In [72]:

```
#import numpy
import numpy as np
#import gaussian nb from sklearn
from sklearn.naive_bayes import GaussianNB
#import the normalizationmodule
from sklearn.preprocessing import StandardScaler
#import pre processing library
from sklearn.decomposition import PCA
#import pipilining module to execute stepwise
from sklearn.pipeline import make_pipeline
#imprt modules to get the metrics of the algorithm
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, classification_report
#import library for class balancing
from imblearn.over_sampling import SMOTE
#import for splitting of training data
from sklearn.model_selection import train_test_split
# Load training and testing data
X = np.loadtxt('X_train.csv', delimiter=',', skiprows=1)
X_test_actual = np.loadtxt('X_test.csv', delimiter=',', skiprows=1)
y = np.loadtxt('y_train.csv', delimiter=',', skiprows=1)[:, 1]
#varible for class balancing
sm = SMOTE()
#divide the training data and testing data without shuffling the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, shuffle=False)
#fit the sample and balance the class for calculating the metrics on the algorithm
X_train_res, y_train_res = sm.fit_sample(X_train, y_train)
#fit on the actual data to balance them for actual
X_train_res_actual, y_train_res_actual = sm.fit_sample(X, y)
#maintain 81% of the variance in the output and select the optimum number of components
pca = PCA(0.81)
#make a pipeline of steps -> first normalize the data, pre-process the data and then perform the gaussian naive bayes
gnb = make_pipeline(StandardScaler(), pca.fit(X_test), GaussianNB())
#same as above line for the actual classification to submit
gnb_actual = make_pipeline(StandardScaler(), pca.fit(X_test_actual), GaussianNB())
#train the X and Y data and then test it on the X_test to get final output file
y_pred = gnb.fit(X_train_res, y_train_res).predict(X_test)
#same as above line for the actual training and prediction
y_pred_actual = gnb_actual.fit(X_train_res_actual, y_train_res_actual).predict(X_test_actual)
#prepare the data to write to csv file - both for metrics and csv
n_points = X_test.shape[0]
n_points_actual = X_test_actual.shape[0]
```

```
#get third column to the csv file
y_pred_pp_actual = np.ones((n_points_actual, 2))
#first column is all X_test columns
y_pred_pp_actual[:, 0] = range(n_points_actual)
#second column is the predicted class
y_pred_pp_actual[:, 1] = y_pred_actual
#calculate accuracy, f1 score and the confusion matrix based on the sample split up test data
print("Accuracy: {:.2f}%".format(accuracy_score(y_test, y_pred) * 100))
print("F1 Score: {:.2f}".format(f1_score(y_test, y_pred) * 100))
print("COnfusion Matrix:\n", confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
#create the file with the delimeter ',' and remove the '#' from the header in the file
np.savetxt('Gaussian-Naive-Bayes-Final-Output.csv', y_pred_pp_actual, fmt='%d', delimiter=",", header="Id,EpiOrStroma", c
```

```
Accuracy: 90.56%
F1 Score: 87.02
COnfusion Matrix:
 [[ 57   8]
 [  9 106]]
              precision    recall  f1-score   support

         1.0       0.86      0.88      0.87        65
         2.0       0.93      0.92      0.93       115

   micro avg       0.91      0.91      0.91       180
   macro avg       0.90      0.90      0.90       180
weighted avg       0.91      0.91      0.91       180
```

##

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```