Data Warehousing & Data Mining LAB - G2
EXPERIMENT 9

- ASHISH KUMAR
- 2K18/SE/041

Aim:- Write a program to implement KNN (K-Nearest Neighbour) Algorithm in any Language.

Theory: -

K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on **Supervised learning technique**. KNN 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. KNN 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.

KNN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems. KNN 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.

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.

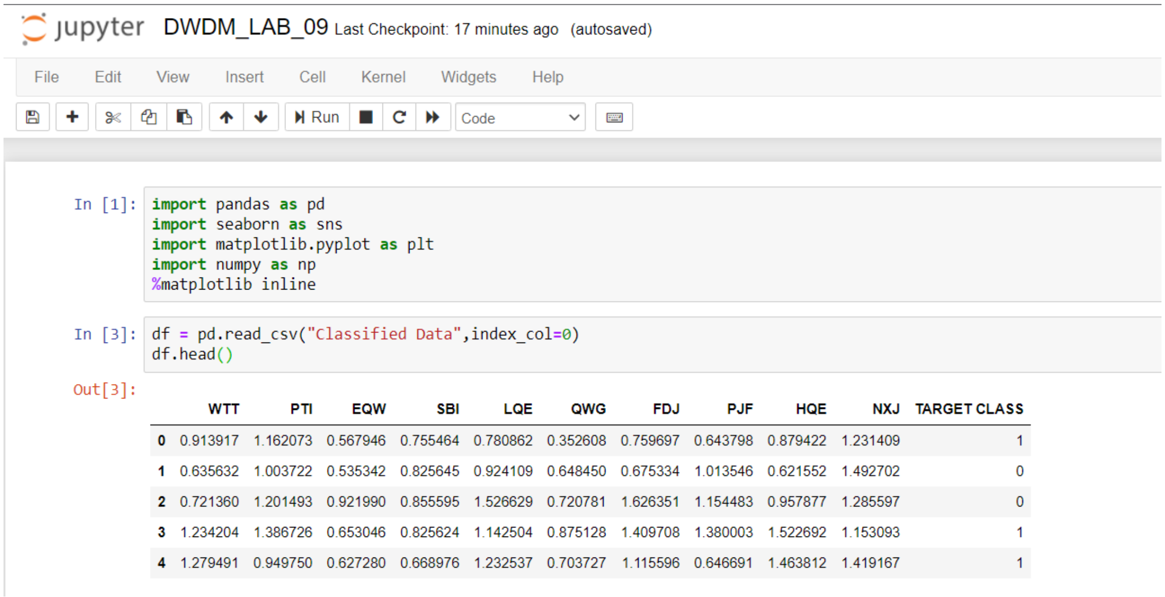
Note: I have used **"Classified Data"** for sample data in this experiment**.**

Source Code (in python):

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report,confusion_matrix
%matplotlib inline

df = pd.read_csv("Classified Data",index_col=0)
df.head()

scaler = StandardScaler()
scaler.fit(df.drop('TARGET CLASS',axis=1))
scaled_features = scaler.transform(df.drop('TARGET CLASS',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()

X_train, X_test, y_train, y_test = train_test_split(scaled_features,df['TARGET CLASS'],
                                                    test_size=0.30)

knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train,y_train)
pred = knn.predict(X_test)

print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))

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')
```

```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

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

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))

# NOW WITH K=23
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))
```

<u>OUTPUT-</u>

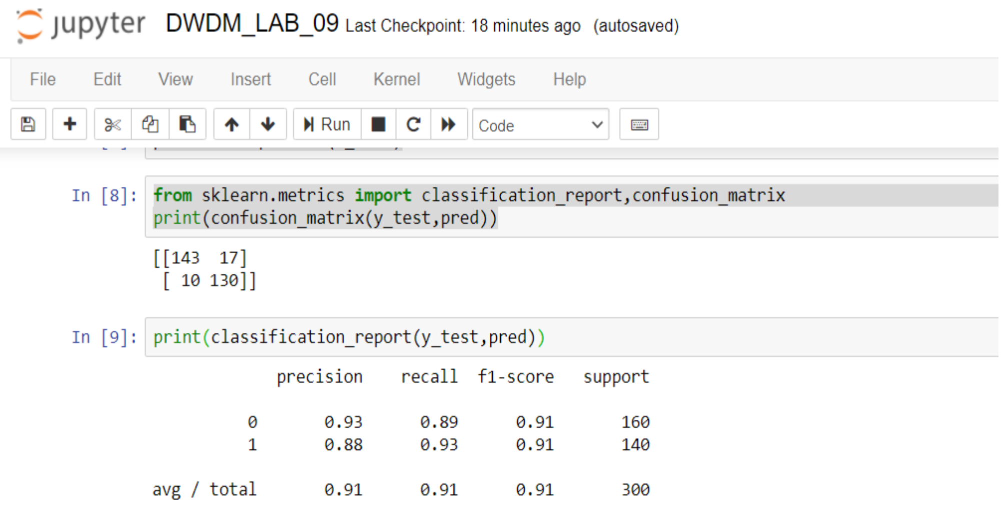


In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline
```

In [3]:
```
df = pd.read_csv("Classified Data",index_col=0)
df.head()
```

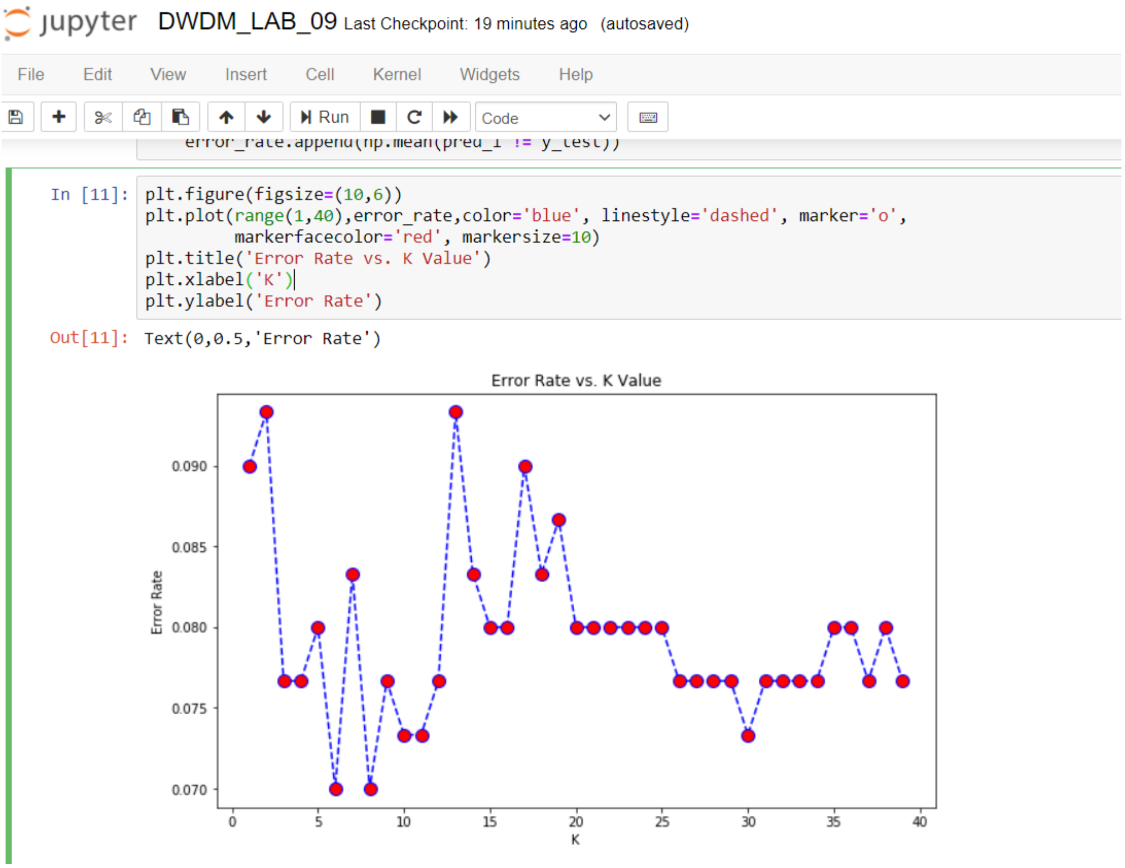
Out[3]:

	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


jupyter DWDM_LAB_09 Last Checkpoint: 18 minutes ago (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

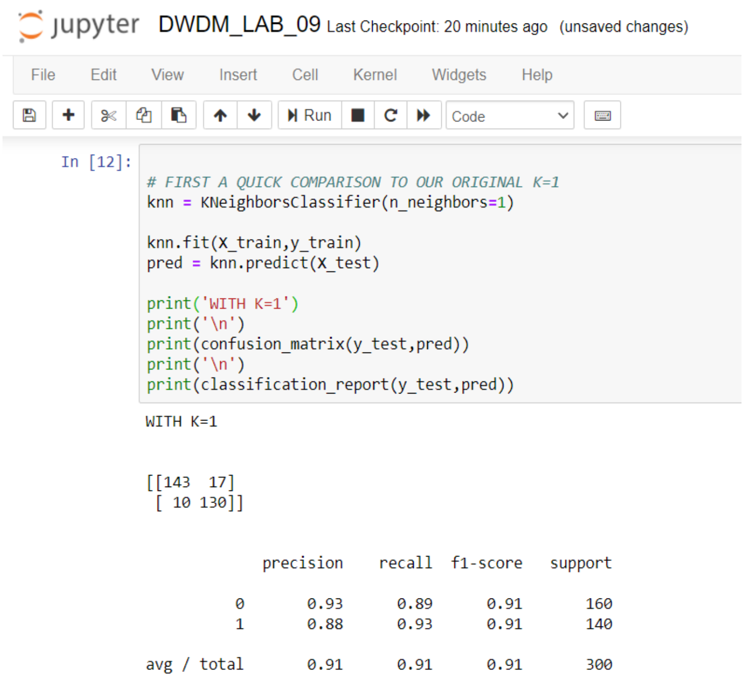
```
In [8]: from sklearn.metrics import classification_report,confusion_matrix
        print(confusion_matrix(y_test,pred))

        [[143  17]
         [ 10 130]]
```

```
In [9]: print(classification_report(y_test,pred))

                     precision    recall  f1-score   support

                  0       0.93      0.89      0.91       160
                  1       0.88      0.93      0.91       140

        avg / total       0.91      0.91      0.91       300
```



```
            error_rate.append(np.mean(pred_i != y_test))
```

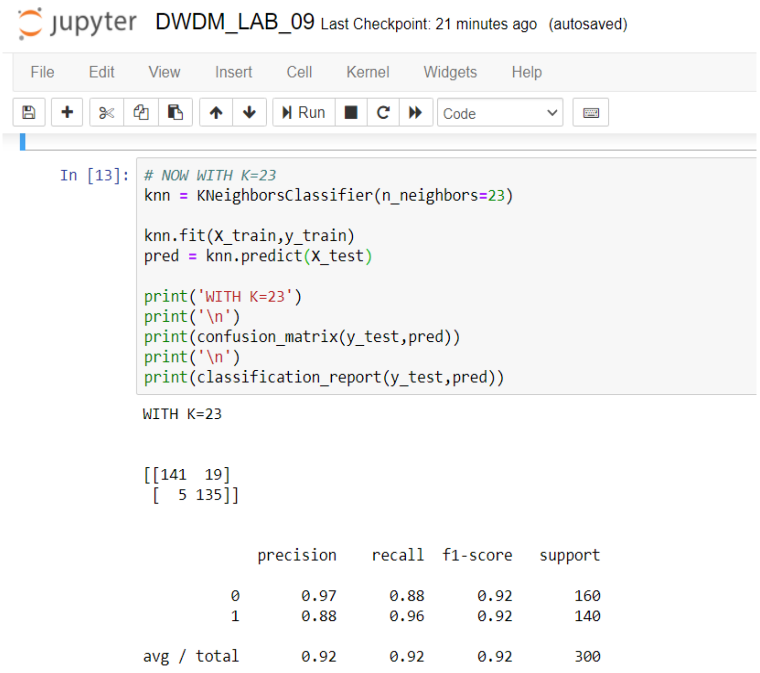
```
In [11]: plt.figure(figsize=(10,6))
         plt.plot(range(1,40),error_rate,color='blue', linestyle='dashed', marker='o',
                  markerfacecolor='red', markersize=10)
         plt.title('Error Rate vs. K Value')
         plt.xlabel('K')
         plt.ylabel('Error Rate')

Out[11]: Text(0,0.5,'Error Rate')
```


jupyter
DWDM_LAB_09 Last Checkpoint: 20 minutes ago (unsaved changes)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Run
Code

In [12]:

```
# FIRST A QUICK COMPARISON TO OUR ORIGINAL K=1
knn = KNeighborsClassifier(n_neighbors=1)

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

print('WITH K=1')
print('\n')
print(confusion_matrix(y_test,pred))
print('\n')
print(classification_report(y_test,pred))
```

```
WITH K=1


[[143  17]
 [ 10 130]]


             precision    recall  f1-score   support

          0       0.93      0.89      0.91       160
          1       0.88      0.93      0.91       140

avg / total       0.91      0.91      0.91       300
```



In [13]:

```
# NOW WITH K=23
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))
```

```
WITH K=23


[[141  19]
 [  5 135]]


             precision    recall  f1-score   support

          0       0.97      0.88      0.92       160
          1       0.88      0.96      0.92       140

avg / total       0.92      0.92      0.92       300
```

Findings and Learning:

- We have successfully implemented KNN Algorithm in Python.
- We have learnt about the applications, strengths and weaknesses of KNN algorithm.