1. **DTC with LDA algo having test\_size=0.2 and n\_component=1:**

Code:

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('diabetes\_1.csv')

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

from sklearn.impute import SimpleImputer

imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')

imputer.fit(X[:, 1:11])

X[:, 1:11] = imputer.transform(X[:, 1:11])

# Splitting the dataset into the Training set and Test set

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Applying LDA

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA

lda = LDA(n\_components=1)

X\_train = lda.fit\_transform(X\_train, y\_train)

X\_test = lda.transform(X\_test)

#DTC

from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(criterion='entropy', random\_state=0)

classifier.fit(X\_train, y\_train)

y\_pred=classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix, accuracy\_score

cm = confusion\_matrix(y\_test, y\_pred)

print(cm)

accuracy\_score(y\_test, y\_pred)

from sklearn.metrics import classification\_report

#print ('Accuracy for Logistic Regression Classifier :', accuracy\_score(y\_test, y\_pred))

#print ('\n confussion matrix for Logistic Regression Classifier:\n',confusion\_matrix(y\_test, y\_pred))

print(classification\_report(y\_test, y\_pred))

# True Positive (TP): we predict a label of 1 (positive), and the true label is 1.

tp = np.sum(np.logical\_and(y\_test == 1, y\_pred == 1))

# True Negative (TN): we predict a label of 0 (negative), and the true label is 0.

tn = np.sum(np.logical\_and(y\_test == 0, y\_pred == 0))

# False Positive (FP): we predict a label of 1 (positive), but the true label is 0.

fp = np.sum(np.logical\_and(y\_test == 1, y\_pred == 0))

# False Negative (FN): we predict a label of 0 (negative), but the true label is 1.

fn = np.sum(np.logical\_and(y\_test == 0, y\_pred== 1))

# Sensitivity, hit rate, recall, or true positive rate

tpr = tp/(tp+fn)

# Specificity or true negative rate

tnr = tn/(tn+fp)

# Precision or positive predictive value

ppv = tp/(tp+fp)

# Negative predictive value

npv = tn/(tn+fn)

# Fall out or false positive rate

fpr = fp/(fp+tn)

# False negative rate

fnr = fn/(tp+fn)

# False discovery rate

fdr = fp/(tp+fp)

# Overall accuracy

acc = (tp+tn)/(tp+fp+fn+tn)

if tp>0:

precision=float(tp)/(tp+fp)

recall=float(tp)/(tp+fn)

print ('\n confussion matrix for Logistic Regression Classifier:\n',confusion\_matrix(y\_test, y\_pred))

print('\nTrue Positive : %d'%(tp))

print('\nTrue Negative : %d'%(tn))

print('\nFalse Positive : %d'%(fp))

print('\nFalse Negative : %d'%(fn))

print('\nSensitivity, hit rate, recall, or true positive rate : %f' %(tpr))

print('\nSpecificity or true negative rate : %f' %(tnr))

print('\nPrecision or positive predictive value : %f' %(ppv))

print('\nNegative predictive value : %f'%(npv))

print('\nFall out or false positive rate: %f' %(fpr))

print('\nFalse negative rate : %f' %(fnr))

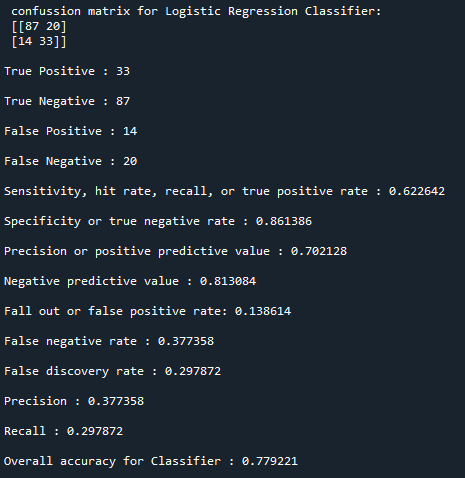
print('\nFalse discovery rate : %f' %(fdr))

print('\nPrecision : %f' %(fnr))

print('\nRecall : %f' %(fdr))

print('\nOverall accuracy for Classifier : %f' %(acc))

**OUTPUT:**



1. **Catboost with LDA algo having test\_size=0.4 and n\_component=1:**

**Code:**

**import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**# Importing the dataset**

**dataset = pd.read\_csv('diabetes\_1.csv')**

**X = dataset.iloc[:, :-1].values**

**y = dataset.iloc[:, -1].values**

**from sklearn.impute import SimpleImputer**

**imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')**

**imputer.fit(X[:, 1:8])**

**X[:, 1:8] = imputer.transform(X[:, 1:8])**

**# Splitting the dataset into the Training set and Test set**

**from sklearn.model\_selection import train\_test\_split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.4, random\_state = 0)**

**# Feature Scaling**

**from sklearn.preprocessing import StandardScaler**

**sc = StandardScaler()**

**X\_train = sc.fit\_transform(X\_train)**

**X\_test = sc.transform(X\_test)**

**#LDA**

**from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA**

**lda = LDA(n\_components=1)**

**X\_train = lda.fit\_transform(X\_train, y\_train)**

**X\_test = lda.transform(X\_test)**

**#CatBoost**

**from catboost import CatBoostClassifier**

**classifier = CatBoostClassifier()**

**classifier.fit(X\_train, y\_train)**

**# Predicting the Test set results**

**y\_pred = classifier.predict(X\_test)**

**# Making the Confusion Matrix**

**from sklearn.metrics import confusion\_matrix, accuracy\_score**

**cm = confusion\_matrix(y\_test, y\_pred)**

**print(cm)**

**accuracy\_score(y\_test, y\_pred)**

**from sklearn.metrics import classification\_report**

**print(classification\_report(y\_test, y\_pred))**

**# True Positive (TP): we predict a label of 1 (positive), and the true label is 1.**

**tp = np.sum(np.logical\_and(y\_test == 1, y\_pred == 1))**

**# True Negative (TN): we predict a label of 0 (negative), and the true label is 0.**

**tn = np.sum(np.logical\_and(y\_test == 0, y\_pred == 0))**

**# False Positive (FP): we predict a label of 1 (positive), but the true label is 0.**

**fp = np.sum(np.logical\_and(y\_test == 1, y\_pred == 0))**

**# False Negative (FN): we predict a label of 0 (negative), but the true label is 1.**

**fn = np.sum(np.logical\_and(y\_test == 0, y\_pred== 1))**

**# Sensitivity, hit rate, recall, or true positive rate**

**tpr = tp/(tp+fn)**

**# Specificity or true negative rate**

**tnr = tn/(tn+fp)**

**# Precision or positive predictive value**

**ppv = tp/(tp+fp)**

**# Negative predictive value**

**npv = tn/(tn+fn)**

**# Fall out or false positive rate**

**fpr = fp/(fp+tn)**

**# False negative rate**

**fnr = fn/(tp+fn)**

**# False discovery rate**

**fdr = fp/(tp+fp)**

**# Overall accuracy**

**acc = (tp+tn)/(tp+fp+fn+tn)**

**if tp>0:**

**precision=float(tp)/(tp+fp)**

**recall=float(tp)/(tp+fn)**

**print ('\n confussion matrix for Logistic Regression Classifier:\n',confusion\_matrix(y\_test, y\_pred))**

**print('\nTrue Positive : %d'%(tp))**

**print('\nTrue Negative : %d'%(tn))**

**print('\nFalse Positive : %d'%(fp))**

**print('\nFalse Negative : %d'%(fn))**

**print('\nSensitivity, hit rate, recall, or true positive rate : %f' %(tpr))**

**print('\nSpecificity or true negative rate : %f' %(tnr))**

**print('\nPrecision or positive predictive value : %f' %(ppv))**

**print('\nNegative predictive value : %f'%(npv))**

**print('\nFall out or false positive rate: %f' %(fpr))**

**print('\nFalse negative rate : %f' %(fnr))**

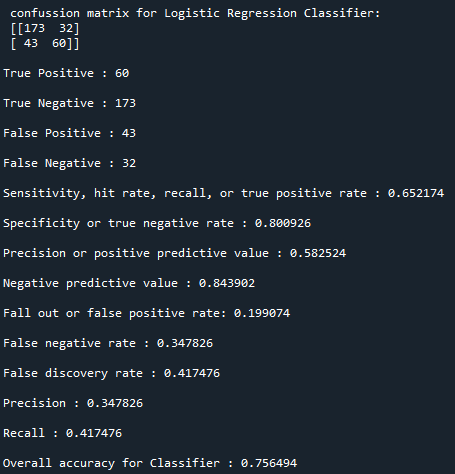
**print('\nFalse discovery rate : %f' %(fdr))**

**print('\nPrecision : %f' %(fnr))**

**print('\nRecall : %f' %(fdr))**

**print('\nOverall accuracy for Classifier : %f' %(acc))**

**OUTPUT:**

****

1. **Catboost with LDA algo having test\_size=0.6 and n\_component=1:**

**Code:  
import numpy as np**

**import matplotlib.pyplot as plt**

**import pandas as pd**

**# Importing the dataset**

**dataset = pd.read\_csv('diabetes\_1.csv')**

**X = dataset.iloc[:, :-1].values**

**y = dataset.iloc[:, -1].values**

**from sklearn.impute import SimpleImputer**

**imputer = SimpleImputer(missing\_values=np.nan, strategy='mean')**

**imputer.fit(X[:, 1:8])**

**X[:, 1:8] = imputer.transform(X[:, 1:8])**

**# Splitting the dataset into the Training set and Test set**

**from sklearn.model\_selection import train\_test\_split**

**X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.6, random\_state = 0)**

**# Feature Scaling**

**from sklearn.preprocessing import StandardScaler**

**sc = StandardScaler()**

**X\_train = sc.fit\_transform(X\_train)**

**X\_test = sc.transform(X\_test)**

**#LDA**

**from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis as LDA**

**lda = LDA(n\_components=1)**

**X\_train = lda.fit\_transform(X\_train, y\_train)**

**X\_test = lda.transform(X\_test)**

**#CatBoost**

**from catboost import CatBoostClassifier**

**classifier = CatBoostClassifier()**

**classifier.fit(X\_train, y\_train)**

**# Predicting the Test set results**

**y\_pred = classifier.predict(X\_test)**

**# Making the Confusion Matrix**

**from sklearn.metrics import confusion\_matrix, accuracy\_score**

**cm = confusion\_matrix(y\_test, y\_pred)**

**print(cm)**

**accuracy\_score(y\_test, y\_pred)**

**from sklearn.metrics import classification\_report**

**print(classification\_report(y\_test, y\_pred))**

**# True Positive (TP): we predict a label of 1 (positive), and the true label is 1.**

**tp = np.sum(np.logical\_and(y\_test == 1, y\_pred == 1))**

**# True Negative (TN): we predict a label of 0 (negative), and the true label is 0.**

**tn = np.sum(np.logical\_and(y\_test == 0, y\_pred == 0))**

**# False Positive (FP): we predict a label of 1 (positive), but the true label is 0.**

**fp = np.sum(np.logical\_and(y\_test == 1, y\_pred == 0))**

**# False Negative (FN): we predict a label of 0 (negative), but the true label is 1.**

**fn = np.sum(np.logical\_and(y\_test == 0, y\_pred== 1))**

**# Sensitivity, hit rate, recall, or true positive rate**

**tpr = tp/(tp+fn)**

**# Specificity or true negative rate**

**tnr = tn/(tn+fp)**

**# Precision or positive predictive value**

**ppv = tp/(tp+fp)**

**# Negative predictive value**

**npv = tn/(tn+fn)**

**# Fall out or false positive rate**

**fpr = fp/(fp+tn)**

**# False negative rate**

**fnr = fn/(tp+fn)**

**# False discovery rate**

**fdr = fp/(tp+fp)**

**# Overall accuracy**

**acc = (tp+tn)/(tp+fp+fn+tn)**

**if tp>0:**

**precision=float(tp)/(tp+fp)**

**recall=float(tp)/(tp+fn)**

**print ('\n confussion matrix for Logistic Regression Classifier:\n',confusion\_matrix(y\_test, y\_pred))**

**print('\nTrue Positive : %d'%(tp))**

**print('\nTrue Negative : %d'%(tn))**

**print('\nFalse Positive : %d'%(fp))**

**print('\nFalse Negative : %d'%(fn))**

**print('\nSensitivity, hit rate, recall, or true positive rate : %f' %(tpr))**

**print('\nSpecificity or true negative rate : %f' %(tnr))**

**print('\nPrecision or positive predictive value : %f' %(ppv))**

**print('\nNegative predictive value : %f'%(npv))**

**print('\nFall out or false positive rate: %f' %(fpr))**

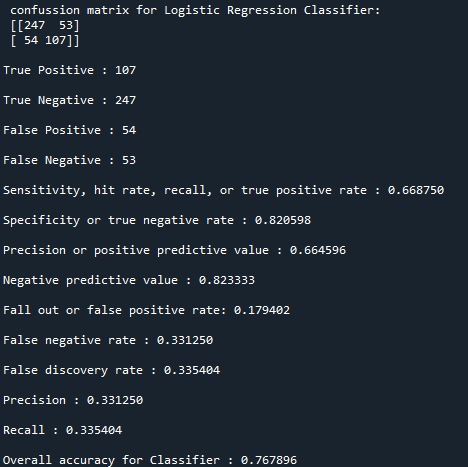
**print('\nFalse negative rate : %f' %(fnr))**

**print('\nFalse discovery rate : %f' %(fdr))**

**print('\nPrecision : %f' %(fnr))**

**print('\nRecall : %f' %(fdr))**

**print('\nOverall accuracy for Classifier : %f' %(acc))**

**OUTPUT:  
**