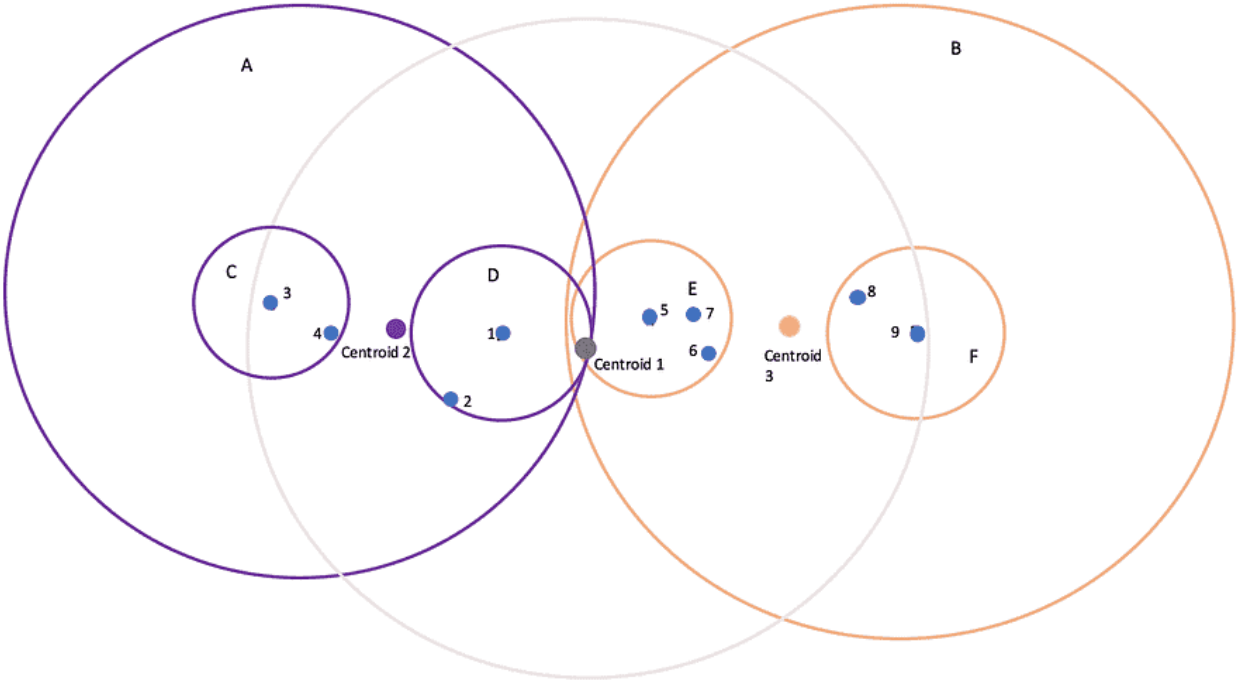


Ball Tree

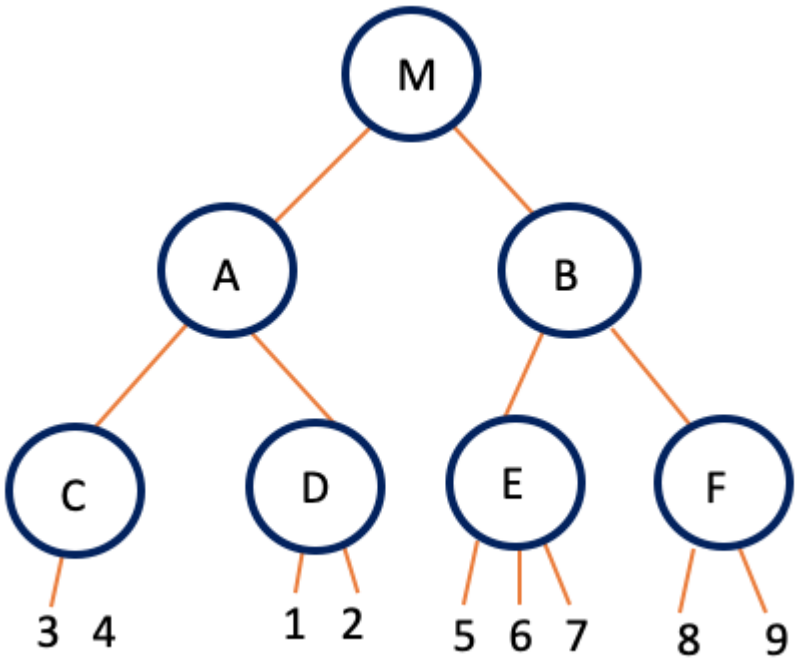
Similar to k-d trees, Ball trees are also hierarchical data structure. These are very efficient specially in case of higher dimensions.

- Two clusters are created initially
- All the data points must belong to atleast one of the clusters.
- One point cannot be in both clusters.

- Distance of the point is calculated from the centroid of the each cluster. The point closer to the centroid goes into that particular cluster.
- Each cluster is then divided into sub clusters again, and then the points are classified into each cluster on the basis of distance from centroid.
- This is how the clusters are kept to be divided till a certain depth.



The final resulting Ball Tree as follows,



Comparison and Summary

Brute Force may be the most accurate method due to the consideration of all data points. Hence, no data point is assigned to a false cluster. For small data sets, Brute Force is justifiable, however, for increasing data the KD or Ball Tree is better alternatives due to their speed and efficiency.

KNN model implementation

Let's start the application by importing all the required packages. Then read the telecommunication data file using read_csv() function.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import NullFormatter
import pandas as pd
import matplotlib.ticker as ticker
%matplotlib
inlinedf = pd.read_csv('Telecustomers.csv')
df.head()
```

	region	tenure	age	marital	address	income	ed	employ	retire	gender	reside	custcat
0	2	13	44	1	9	64.0	4	5	0.0	0	2	1
1	3	11	33	1	7	136.0	5	5	0.0	0	6	4
2	3	68	52	1	24	116.0	1	29	0.0	1	2	3
3	2	33	33	0	12	33.0	2	0	0.0	1	1	1
4	2	23	30	1	9	30.0	1	2	0.0	0	4	3

Dataset

As you can see, there are 12 columns, namely as region, tenure, age, marital, address, income, ed, employ, retire, gender, reside, and custcat. We have a target column, 'custcat' categorizes the customers into four groups:

- 1- Basic Service
- 2- E-Service
- 3- Plus Service
- 4- Total Service

```
X = df.drop(['custcat'], axis = 1)
y = df['custcat']
from sklearn import preprocessing
X = preprocessing.StandardScaler().fit(X).transform(X.astype(float))

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split( X, y,
test_size=0.2, random_state=4)

from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
#Train Model and Predict
k = 4
neigh = KNeighborsClassifier(n_neighbors = k).fit(X_train,y_train)
Pred_y = neigh.predict(X_test)
```

```
print("Accuracy of model at K=4 is",metrics.accuracy_score(y_test,
Pred_y))
```

```
Accuracy of model at K=4 is 0.32
```

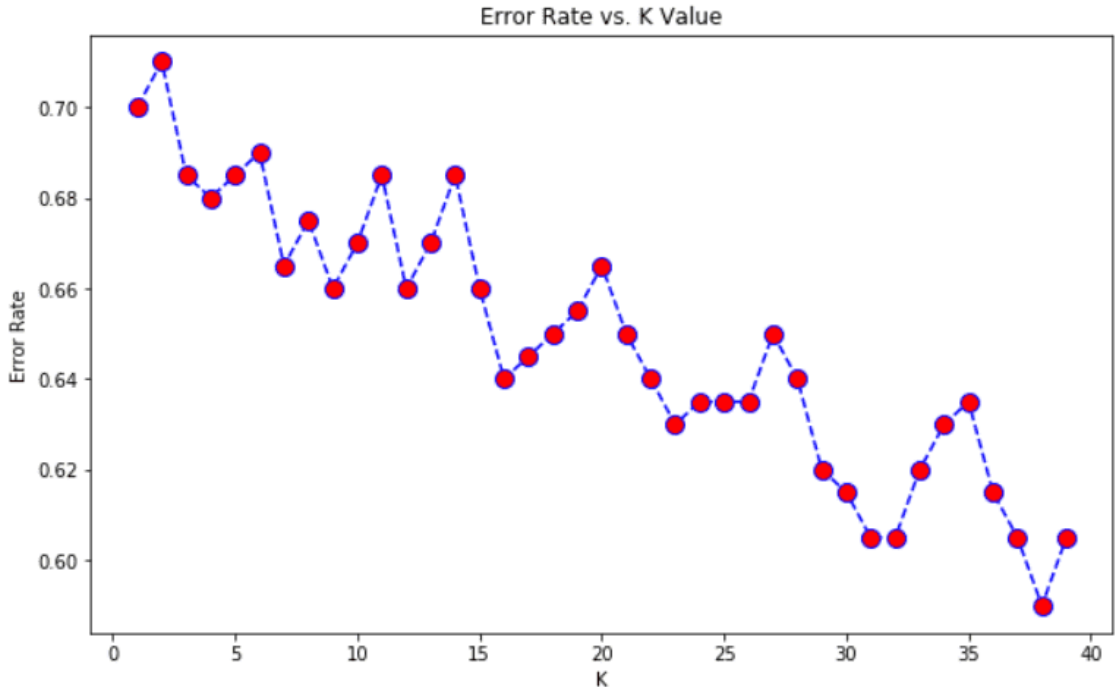
- We collect all independent data features into the X data-frame and target field into a y data-frame. Then we manipulate the data and normalize it.
- After splitting the data, we take 0.8% data for training and remaining for testing purposes.
- We import the classifier model from the sklearn library and fit the model by initializing K=4. So we have achieved an accuracy of 0.32 here.

Now it's time to improve the model and find out the optimal k value.

```
error_rate = []
for i in range(1,40):
 knn = KNeighborsClassifier(n_neighbors=i)
 knn.fit(X_train,y_train)
 pred_i = knn.predict(X_test)
 error_rate.append(np.mean(pred_i != y_test))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('Error Rate vs. K Value')
plt.xlabel('K')
plt.ylabel('Error Rate')
print("Minimum error:-",min(error_rate),"at K =",
error_rate.index(min(error_rate)))
```

```
Minimum error:- 0.59 at K = 37
```



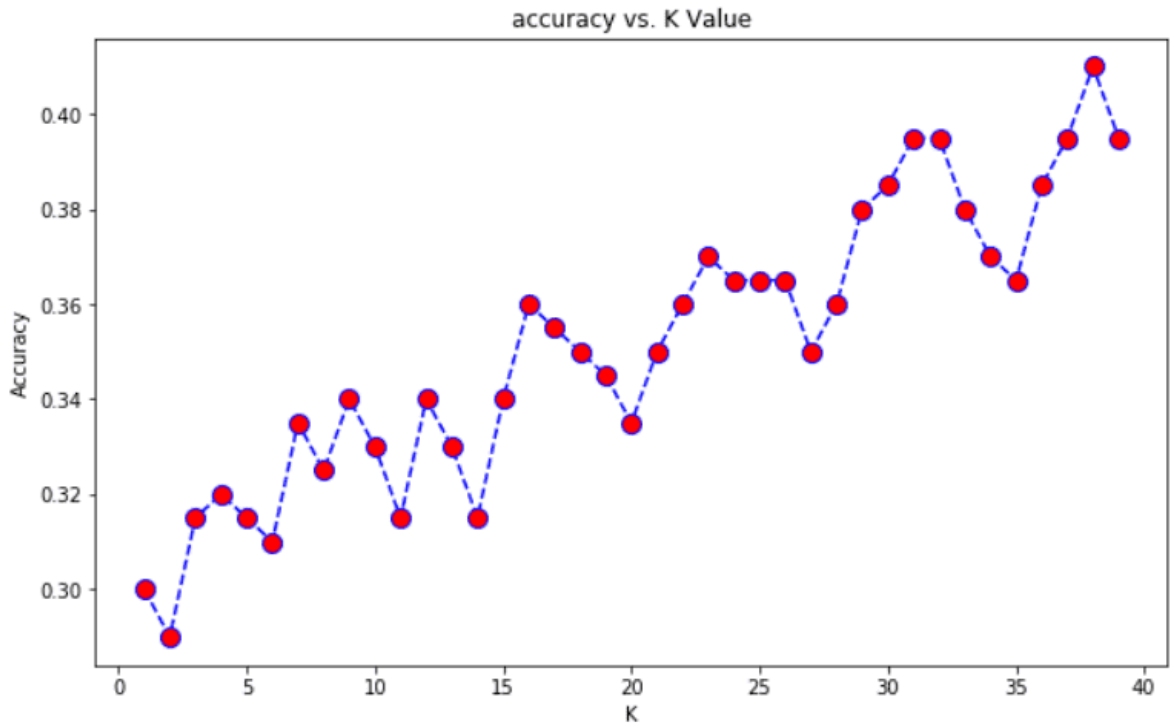
From the plot, you can see that the smallest error we got is 0.59 at K=37.

Further on, we visualize the plot between accuracy and K value.

```
acc = []
# Will take some time
from sklearn import metrics
for i in range(1,40):
    neigh = KNeighborsClassifier(n_neighbors=i).fit(X_train,y_train)
    yhat = neigh.predict(X_test)
    acc.append(metrics.accuracy_score(y_test, yhat))

plt.figure(figsize=(10,6))
plt.plot(range(1,40),acc,color = 'blue',linestyle='dashed',
         marker='o',markerfacecolor='red', markersize=10)
plt.title('accuracy vs. K Value')
plt.xlabel('K')
plt.ylabel('Accuracy')
print("Maximum accuracy:-",max(acc),"at K =",acc.index(max(acc)))
```

```
Maximum accuracy:- 0.41 at K = 37
```



Now you see the improved results. We got the accuracy of 0.41 at K=37. As we already derived the error plot and got the minimum error at k=37, so we will get better efficiency at that K value.