Group B

Experiment No: 5

Aim: Implement K-Nearest Neighbors algorithm on diabetes.csv dataset. Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.

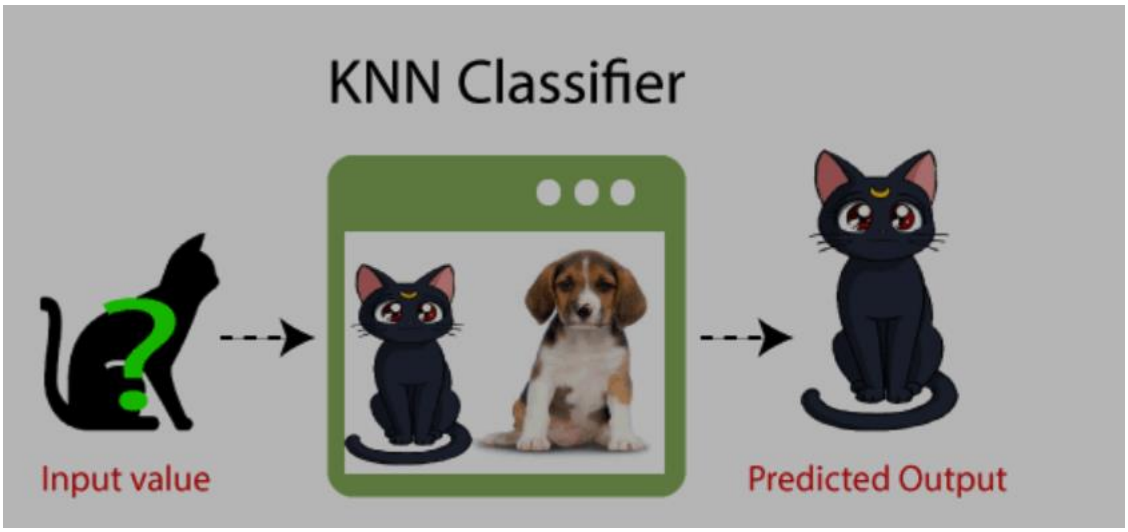
Theory:

K-Nearest Neighbor (KNN) Algorithm

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 available cases and put the new case into the category that is most similar to the available categories. 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 training set 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.



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**

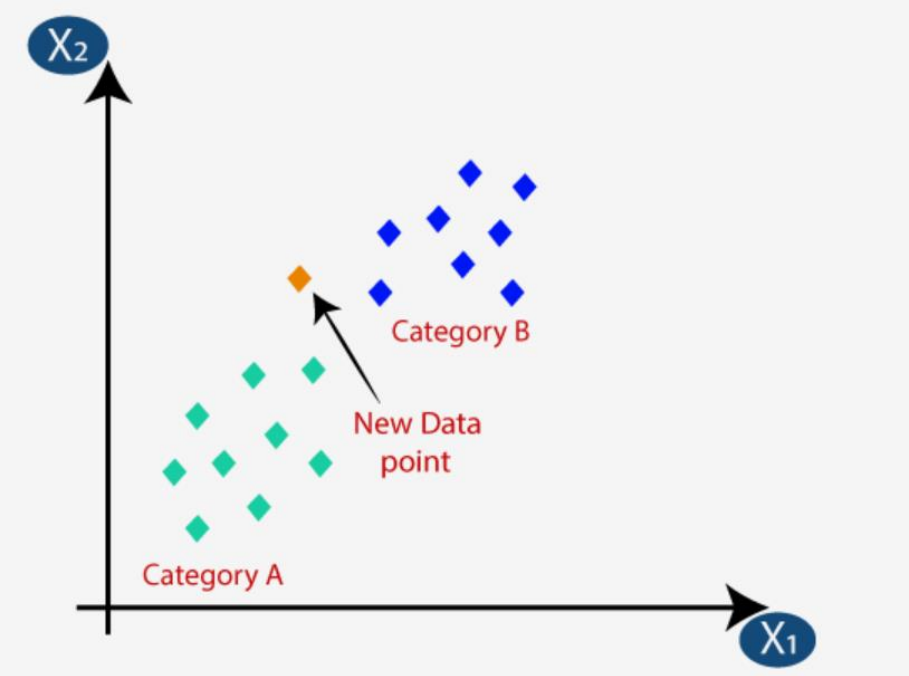
Step-3: Take the K nearest neighbors as per the calculated Euclidean distance.

Step-4: Among these k neighbors, count the number of the data points in each category.

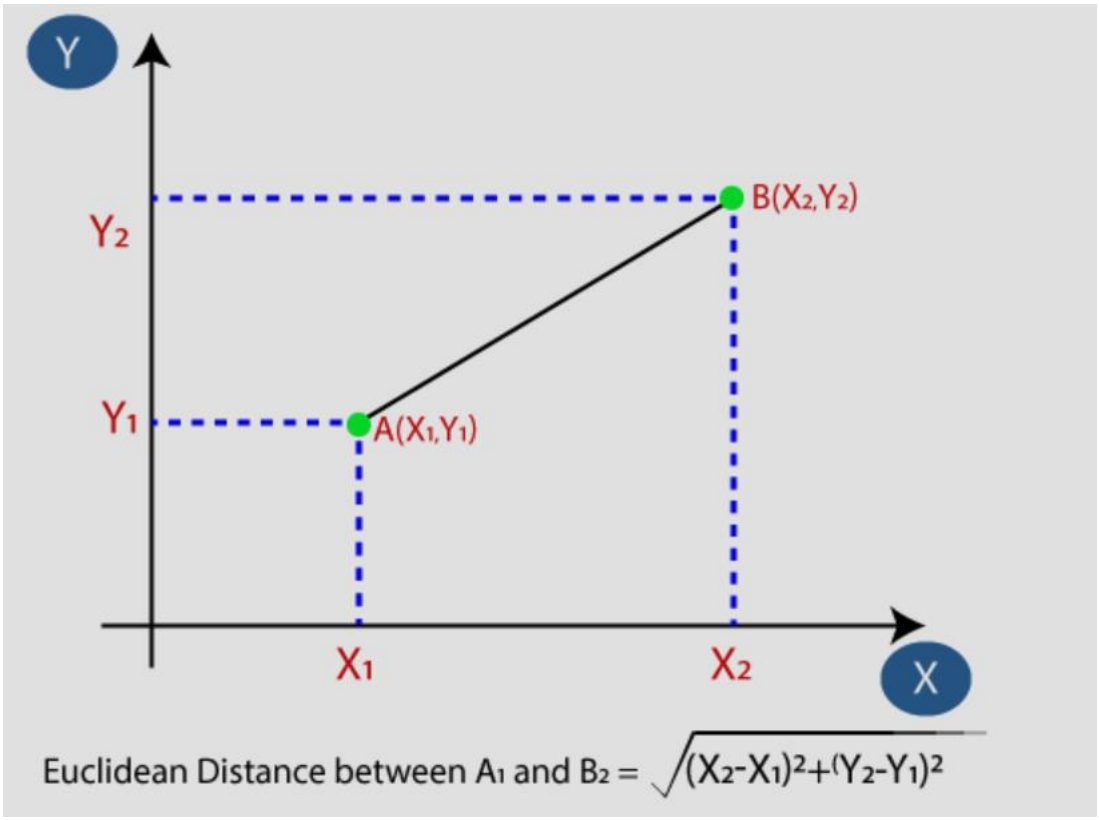
Step-5: Assign the new data points to that category for which the number of the neighbor is maximum.

Step-6: Our model is ready.

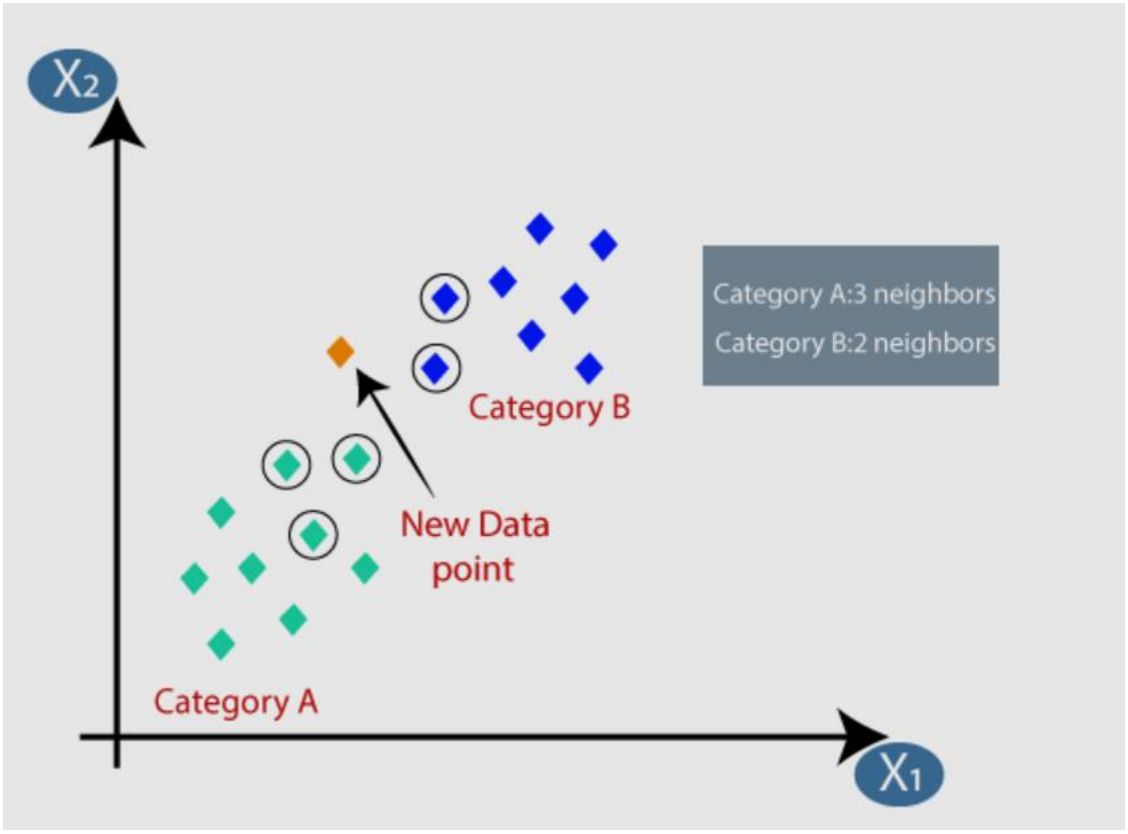
Suppose we have a new data point and we need to put it in the required category. Consider the below image:



- 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:



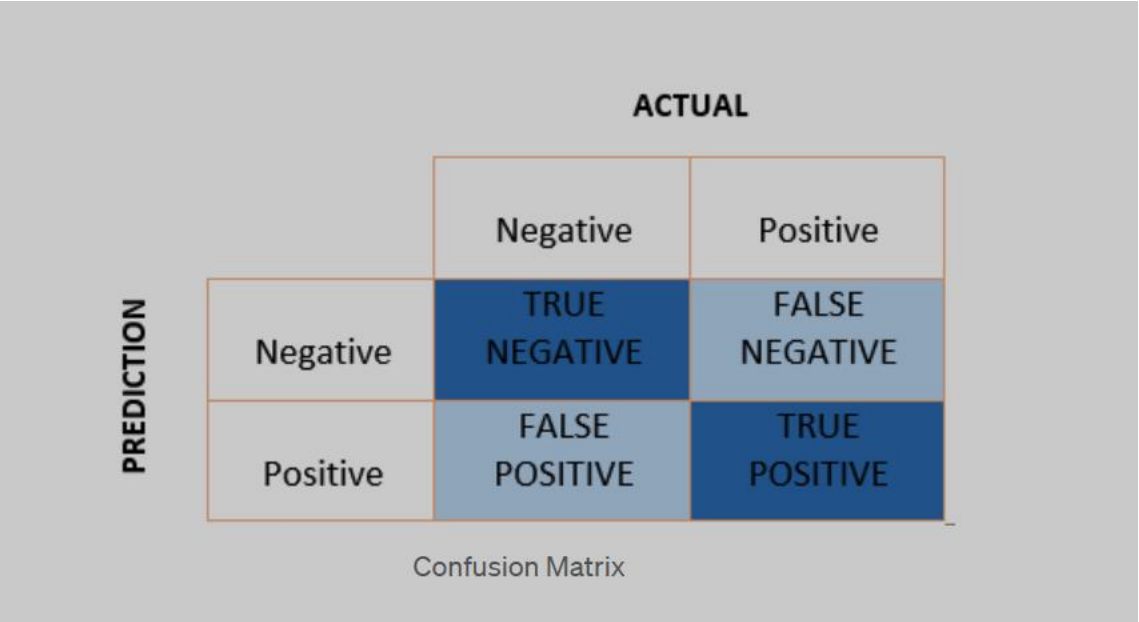
By calculating the Euclidean distance we got the nearest neighbors, as three nearest neighbors in category A and two nearest neighbors in category B. Consider the below image:



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

confusion matrix

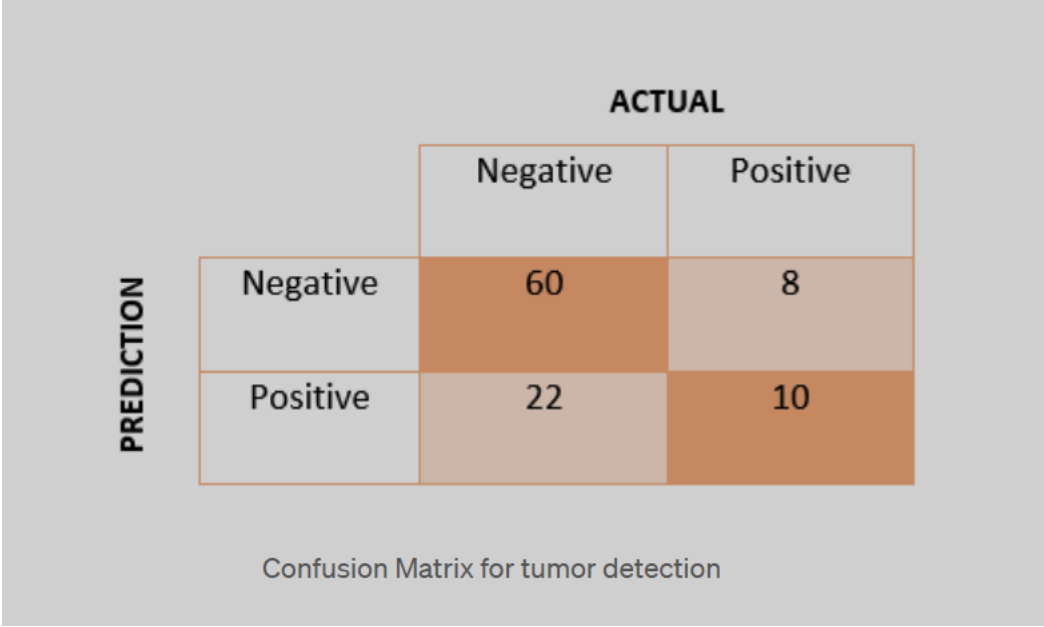
It is a matrix of size 2×2 for binary classification with actual values on one axis and predicted on another.



Confusion Matrix

EXAMPLE

A machine learning model is trained to predict tumor in patients. The test dataset consists of 100 people.



Confusion Matrix for tumor detection

True Positive (TP): model correctly predicts the positive class (prediction and actual both are positive). In the above example, **10 people** who have tumors are predicted positively by the model.

True Negative (TN):model correctly predicts the negative class (prediction and actual both are negative). In the above example, **60 people** who don't have tumors are predicted negatively by the model.

False Positive (FP) — model gives the wrong prediction of the negative class (predicted-positive, actual-negative). In the above example, **22 people** are predicted as positive of having a tumor, although they don't have a tumor. FP is also called a **TYPE I** error.

False Negative (FN) — model wrongly predicts the positive class (predicted-negative, actual-positive). In the above example, **8 people** who have tumors are predicted as negative. FN is also called a **TYPE II** error. With the help of these four values, we can calculate True Positive Rate (TPR), False Negative Rate (FPR), True Negative Rate (TNR), and False Negative Rate (FNR).

$$TPR = \frac{TP}{Actual\ Positive} = \frac{TP}{TP + FN}$$

$$FNR = \frac{FN}{Actual\ Positive} = \frac{FN}{TP + FN}$$

$$TNR = \frac{TN}{Actual\ Negative} = \frac{TN}{TN + FP}$$

$$FPR = \frac{FP}{Actual\ Negative} = \frac{FP}{TN + FP}$$

Precision: Out of all the positive predicted, what percentage is truly positive.

$$Precision = \frac{TP}{TP + FP}$$

The precision value lies between 0 and 1.

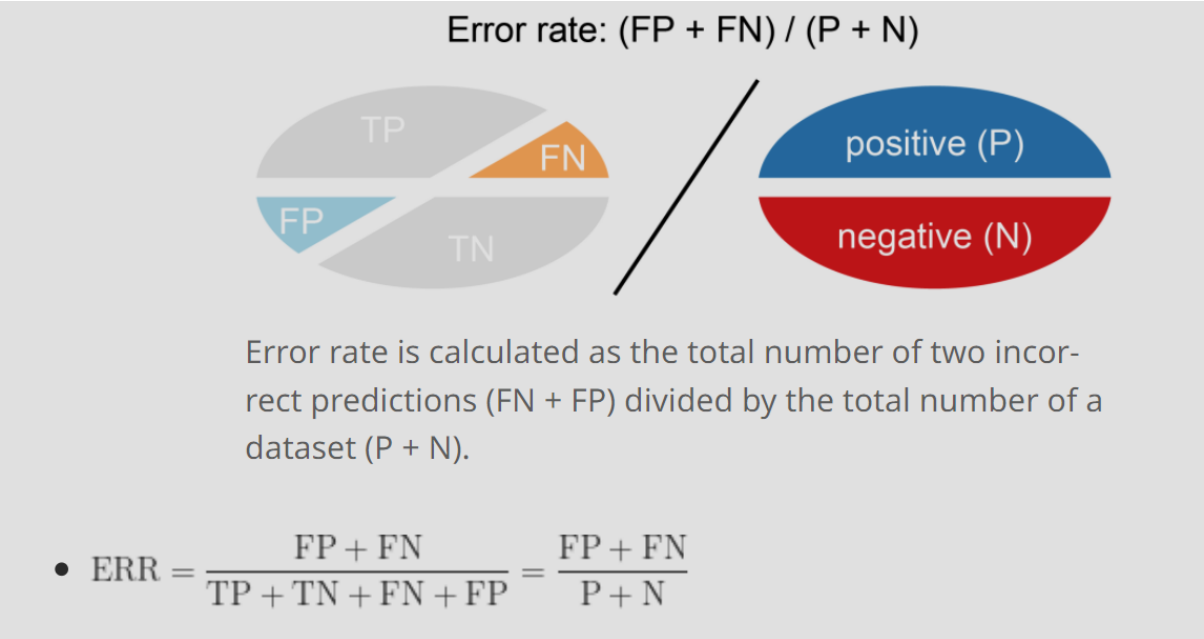
Recall

Out of the total positive, what percentage are predicted positive. It is the same as TPR (true positive rate).

$$Recall = \frac{TP}{TP + FN}$$

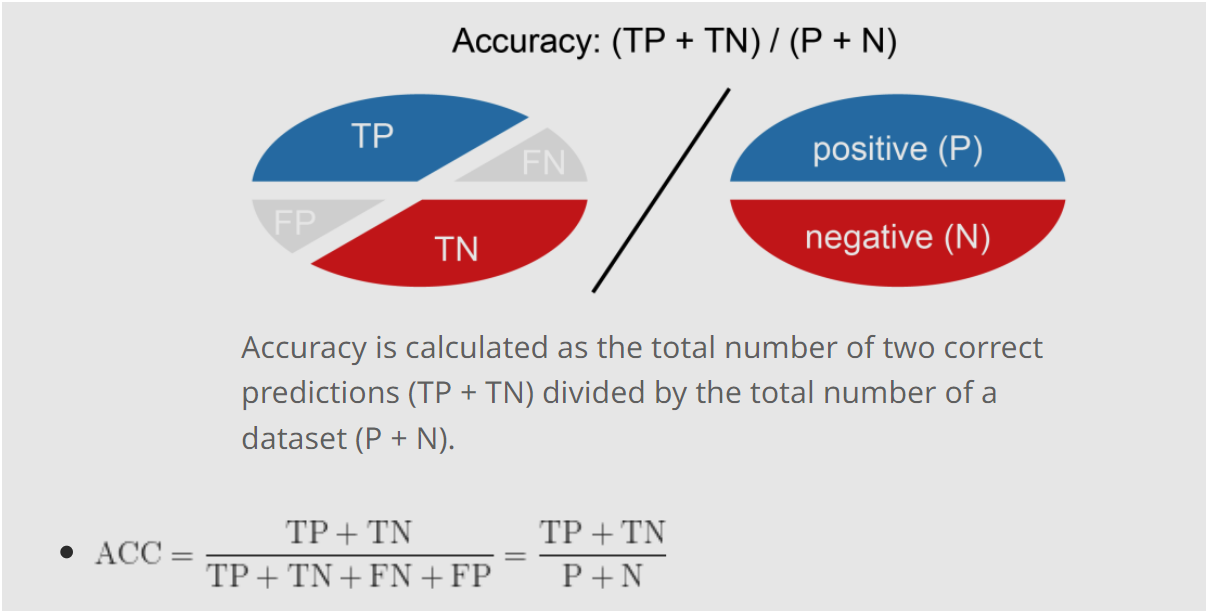
Error rate:

Error rate (ERR) is calculated as the number of all incorrect predictions divided by the total number of the dataset. The best error rate is 0.0, whereas the worst is 1.0.



Accuracy

Accuracy (ACC) is calculated as the number of all correct predictions divided by the total number of the dataset. The best accuracy is 1.0, whereas the worst is 0.0. It can also be calculated by 1 – ERR.



Accuracy is calculated as the total number of two correct predictions (TP + TN) divided by the total number of a dataset (P + N).

- $\mathrm{ACC} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{TP} + \mathrm{TN} + \mathrm{FN} + \mathrm{FP}} = \frac{\mathrm{TP} + \mathrm{TN}}{\mathrm{P} + \mathrm{N}}$

//code

//KNN algorithm on diabetes dataset

```
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

df=pd.read_csv('diabetes.csv')
df.columns
df.isnull().sum()
```

Outcome is the label/target, other columns are features

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

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, random_state = 42)
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)

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

print("Confusion matrix: ")
cs = metrics.confusion_matrix(y_test,y_pred)
print(cs)

print("Acccuracy ",metrics.accuracy_score(y_test,y_pred))
```

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

```
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))


print("Precision score",metrics.precision_score(y_test,y_pred))
print("Recall score ",metrics.recall_score(y_test,y_pred))
print("Classification report ",metrics.classification_report(y_test,y_p
red))
```

Conclusion: In this way we have successfully applied K-Nearest Neighbor Algorithm on Diabetes dataset & computed confusion matrix, accuracy, error rate, precision and recall on the given dataset.