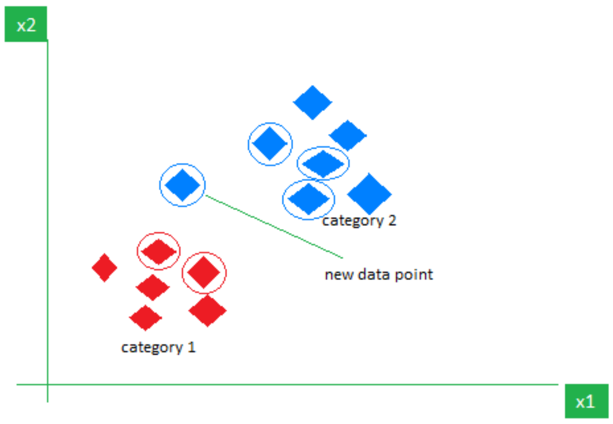
K-Nearest Neighbor (KNN) Algorithm

K-Nearest Neighbors (KNN) is a simple way to classify things by looking at what's nearby. Imagine a streaming service wants to predict if a new user is likely to cancel their subscription (churn) based on their age. They check the ages of its existing users and whether they churned or stayed. If most of the "K" closest users in age of new user canceled their subscription KNN will predict the new user might churn too. The key idea is that users with similar ages tend to have similar behaviors and KNN uses this closeness to make decisions.

Getting Started with K-Nearest Neighbors

K-Nearest Neighbors is also called as 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.

As an example, consider the following table of data points containing two features:



KNN Algorithm working visualization

The new point is classified as **Category 2** because most of its closest neighbors are blue squares. KNN assigns the category based on the majority of nearby points.

The image shows how KNN predicts the category of a **new data point** based on its closest neighbours.

- The **red diamonds** represent **Category 1** and the **blue squares** represent **Category 2**.
- The **new data point** checks its closest neighbours (circled points).
- Since the majority of its closest neighbours are blue squares (Category 2) KNN predicts the new data point belongs to Category 2.

KNN works by using proximity and majority voting to make predictions.

What is 'K' in K Nearest Neighbour ?

In the **k-Nearest Neighbours (k-NN)** algorithm **k** is just a number that tells the algorithm how many nearby points (neighbours) to look at when it makes a decision.

Example:

Imagine you're deciding which fruit it is based on its shape and size. You compare it to fruits you already know.

- If **k = 3**, the algorithm looks at the 3 closest fruits to the new one.
- If 2 of those 3 fruits are apples and 1 is a banana, the algorithm says the new fruit is an apple because most of its neighbours are apples.

How to choose the value of k for KNN Algorithm?

The value of k is critical in KNN as it determines the number of neighbors to consider when making predictions. Selecting the optimal value of k depends on the characteristics of the input data.

If the dataset has significant outliers or noise a higher k can help smooth out the predictions and reduce the influence of noisy data. However, choosing very high value can lead to underfitting where the model becomes too simplistic.

Statistical Methods for Selecting k:

- **Cross-Validation**: A robust method for selecting the best k is to perform k-fold cross-validation. This involves splitting the data into k subsets training the model on some subsets and testing it on the remaining ones and repeating this for each subset. The value of k that results in the highest average validation accuracy is usually the best choice.
- **Elbow Method**: In the **elbow method** we plot the model's error rate or accuracy for different values of k. As we increase k the error usually decreases initially. However, after a certain point the error rate starts to decrease more slowly. This point where the curve forms an "elbow" that point is considered as best k.
- **Odd Values for k**: It's also recommended to choose an odd value for k especially in classification tasks to avoid ties when deciding the majority class.

Distance Metrics Used in KNN Algorithm

KNN uses distance metrics to identify nearest neighbour, these neighbours are used for classification and regression task. To identify nearest neighbour we use below distance metrics:

1. Euclidean Distance

Euclidean distance is defined as the straight-line distance between two points in a plane or space. You can think of it like the shortest path you would walk if you were to go directly from one point to another.

$$\text{distance}(x, X_i) = \sqrt{\sum_{j=1}^{d}(x_j - X_{i_j})^2]}$$

2. Manhattan Distance

This is the total distance you would travel if you could only move along horizontal and vertical lines (like a grid or city streets). It's also called "taxicab distance" because a taxi can only drive along the grid-like streets of a city.

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

3. Minkowski Distance

Minkowski distance is like a family of distances, which includes both **Euclidean** and **Manhattan distances** as special cases.

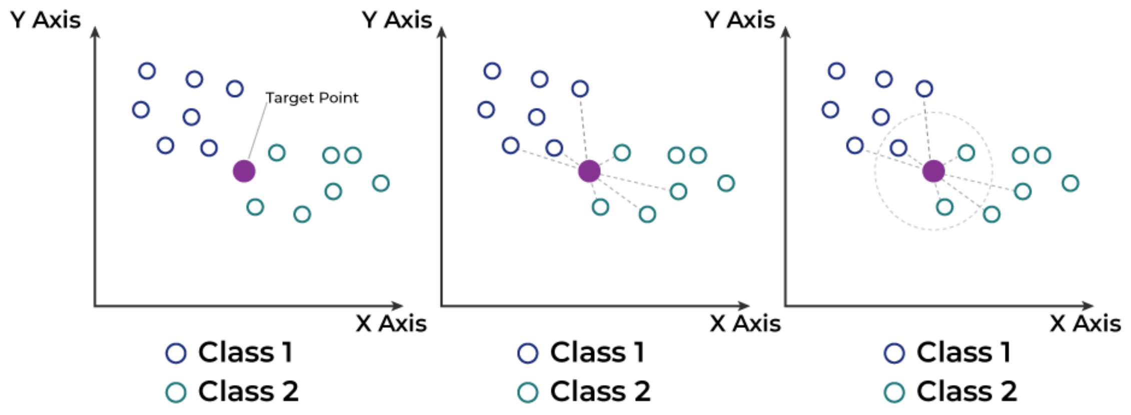
$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^p\right)^{\frac{1}{p}}$$

From the formula above we can say that when p = 2 then it is the same as the formula for the Euclidean distance and when p = 1 then we obtain the formula for the Manhattan distance.

So, you can think of Minkowski as a flexible distance formula that can look like either Manhattan or Euclidean distance depending on the value of p

Working of KNN algorithm

The K-Nearest Neighbors (KNN) algorithm operates on the principle of similarity where it predicts the label or value of a new data point by considering the labels or values of its K nearest neighbors in the training dataset.



Step-by-Step explanation of how KNN works is discussed below:

Step 1: Selecting the optimal value of K

- K represents the number of nearest neighbors that needs to be considered while making prediction.

Step 2: Calculating distance

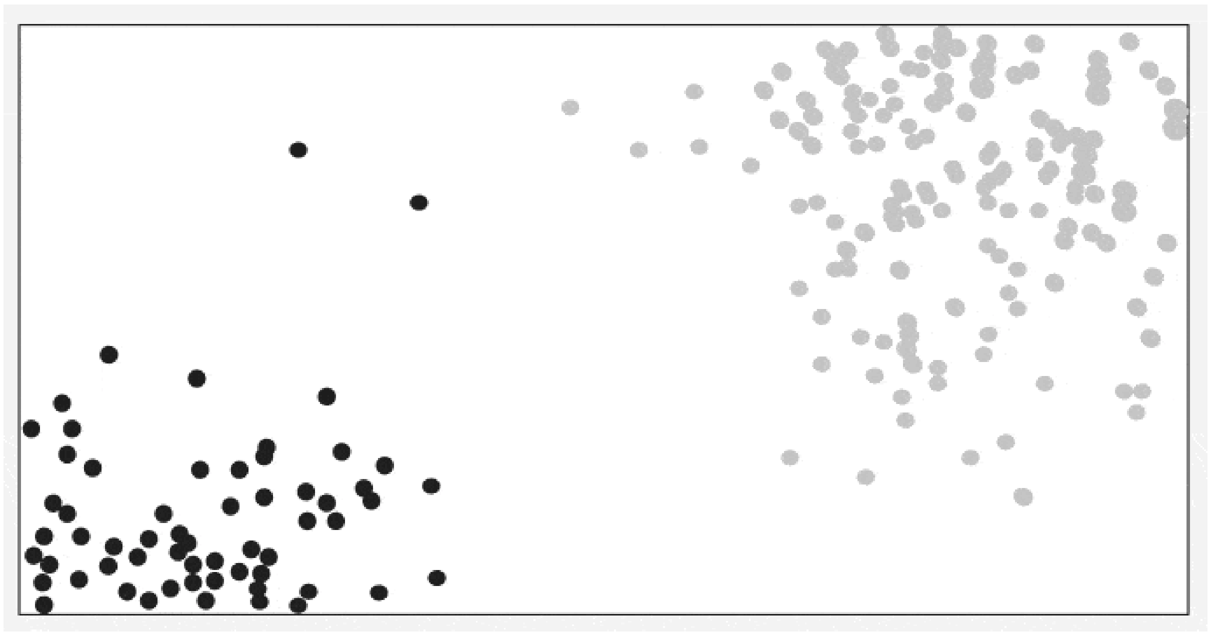
- To measure the similarity between target and training data points Euclidean distance is used. Distance is calculated between data points in the dataset and target point.

Step 3: Finding Nearest Neighbors

- The k data points with the smallest distances to the target point are nearest neighbors.

Step 4: Voting for Classification or Taking Average for Regression

- When you want to classify a data point into a category (like spam or not spam), the K-NN algorithm looks at the **K closest points** in the dataset. These closest points are called neighbors. The algorithm then looks at which category the neighbors belong to and picks the one that appears the most. This is called **majority voting**.
- In regression, the algorithm still looks for the **K closest points**. But instead of voting for a class in classification, it takes the **average** of the values of those K neighbors. This average is the predicted value for the new point for the algorithm.



Working of KNN Algorithm

It shows how a test point is classified based on its nearest neighbors. As the test point moves the algorithm identifies the closest 'k' data points i.e 5 in this case and assigns test point the majority class label that is grey label class here.

Python Implementation of KNN Algorithm

1. Importing Libraries:

import numpy as np

from collections import Counter

- **Counter**: is used to count the occurrences of elements in a list or iterable. In KNN after finding the k nearest neighbors labels Counter helps count how many times each label appears.

2. Defining the Euclidean Distance Function:

def euclidean_distance(point1, point2):

```
    return np.sqrt(np.sum((np.array(point1) - np.array(point2))**2))
```

- **euclidean_distance**: to calculate euclidean distance between points

3. KNN Prediction Function:

```
def knn_predict(training_data, training_labels, test_point, k):
    distances = []
    for i in range(len(training_data)):
        dist = euclidean_distance(test_point, training_data[i])
        distances.append((dist, training_labels[i]))
    distances.sort(key=lambda x: x[0])
    k_nearest_labels = [label for _, label in distances[:k]]
    return Counter(k_nearest_labels).most_common(1)[0][0]
```

- **distances.append**: Each distance is paired with the corresponding label (training_labels[i]) of the training data. This pair is stored in a list called distances.
- **distances.sort**: The list of distances is sorted in ascending order so that the closest points are at the beginning of the list.
- **k_nearest_labels**: The function then selects the labels of the k closest neighbors.
- The labels of the k nearest neighbors are counted using the Counter class, and the most frequent label is returned as the prediction for the test_point. This is based on the majority vote of the k neighbors.

4. Training Data, Labels and Test Point:

```
training_data = [[1, 2], [2, 3], [3, 4], [6, 7], [7, 8]]

training_labels = ['A', 'A', 'A', 'B', 'B']

test_point = [4, 5]

k = 3
```

5. Prediction and Output:

```
prediction = knn_predict(training_data, training_labels, test_point, k)

print(prediction)
```

Output:

```
A
```

The algorithm calculates the distances of the test point [4, 5] to all training points, selects the 3 closest points (as k = 3), and determines their labels. Since the majority of the closest points are labelled **'A'**, the test point is classified as **'A'**.

Applications of the KNN Algorithm

Here are some real life applications of KNN Algorithm.

- **Recommendation Systems**: Many recommendation systems, such as those used by Netflix or Amazon, rely on KNN to suggest products or content. KNN observes at user behavior and finds similar users. If user A and user B have similar preferences, KNN might recommend movies that user A liked to user B.
- **Spam Detection**: KNN is widely used in filtering spam emails. By comparing the features of a new email with those of previously labeled spam and non-spam emails, KNN can predict whether a new email is spam or not.
- **Customer Segmentation**: In marketing firms, KNN is used to segment customers based on their purchasing behavior . By comparing new customers to existing customers, KNN can easily group customers into segments with similar choices and preferences. This helps businesses target the right customers with right products or advertisements.
- **Speech Recognition**: KNN is often used in speech recognition systems to transcribe spoken words into text. The algorithm compares the features of the spoken input with those of known speech patterns. It then predicts the most likely word or command based on the closest matches.

Advantages and Disadvantages of the KNN Algorithm

Advantages:

- **Easy to implement:** The KNN algorithm is easy to implement because its complexity is relatively low as compared to other machine learning algorithms.
- **No training required:** KNN stores all data in memory and doesn't require any training so when new data points are added it automatically adjusts and uses the new data for future predictions.
- **Few Hyperparameters:** The only parameters which are required in the training of a KNN algorithm are the value of k and the choice of the distance metric which we would like to choose from our evaluation metric.
- **Flexible**: It works for **Classification** problem like is this email spam or not? and also work for **Regression task** like predicting house prices based on nearby similar houses.

Disadvantages:

- **Doesn't scale well:** KNN is considered as a "lazy" algorithm as it is very slow especially with large datasets
- **Curse of Dimensionality:** When the number of features increases KNN struggles to classify data accurately a problem known as curse of dimensionality.
- **Prone to Overfitting:** As the algorithm is affected due to the curse of dimensionality it is prone to the problem of overfitting as well.

Implementation of KNN classifier using Sklearn

Last Updated : 28 Nov, 2019

-
-
-

Prerequisite: K-Nearest Neighbours Algorithm

K-Nearest Neighbors is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the given data).

This article will demonstrate how to implement the **K-Nearest neighbors classifier algorithm** using Sklearn library of Python.

Step 1: Importing the required Libraries

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.neighbors import KNeighborsClassifier

import matplotlib.pyplot as plt

import seaborn as sns
```

Step 2: Reading the Dataset

```
cd C:\Users\Dev\Desktop\Kaggle\Breast_Cancer

# Changing the read file location to the location of the file


df = pd.read_csv('data.csv')


y = df['diagnosis']

X = df.drop('diagnosis', axis = 1)

X = X.drop('Unnamed: 32', axis = 1)

X = X.drop('id', axis = 1)
```

```
# Separating the dependent and independent variable


X_train, X_test, y_train, y_test = train_test_split(
            X, y, test_size = 0.3, random_state = 0)
# Splitting the data into training and testing data
```

Step 3: Training the model

```
K = []
training = []
test = []
scores = {}


for k in range(2, 21):
    clf = KNeighborsClassifier(n_neighbors = k)
    clf.fit(X_train, y_train)


    training_score = clf.score(X_train, y_train)
    test_score = clf.score(X_test, y_test)
    K.append(k)


    training.append(training_score)
    test.append(test_score)
    scores[k] = [training_score, test_score]
```

Step 4: Evaluating the model

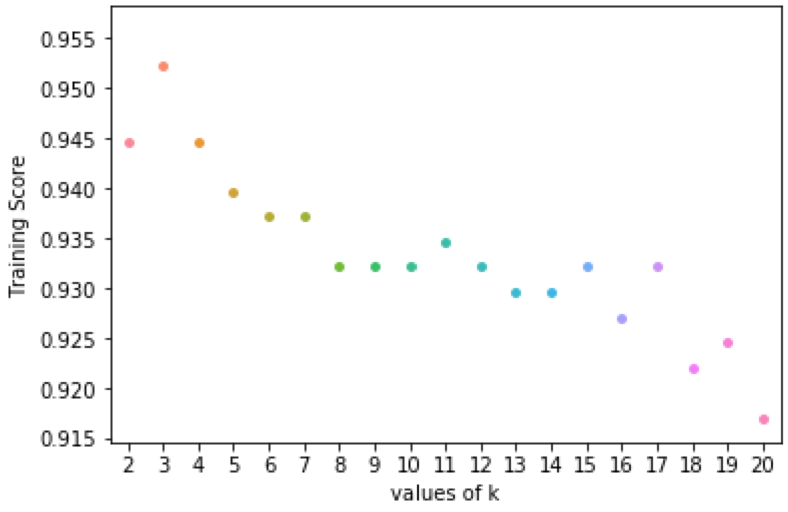
```
for keys, values in scores.items():
    print(keys, ':', values)
```

```
2 : [0.9447236180904522, 0.9298245614035088]
3 : [0.9522613065326633, 0.9181286549707602]
4 : [0.9447236180904522, 0.9298245614035088]
5 : [0.9396984924623115, 0.9473684210526315]
6 : [0.9371859296482412, 0.9473684210526315]
7 : [0.9371859296482412, 0.9532163742690059]
8 : [0.9321608040201005, 0.9532163742690059]
9 : [0.9321608040201005, 0.9590643274853801]
10 : [0.9321608040201005, 0.9649122807017544]
11 : [0.9346733668341709, 0.9649122807017544]
12 : [0.9321608040201005, 0.9649122807017544]
13 : [0.9296482412060302, 0.9649122807017544]
14 : [0.9296482412060302, 0.9649122807017544]
15 : [0.9321608040201005, 0.9649122807017544]
16 : [0.9271356783919598, 0.9649122807017544]
17 : [0.9321608040201005, 0.9649122807017544]
18 : [0.9221105527638191, 0.9649122807017544]
19 : [0.9246231155778895, 0.9649122807017544]
20 : [0.9170854271356784, 0.9649122807017544]
```

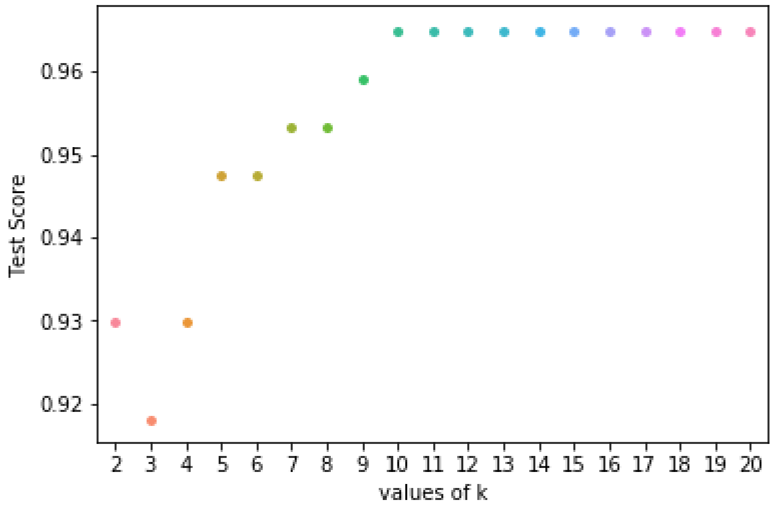
We now try to find the optimum value for ‘k’ ie the number of nearest neighbors.

Step 5: Plotting the training and test scores graph

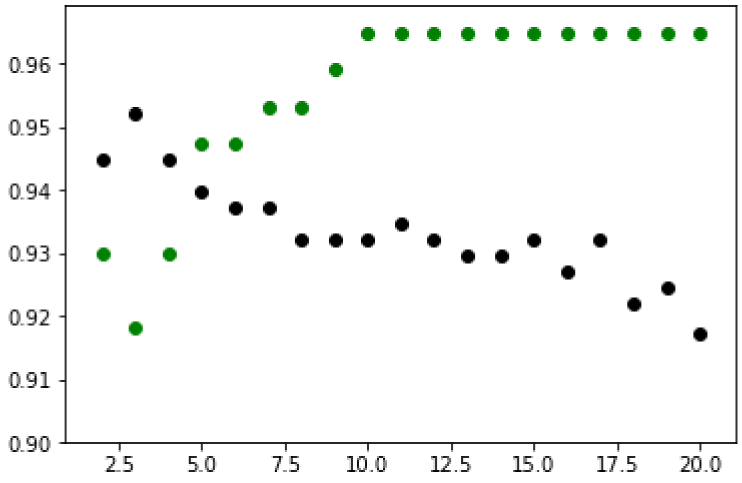
```
ax = sns.stripplot(K, training);

ax.set(xlabel ='values of k', ylabel ='Training Score')


plt.show()

# function to show plot
```



```
ax = sns.stripplot(K, test);

ax.set(xlabel ='values of k', ylabel ='Test Score')

plt.show()
```



```
plt.scatter(K, training, color ='k')

plt.scatter(K, test, color ='g')

plt.show()

# For overlapping scatter plots
```



From the above scatter plot, we can come to the conclusion that the optimum value of k will be around 5.