COMPBEB1326
72017912C

Page No.
Date

GROUP B : ASSIGNMENT No. 5

Title: K-Nearest Neighbour Algorithm

Objective: To implement K-Nearest Neighbour algorithm on given dataset.

Problem Statement:

Implement K-Nearest Neighbours algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision & recall on the given dataset.

Hardware & Software Requirement:

1) PC / Laptop
2) Any Operating System
3) Python
4) Jupyter Notebook

Theory:

K-Nearest Neighbour (KNN):

1) K-Nearest Neighbour is one of the simplest Machine learning algorithms based on Supervised Learning Technique
2) KNN algorithm assumes the similarity between the new case/data & available cases & put the new case into the category that is most similar to the available categories.
3) KNN algorithm stores all the available data & classifies a new data point based on the similarity. This means when a new data appears then it can be easily

classified into a well suite category by using KNN algorithm.

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

5> KNN is a non-perrametric algorithm, which means it does not make any assumptions an underlying data.

6> It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset & at the time af classification, it performs an action an the dataset.

7> KNN algorithm at the training phase just stores the dataset & when it gets new data, then it classifies that data into a category that is much similar to the new data.

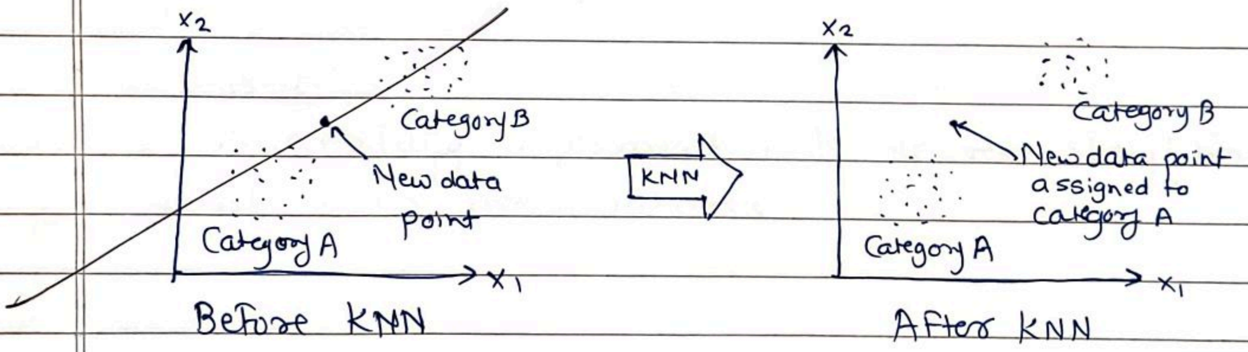
Need af KNN Algorithm:

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

Page No.
Date

Algorithm :

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



Before KNN After KNN

Euclidean Distance $= \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

Selecting value af k :

1) There is no particular way to determine the best value for "K", so we need to try some values to find the best out af them. The most preferred value for K is 5.

2) A very low value for K such as K=1 or K=2, can be noisy & lead to the effects af outliers in the model.

3) Large values for K are good, but it may find some difficulties.

Advantages:

1) It is simple to implement.
2) It is robust to the noisy training data.
3) It can be more effective if the training data is large.

Disadvantages:

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

Conclusion:

Successfully implemented K-Nearest Neighbour algorithm on given dataset.

17/10/22

In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
```

In [2]:
```
df=pd.read_csv('diabetes.csv')
```

In [3]:
```
df.columns
```

Out[3]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'Pedigree', 'Age', 'Outcome'],
      dtype='object')
```

Check for null values. If present remove null values from the dataset

In [4]:
```
df.isnull().sum()
```

Out[4]:
```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

In [ ]:

Outcome is the label/target, other columns are features

In [5]:
```
X = df.drop('Outcome',axis = 1)
y = df['Outcome']
```

In [6]:
```
from sklearn.preprocessing import scale
X = scale(X)
# split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, rando
```

In [7]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
```

```
In [8]: print("Confusion matrix: ")
        cs = metrics.confusion_matrix(y_test,y_pred)
        print(cs)
```

```
Confusion matrix:
[[123  28]
 [ 37  43]]
```

```
In [9]: print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

```
Acccuracy  0.7186147186147186
```

Classification error rate: proportion of instances misclassified over the whole set of instances. Error rate is calculated as the total number of two incorrect predictions (FN + FP) divided by the total number of a dataset (examples in the dataset.

Also error_rate = 1- accuracy

```
In [10]: total_misclassified = cs[0,1] + cs[1,0]
         print(total_misclassified)
         total_examples = cs[0,0]+cs[0,1]+cs[1,0]+cs[1,1]
         print(total_examples)
         print("Error rate",total_misclassified/total_examples)
         print("Error rate ",1-metrics.accuracy_score(y_test,y_pred))
```

```
65
231
Error rate 0.2813852813852814
Error rate  0.2813852813852814
```

```
In [11]: print("Precision score",metrics.precision_score(y_test,y_pred))
```

```
Precision score 0.6056338028169014
```

```
In [12]: print("Recall score ",metrics.recall_score(y_test,y_pred))
```

```
Recall score  0.5375
```

```
In [13]: print("Classification report ",metrics.classification_report(y_test,y_pred))
```

```
Classification report                precision    recall  f1-score   support

           0       0.77      0.81      0.79       151
           1       0.61      0.54      0.57        80

    accuracy                           0.72       231
   macro avg       0.69      0.68      0.68       231
weighted avg       0.71      0.72      0.71       231
```