Topic: K-Nearest Neighbors (KNN)

Student Number: 23033174

Name:-Nandini

Email id :-reddynandini174@gmail.com

Introduction :

K-Nearest Neighbours (K-NN) is a popular supervised machine learning technique due to its simplicity and intuitiveness. It is particularly useful for classification and regression It is a straightforward, understandable strategy based on the notion of "learning by analogy." Instead of creating a model during the training phase, K-NN produces predictions based on the nearest data points (neighbours) to a new input.

K-Nearest Neighbor(KNN) Algorithm steps:-

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 availablecases and put the new case into the category that is most similar to the availablecategories.

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.

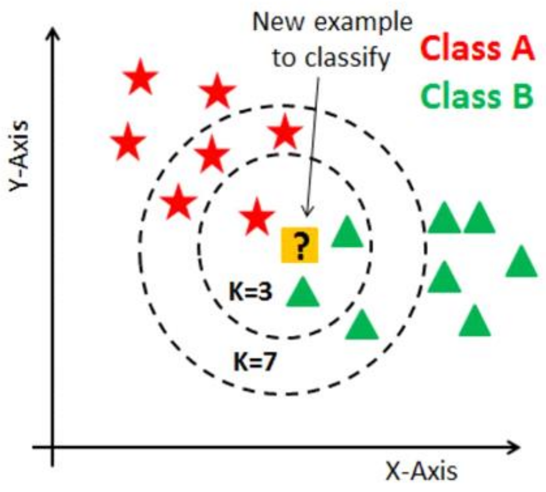
K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

It is also called a lazy learner algorithm because it does not learn from the trainingset 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.

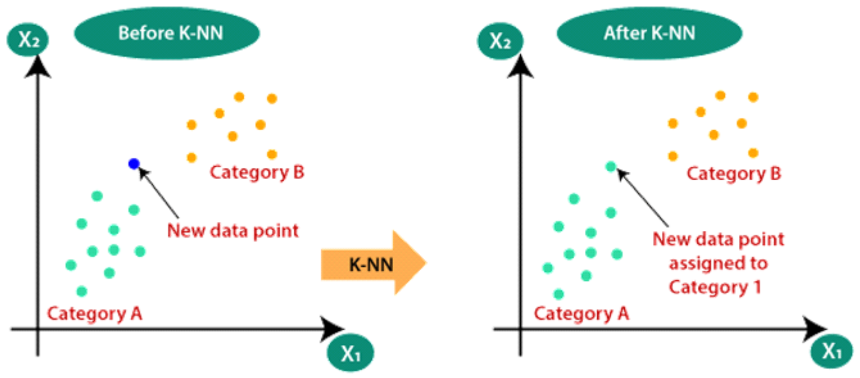
Example: Suppose, we have an image of a creature that looks similar to cat and dog, but we want to know either it is a cat or dog. So for this identification, we can use the KNN algorithm, as it works on a similarity measure. Our KNN model will find the similar features of the new data set to the cats and dogs images and based on the most similar features it will put it in either cat or dog category.

KNN Classifier
Input value
Predicted Output



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 Works:-

Data Representation: All data points are plotted in an n-dimensional feature space, with nn representing the number of features.

Predictions for New Data: When a new data point is introduced, K-NN evaluates the k nearest points from the training dataset.

The algorithm determines the "distance" between new and existing points using metrics such as

Euclidean distance (default for continuous data).

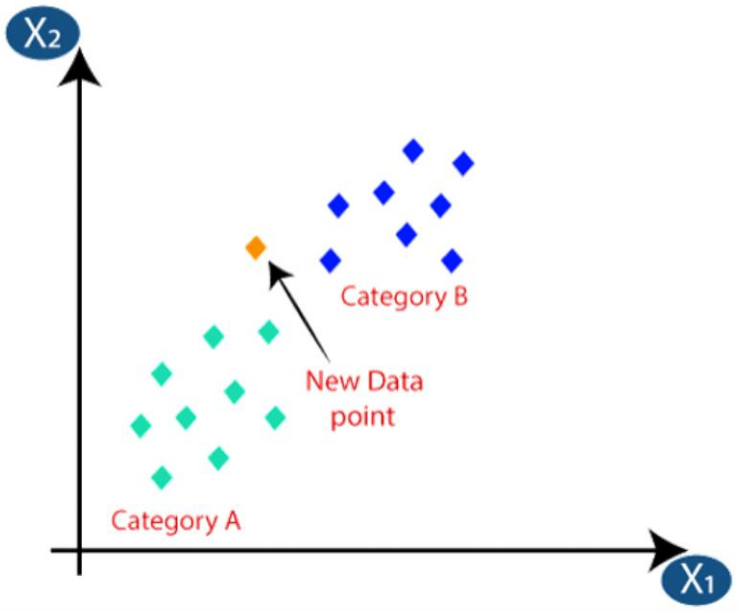
Manhattan distance and cosine similarity (for high-dimensional or text data).

Decision Rule:

To classify, assign the class with the highest frequency among the k neighbours.

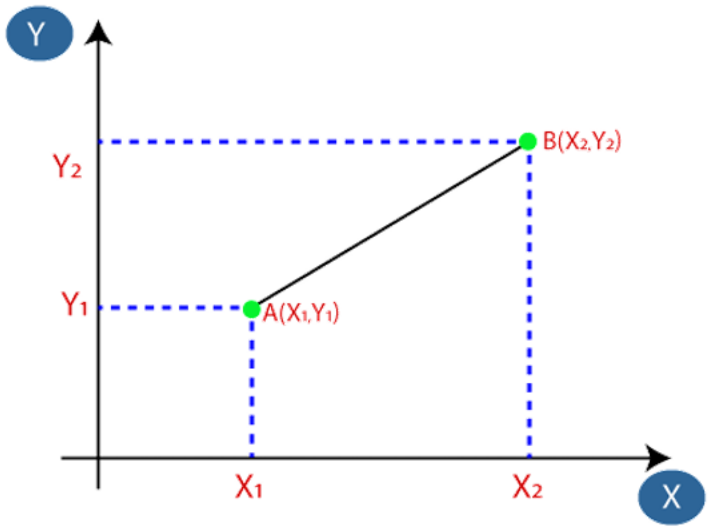
For Regression, predict the average or weighted average of the neighbours' values.

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.

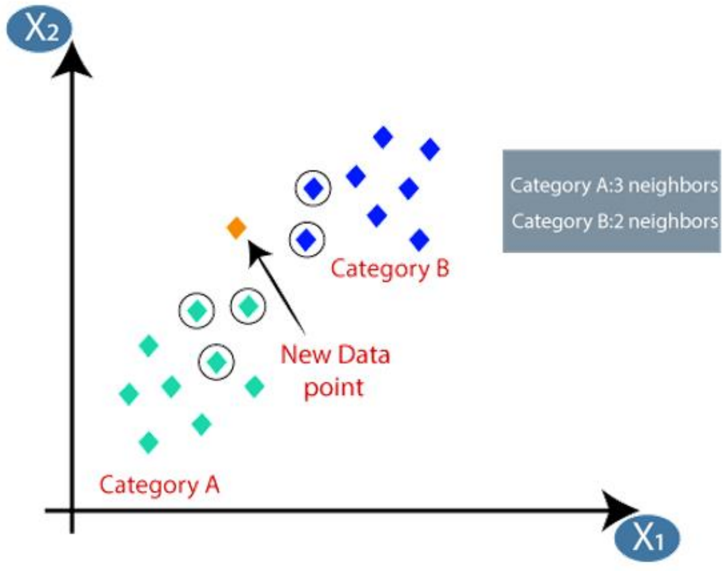
Next, we will calculate the Euclidean distance between the data points. The Euclidean distance is the distance between two points, which we have already studied in geometry. It can be calculated as:



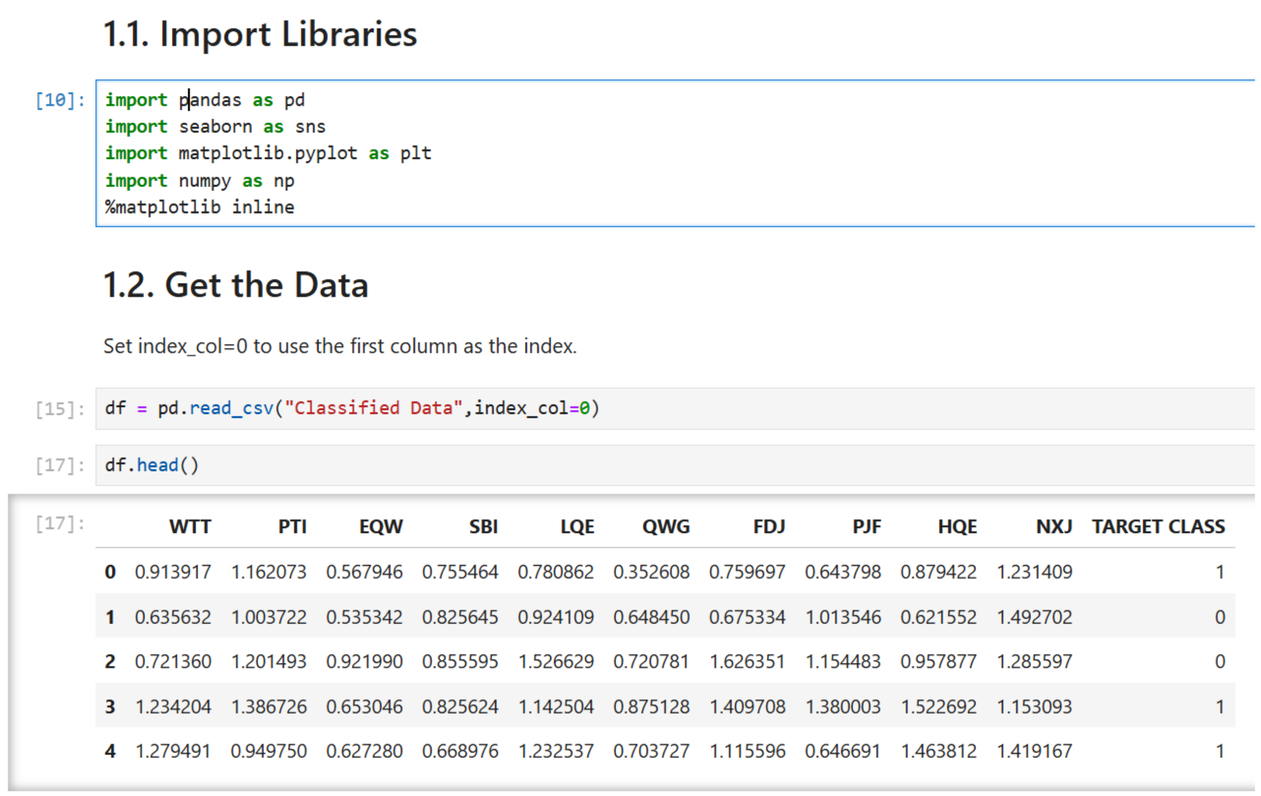
Euclidean distance between A1 and B2=(x2-x2)2+(y2-y2)2

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.



1.1. Import Libraries

```
[10]: import pandas as pd
      import seaborn as sns
      import matplotlib.pyplot as plt
      import numpy as np
      %matplotlib inline
```

1.2. Get the Data

Set index_col=0 to use the first column as the index.

```
[15]: df = pd.read_csv("Classified Data",index_col=0)
```

```
[17]: df.head()
```

[17]:

	WTT	PTI	EQW	SBI	LQE	QWG	FDJ	PJF	HQE	NXJ	TARGET CLASS
0	0.913917	1.162073	0.567946	0.755464	0.780862	0.352608	0.759697	0.643798	0.879422	1.231409	1
1	0.635632	1.003722	0.535342	0.825645	0.924109	0.648450	0.675334	1.013546	0.621552	1.492702	0
2	0.721360	1.201493	0.921990	0.855595	1.526629	0.720781	1.626351	1.154483	0.957877	1.285597	0
3	1.234204	1.386726	0.653046	0.825624	1.142504	0.875128	1.409708	1.380003	1.522692	1.153093	1
4	1.279491	0.949750	0.627280	0.668976	1.232537	0.703727	1.115596	0.646691	1.463812	1.419167	1

This part imports the required Python libraries for data processing, visualisation, and numerical computations.

pandas as pd: Used to manage and manipulate structured data (dataframes and series).

seaborn (sns): A library for producing visually appealing statistical charts.

matplotlib.pyplot as plt is a package for making simple plots and visualisations.

Numpy as np: Used for numerical computations and array management.

%matplotlib inline: Ensures that plots are displayed inline in the notebook

The pd.read_csv() function reads data from the file "Classified Data".

index_col=0: Uses the dataset's initial column as an index, making it easier to access rows.

The dataset consists of the following features:-

Feature :- columns include numerical features such as WTT, PTI, EQW, SBI, and LQE. These most likely correspond to variables or measures for each data point.

Target class: is the classification label (for example, 1 or 0). This implies a binary classification issue in which the goal is to predict the target class using the other feature columns.

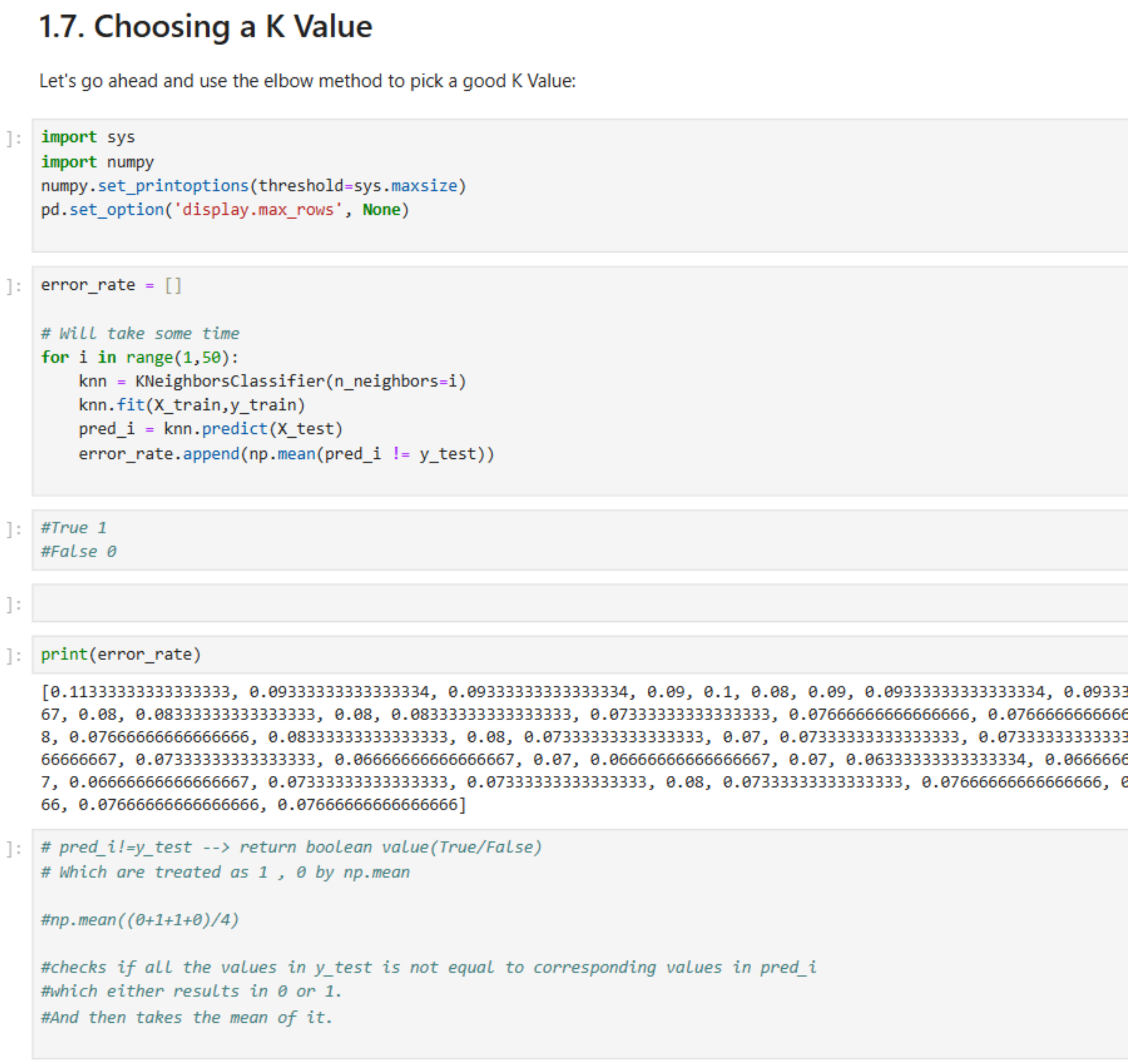
How to select the value of K in the K-NN Algorithm?

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.



1.7. Choosing a K Value

Let's go ahead and use the elbow method to pick a good K Value:

```
]: import sys
   import numpy
   numpy.set_printoptions(threshold=sys.maxsize)
   pd.set_option('display.max_rows', None)
```

```
]: error_rate = []

   # Will take some time
   for i in range(1,50):
       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))
```

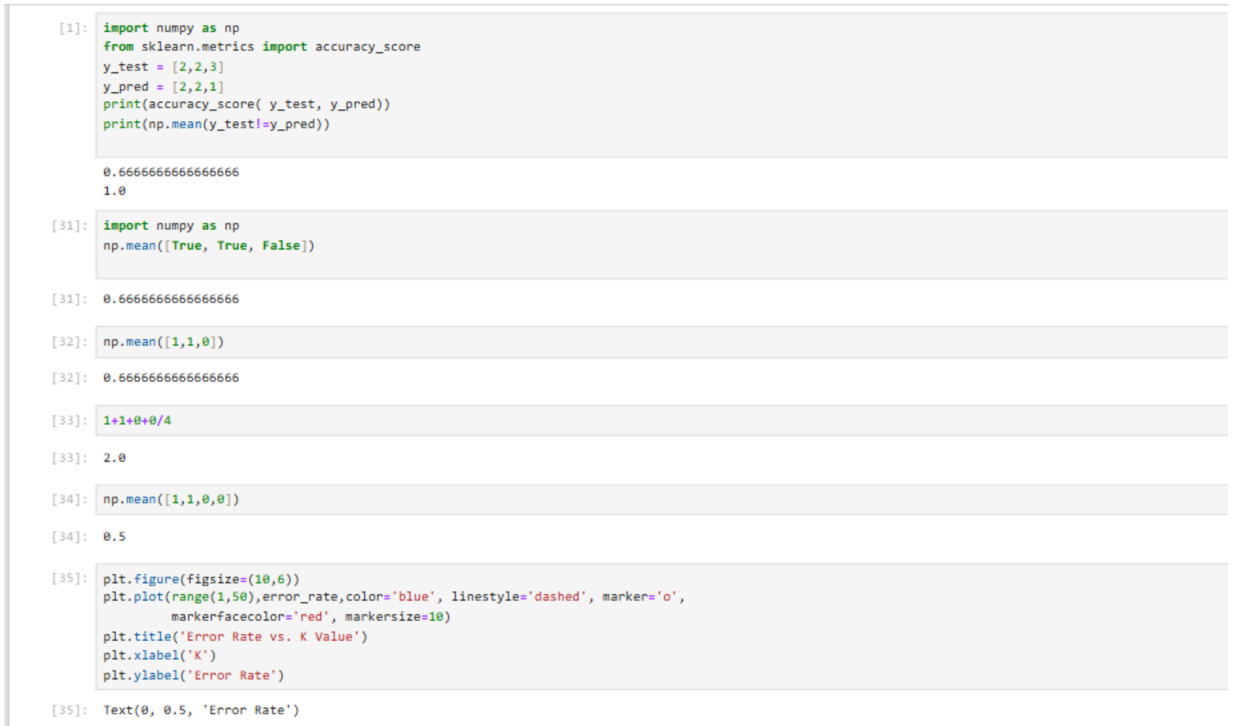
```
]: #True 1
   #False 0
```

```
]:
```

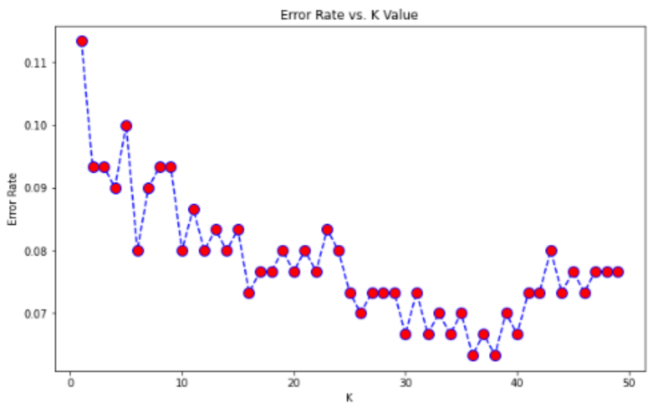
```
]: print(error_rate)
```

```
[0.11333333333333333, 0.09333333333333334, 0.09333333333333334, 0.09, 0.1, 0.08, 0.09, 0.09333333333333334, 0.09333
67, 0.08, 0.08333333333333333, 0.08, 0.08333333333333333, 0.07333333333333333, 0.07666666666666666, 0.0766666666666
8, 0.07666666666666666, 0.08333333333333333, 0.08, 0.07333333333333333, 0.07, 0.07333333333333333, 0.0733333333333
66666667, 0.07333333333333333, 0.06666666666666667, 0.07, 0.06666666666666667, 0.07, 0.06333333333333334, 0.066666
7, 0.06666666666666667, 0.07333333333333333, 0.07333333333333333, 0.08, 0.07333333333333333, 0.07666666666666666, 0
66, 0.07666666666666666, 0.07666666666666666]
```

```
]: # pred_i!=y_test --> return boolean value(True/False)
   # Which are treated as 1 , 0 by np.mean

   #np.mean((0+1+1+0)/4)

   #checks if all the values in y_test is not equal to corresponding values in pred_i
   #which either results in 0 or 1.
   #And then takes the mean of it.
```



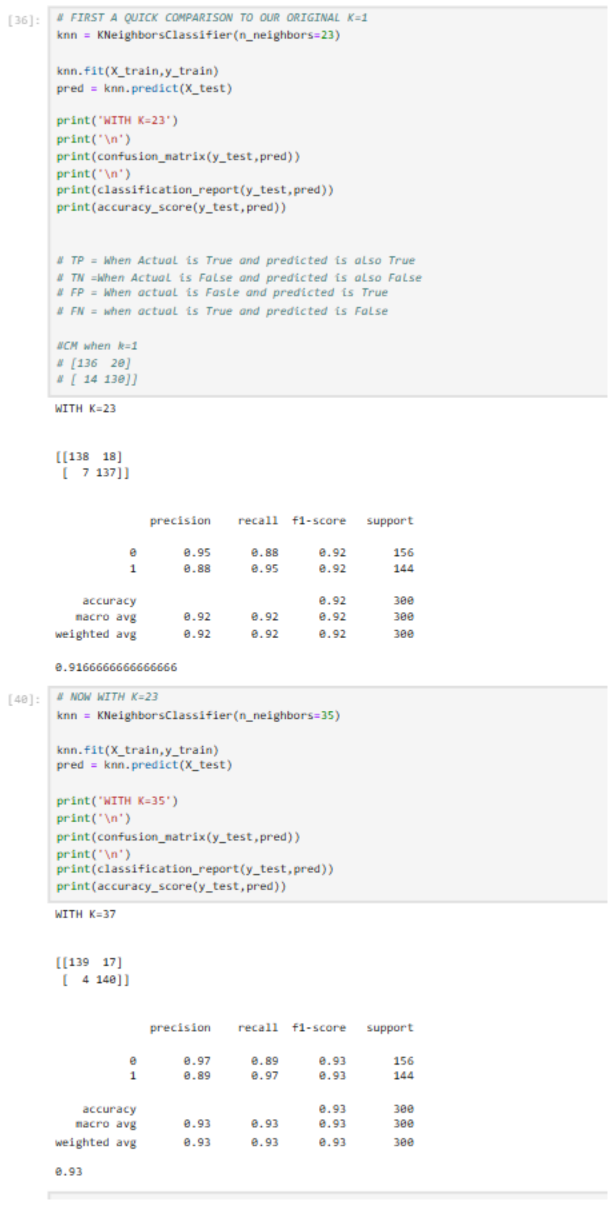
```
[1]: import numpy as np
     from sklearn.metrics import accuracy_score
     y_test = [2,2,3]
     y_pred = [2,2,1]
     print(accuracy_score( y_test, y_pred))
     print(np.mean(y_test!=y_pred))

     0.6666666666666666
     1.0

[31]: import numpy as np
      np.mean([True, True, False])

[31]: 0.6666666666666666

[32]: np.mean([1,1,0])

[32]: 0.6666666666666666

[33]: 1+1+0+0/4

[33]: 2.0

[34]: np.mean([1,1,0,0])

[34]: 0.5

[35]: plt.figure(figsize=(10,6))
      plt.plot(range(1,50),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')

[35]: Text(0, 0.5, 'Error Rate')
```



Here we can see that after arouns K>23 the error rate just tends to hover around 0.06-0.05 Let's retrain the model with that and check the classification report!

The code uses the numpy library and sklearn.metrics routines to calculate accuracy scores and means. A plot of the error rate versus the K value for a K-Nearest Neighbours (KNN) algorithm is also included, created with matplotlib.The plot "Error Rate vs. K Value" depicts the error rate on the y-axis and the K value on the x-axis. The plot shows that after K=23, the error rate stabilises at 0.06-0.05.

Comparison to our original k=23and K=35



```
[36]: # FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=23)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=23')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
print(accuracy_score(y_test,pred))


# TP = When Actual is True and predicted is also True
# TN =When Actual is False and predicted is also False
# FP = When actual is Fasle and predicted is True
# FN = when actual is True and predicted is False

#CM when k=1
# [136  20]
# [ 14 130]]
```

```
WITH K=23


[[138  18]
 [  7 137]]


              precision    recall  f1-score   support

           0       0.95      0.88      0.92       156
           1       0.88      0.95      0.92       144

    accuracy                           0.92       300
   macro avg       0.92      0.92      0.92       300
weighted avg       0.92      0.92      0.92       300

0.9166666666666666
```

```
[40]: # NOW WITH K=23
knn = KNeighborsClassifier(n_neighbors=35)

knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print('WITH K=35')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
print(accuracy_score(y_test,pred))
```

```
WITH K=37


[[139  17]
 [  4 140]]


              precision    recall  f1-score   support

           0       0.97      0.89      0.93       156
           1       0.89      0.97      0.93       144

    accuracy                           0.93       300
   macro avg       0.93      0.93      0.93       300
weighted avg       0.93      0.93      0.93       300

0.93
```

Examines the performance of a K-Nearest Neighbours (KNN) classifier at different K values (23 and 35). The code employs the KNeighborsClassifier from the sklearn.neighbors module to assess the model using a confusion matrix, classification report, and accuracy score.The code assesses the KNN classifier's performance at K values of 23 and 35, displaying the confusion matrix, classification report, and accuracy score for each. The findings show that the classifier works marginally better with K=35, reaching an accuracy of 0.93 versus 0.92 with K=23.

Key Parameters:-

The value of k (k):- Determines the number of neighbours evaluated for prediction.

Small k value:- May overfit because to noise sensitivity, but can capture local patterns.

Large k value :- Allows for smoother judgements, but may result in missing finer information.

The Distance Metric defines how the similarity between data points is calculated. The option is determined by the type of the data.

Advantages :-

Simplicity:- K-NN is easy to learn and implement. It does not require complicated mathematical modelling or parameter adjustment.

Non-Parametric: It makes no assumptions about the underlying data distribution, making it appropriate for non-linear and complex interactions in the data.

Versatility: K-NN is capable of handling both classification and regression jobs effectively,additionally, it supports multi-class classification

Adaptability: K-NN is effective for smaller datasets and problems where interpretability is critical.

Real-world Intuition: K-NN operates on the premise of "learning by analogy," which closely resembles how humans frequently make decisions (for example, based on similar past experiences).

Disadvantages:-

Computational Cost: Distance calculations are required for each prediction, which can be time-consuming for large datasets.

Memory Usage: Saves entire training data, which can be inefficient.

Scale-sensitive: Unless normalised, features with higher ranges dominate distance estimates.

The Curse of Dimensionality states that performance deteriorates as the number of characteristics (dimensions) grows.

When to use K-NN:-

K-NN is especially beneficial in these scenarios:

1. Low to Medium-Sized Datasets

K-NN is computationally demanding since it must calculate the distance between each prediction. It works best with small to medium datasets that have a manageable runtime.

2. Data with clearly defined clusters or groups

K-NN works best with data that contains identifiable patterns or clusters, making it easier to group or categorise points.

3. Problems Requiring Local Information K-NN is effective for identifying local associations between data points, taking into account the nearest neighbours.

4. Mixed Data Types:- K-NN can handle datasets with multiple feature types (numerical and categorical), as long as the suitable distance metric is used.

5. Non-Linear Relationships:- K-NN is non-parametric, hence it can handle Non-linear decision boundaries, which algorithms such as Logistic Regression may struggle with.

Applications of KNN

1. Image and Video Recognition Examples include classifying objects based on pixel similarity and clustering similar image features.

2. Recommendation Systems One example is collaborative filtering, which suggests movies or products based on user preferences.

3. Medical Diagnosis: Sorting patients into disease categories based on symptoms and test results.

4. Anomaly Detection: Identifying fraud by detecting transactions that deviate significantly from their neighbours.

5. Text Classification Example: Classifying texts as spam or not based on word frequency or similarity to labelled samples.

6. Handwriting and Character Recognition Example: Comparing handwritten numbers to previously labelled samples.

7. Customer Segmentation: Identifying clients with similar purchasing habits for targeted marketing.

Conclusion

K-NN is a robust technique that may be used in a variety of real-world applications. It is particularly effective for smaller datasets with apparent patterns and clusters.