VEHICLE LOAN PREDICTION

Text

Description automatically generated

INDIVIDUAL REPORT BY,

Mrunalini Devineni

DATS6103

Professor Amir Jafari

December 6, 2021

INTRODUCTION

Financial institutions incur significant losses due to the default of Vehicle Loans. This situation has led to the constricting of vehicle loan underwriting and increased vehicle loan rejection rates. These institutions also raise the need for a better credit risk scoring model. By doing this, the institutions are trying to accurately predict the Probability of loanee defaulting on a vehicle loan on the due date. In this sense, the credit scores are significant as a tool that can decide which clients can take or not a credit, considering their unique characteristics. The scoring is also a helpful tool for the clients because this can avoid accepting a loan that will not be able to be paid in the future and, consequently, prevent having problems with financial institutions.

Furthermore, the scoring tool is helpful not only for cash loans but also for mortgages and car loans, so this is the main reason that motivated us to choose this topic because it is helpful for many kinds of businesses.

To develop this scoring project, we will analyze an L&T car loan company database from Kaggle. Then we are doing some preprocessing and cleaning of the data to apply the classification models like Decision tree, Random Forest, logistic, and Gradient Boosting. Finally, we will show the results by using the Pyqt5.

The following report will be organized: first, we will describe the data set, then the methodology to be used, and finally, the results and main conclusions

DATA DESCRIPTION

For this study, we have chosen to analyze an L&T company dataset, in charge of cars' sale from kaggle.com. This dataset is like the one that financial institution must build the scoring models that allow them to forecast the approval or rejection of customers. This dataset does not require any cleaning and is equipped to fuel the analysis of this project. The base consists of 40 variables and 233 154 observations assessing a person's attributes ranging from demographic data (date of birth, etc.) and bureau data, like the amount of loan disbursed and the asset's cost.

The following Information regarding the loan and loanee are provided in the datasets:

* **Loanee Information** (Demographic data like age, Identity proof etc.)
* **Loan Information** (Disbursal details, loan to value ratio etc.)
* **Bureau data & history** (Bureau score, number of active accounts, the status of other loans, credit history etc.)

The Dataset available on Kaggle contains:

* Train.csv: a base that contains the training data with details on loan as described in the last section.
* data\_dictionary.csv: a base containing a brief description of each variable

provided in the training and test set.

Individual contribution

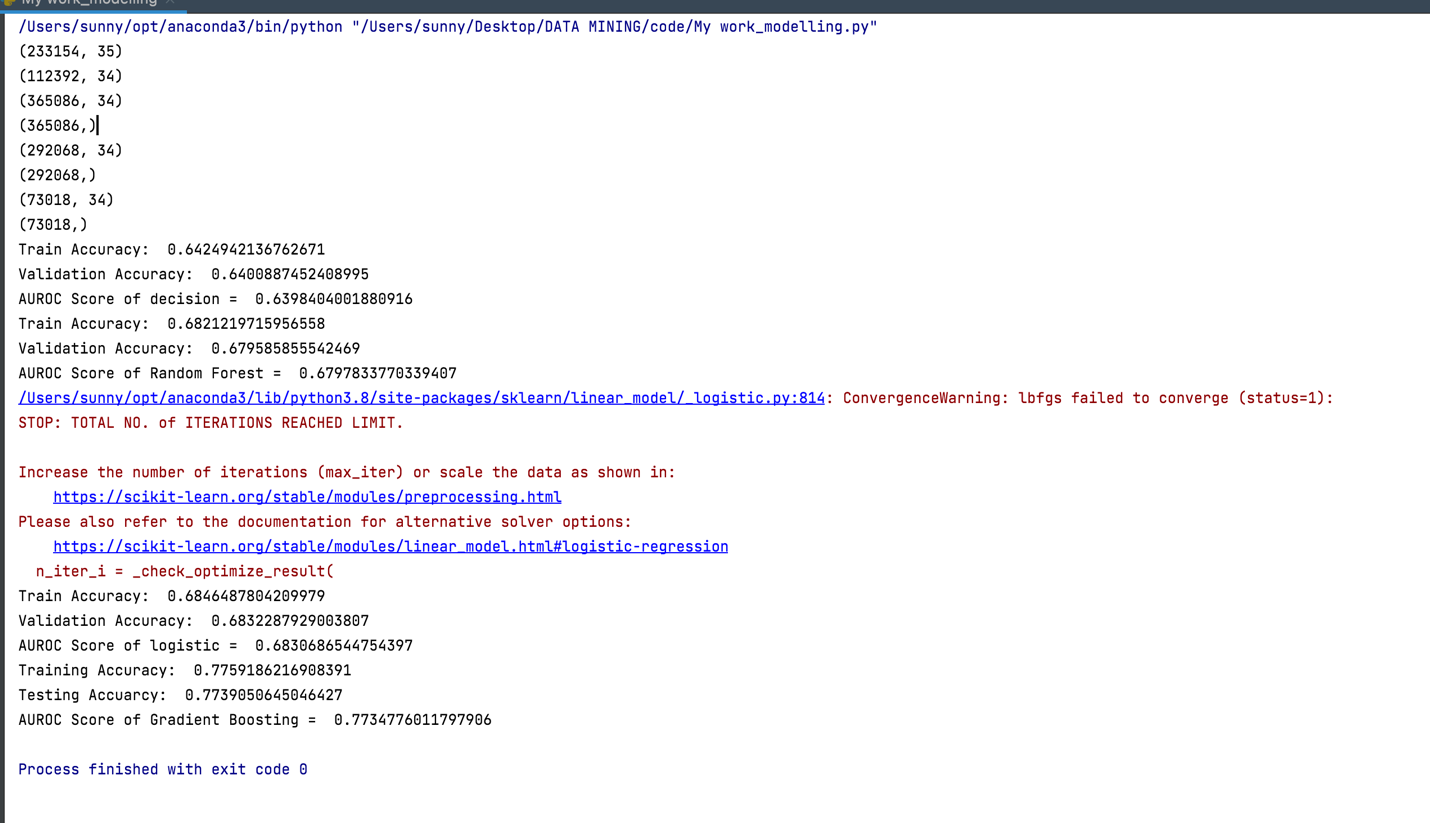
I have worked on every aspect of the project by contributing some percent of my work,

I have done a correlation plot of features in the EDA part, followed by developed a base line model that were implemented further using feature engineering, in addition to this contributed major part on the GUI by integrating correlation plot, Random Forest, Logistic Regression and Gradient boosting classifier using pyqt5 and finding cross validation report of above. To end with I have done the Data modelling section of the report.



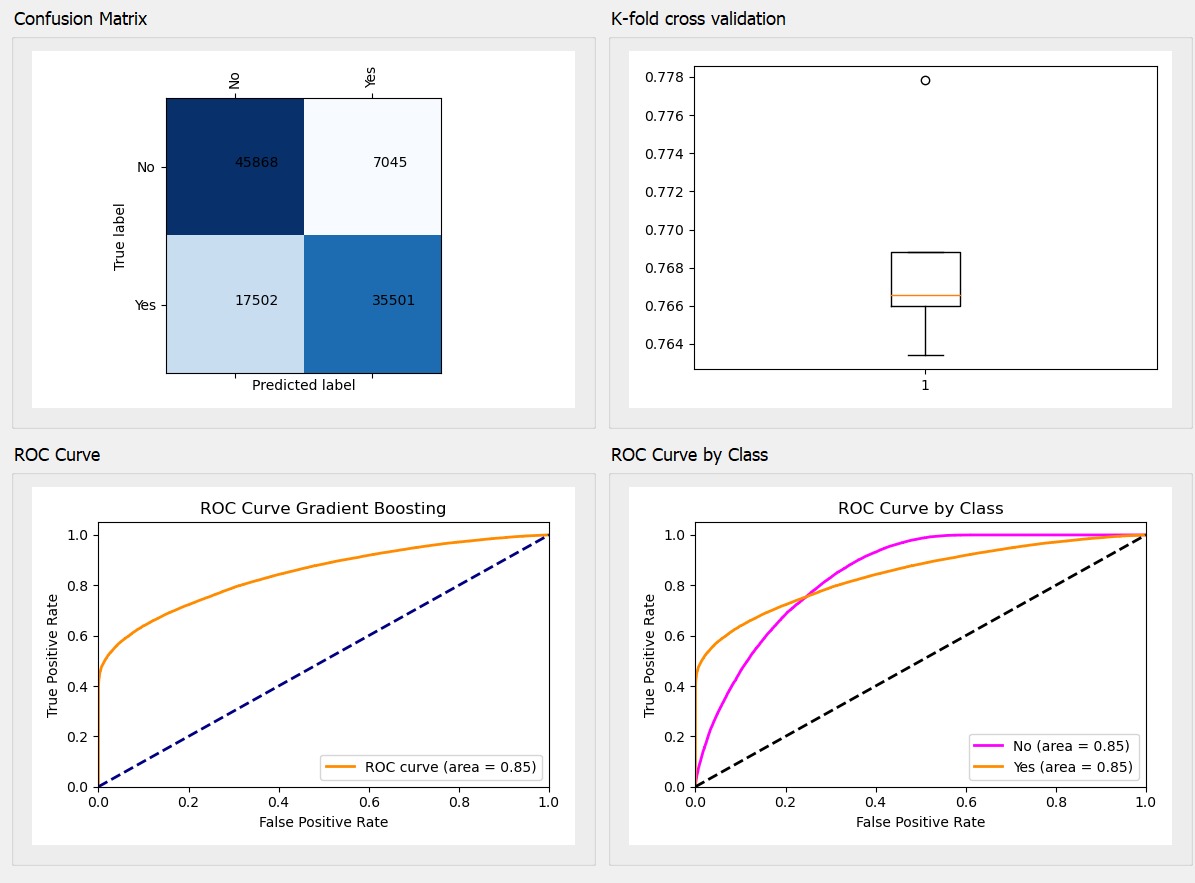
From this plot we could figure out the correlation among features of choice,

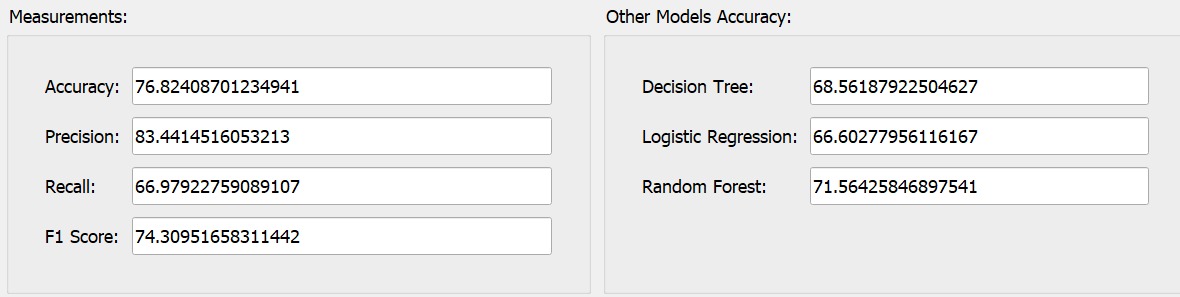
* One could select the features they want to and find out if they are correlated or not. Considering the entire data, we could know that there are features that are correlated for example, Asset cost is correlated with disbursed amount, sanctioned amount is correlated with disbursed amount.



The above are the results of the base model run:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| metrics | Decision tree | Random forest | Logistic Regression | Gradient Boosting |
| Train accuracy | 64.2% | 68.21% | 68.4% | 77.59% |
| Validation accuracy | 64.00% | 67.9% | 68.3% | 77.3% |
| AUCROC score | 63.9% | 67.9% | 68.3% | 77.34% |





The above is the GUI interface of the gradient model developed displaying the confusion matrix that shows there are 35501 of “yes” to loan default. The area under the curve of roc is the representation of the target predicted to be “yes”.

CONCLUSION

We have significantly improved the accuracy and precision of predicting a loan defaulter. We also found that Gradient Boosting gives the best prediction with a high accuracy of 76.82. Further, we can improve the accuracy of this data by applying PCA or other feature selection techniques. We can also use other ensemble methods to get a better result

REFERRENCES

[1] 2019. LT Vehicle Loan Default Prediction. (2019). <https://www.kaggle.com/mamtadhaker/lt-vehicle-loan-defaultpredictiondata_dictionary.csv>

[2] T. Cover &P. Hart Mickey Haggblade.Nearest neighbor pattern classification, IEEE Transactions on Information Theory 2013.

[3] C.J.C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition,” submitted to Data Mining and Knowledge Discovery, 1998.

[4] Lloyd, Stuart P. "Least squares quantization in PCM." Information Theory, IEEE Transactions on 28.2 (1982): 129-137.

[5] Nitesh V. Chawla, Kevin W. Bowyer. SMOTE: synthetic minority oversampling technique, Journal of Artificial Intelligence Research Archive Volume 16 Issue 1, January 2002

[6] Herve Abdi, Lynne J. Williams. Principal component analysis, WIREs Computational Statistics, 2010

[7] Aadhar, <https://uidai.gov.in/>

APPENDIX:

Base model :

**from** sklearn.metrics **import** classification\_report  
**from** sklearn.metrics **import** confusion\_matrix  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.svm **import** SVC  
**from** sklearn.metrics **import** roc\_auc\_score  
**from** sklearn.metrics **import** roc\_curve, auc, log\_loss, brier\_score\_loss  
**from** sklearn.calibration **import** calibration\_curve  
**from** sklearn.linear\_model **import** LogisticRegression  
**from** sklearn **import** feature\_selection  
**from** sklearn **import** metrics  
**from** sklearn.preprocessing **import** label\_binarize  
**from** sklearn.model\_selection **import** cross\_val\_predict  
**import** matplotlib.pyplot **as** plt  
**import** numpy **as** np  
**from** sklearn.decomposition **import** PCA  
  
train = pd.read\_csv(**"train.csv"**)  
test = pd.read\_csv(**"test.csv"**) *#uploaded to Google Colab directly  
  
# Looking at the data headers, these values aren't required  
  
#feature to drop here*train = train.drop([**'UniqueID'**, **'supplier\_id'**, **'Current\_pincode\_ID'**, **'Date.of.Birth'**, **'DisbursalDate'**, **'Employee\_code\_ID'**], axis = 1)  
  
test = test.drop([**'UniqueID'**, **'supplier\_id'**, **'Current\_pincode\_ID'**, **'Date.of.Birth'**, **'DisbursalDate'**, **'Employee\_code\_ID'**], axis = 1)  
  
  
  
print(train.shape)  
print(test.shape)  
  
  
Y = train.iloc[:, -1] *#last column is the the prediction in the training set*Y.shape  
  
X = train.drop([**'loan\_default'**], axis = 1)  
  
X.shape  
  
test\_X = test.iloc[:,:]  
  
X.sample(3) *# Checking whether irrelevant rows are dropped or not*X[**'Employment.Type'**].fillna(**'Self employed'**, inplace = **True**)  
test\_X[**'Employment.Type'**].fillna(**'Self employed'**, inplace = **True**)  
  
X[**'Employment.Type'**].value\_counts()  
  
X[**'Employment.Type'**] = X[**'Employment.Type'**].replace((**'Unemployed'**, **'Salaried'**, **'Self employed'**), (0, 1, 2))  
test\_X[**'Employment.Type'**] = test\_X[**'Employment.Type'**].replace((**'Unemployed'**, **'Salaried'**, **'Self employed'**), (0, 1, 2))  
  
X[**'Employment.Type'**].value\_counts() *#Converted irrelevant strings to numbers for computations while training*X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].value\_counts()  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'No Bureau History Available'**,  
 **'Not Scored: Sufficient History Not Available'**,**'Not Scored: Not Enough Info available on the customer'**,  
 **'Not Scored: No Activity seen on the customer (Inactive)'**,  
 **'Not Scored: No Updates available in last 36 months'**, **'Not Scored: Only a Guarantor'**,  
 **'Not Scored: More than 50 active Accounts found'**),(0, 0, 0, 0, 0, 0, 0))  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'L-Very High Risk'**, **'M-Very High Risk'**), (1, 1))  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'J-High Risk'**, **'K-High Risk'**), (2, 2))  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'H-Medium Risk'**, **'I-Medium Risk'**), (3, 3))  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'E-Low Risk'**, **'F-Low Risk'**, **'G-Low Risk'**), (4, 4, 4))  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'A-Very Low Risk'**, **'B-Very Low Risk'**,  
 **'C-Very Low Risk'**, **'D-Very Low Risk'**), (5, 5, 5, 5))  
  
X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].value\_counts()  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].value\_counts()  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'No Bureau History Available'**,  
 **'Not Scored: Sufficient History Not Available'**,**'Not Scored: Not Enough Info available on the customer'**,  
 **'Not Scored: No Activity seen on the customer (Inactive)'**,  
 **'Not Scored: No Updates available in last 36 months'**, **'Not Scored: Only a Guarantor'**,  
 **'Not Scored: More than 50 active Accounts found'**),(0, 0, 0, 0, 0, 0, 0))  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'L-Very High Risk'**, **'M-Very High Risk'**), (1, 1))  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'J-High Risk'**, **'K-High Risk'**), (2, 2))  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'H-Medium Risk'**, **'I-Medium Risk'**), (3, 3))  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'E-Low Risk'**, **'F-Low Risk'**, **'G-Low Risk'**), (4, 4, 4))  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**] = test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].replace((**'A-Very Low Risk'**, **'B-Very Low Risk'**,  
 **'C-Very Low Risk'**, **'D-Very Low Risk'**), (5, 5, 5, 5))  
  
test\_X[**'PERFORM\_CNS.SCORE.DESCRIPTION'**].value\_counts()  
  
**import** re  
**def** toMonths(str):  
 cache = []  
 **for** k **in** X[str]:  
 temp = int(re.split(**"[yrs mon]+"**, k)[0]) \* 12 + int(re.split(**"[yrs mon]+"**, k)[1])  
 cache.append(temp)  
 **return** cache  
  
**def** toMonthstest(str):  
 cache = []  
 **for** k **in** test\_X[str]:  
 temp = int(re.split(**"[yrs mon]+"**, k)[0]) \* 12 + int(re.split(**"[yrs mon]+"**, k)[1])  
 cache.append(temp)  
 **return** cache  
  
X[**'CREDIT.HISTORY.LENGTH'**] = toMonths(**'CREDIT.HISTORY.LENGTH'**)  
X[**'CREDIT.HISTORY.LENGTH'**][:5]  
  
X[**'AVERAGE.ACCT.AGE'**] = toMonths(**'AVERAGE.ACCT.AGE'**)  
  
X[**'AVERAGE.ACCT.AGE'**][:5]  
  
test\_X[**'CREDIT.HISTORY.LENGTH'**] = toMonthstest(**'CREDIT.HISTORY.LENGTH'**)  
test\_X[**'AVERAGE.ACCT.AGE'**] = toMonthstest(**'AVERAGE.ACCT.AGE'**)  
test\_X[**'AVERAGE.ACCT.AGE'**][0:5]  
  
  
  
  
  
  
**from** imblearn.over\_sampling **import** SMOTE  
  
oversample = SMOTE()  
x\_train, y\_train = oversample.fit\_resample(X, Y.values.ravel())  
  
print(x\_train.shape)  
print(y\_train.shape)  
  
  
*# pca = PCA(n\_components=7).fit(X)  
# X = pca.fit\_transform(X)  
# X = pd.DataFrame(X, columns = ['p1','p2','p3','p4','p5','p6','p7'])  
# test\_df = pd.DataFrame(pca.fit\_transform(train.iloc[:, -1]), columns = ['p1','p2','p3','p4','p5','p6','p7'])  
# #Plotting the Cumulative Summation of the Explained Variance  
# plt.figure(figsize=(15,5))  
# plt.plot(np.cumsum(pca.explained\_variance\_ratio\_))  
# plt.xlabel('Number of Components')  
# plt.ylabel('Variance (%)') #for each component  
# plt.title('Pulsar Dataset Explained Variance')  
# plt.show()  
  
# import numpy as np***from** sklearn.model\_selection **import** train\_test\_split  
  
*#splitting training data into train and validation set*X\_train, X\_valid, Y\_train, Y\_valid = train\_test\_split(x\_train, y\_train, test\_size = 0.2, random\_state = 0)  
  
print(X\_train.shape)  
print(Y\_train.shape)  
  
print(X\_valid.shape)  
print(Y\_valid.shape)  
*#***from** sklearn.preprocessing **import** StandardScaler  
  
scalar = StandardScaler()  
X\_train = scalar.fit\_transform(X\_train)  
X\_valid = scalar.transform(X\_valid)  
test\_X = scalar.transform(test\_X)  
  
  
**from** sklearn.metrics **import** roc\_auc\_score  
  
modelXG = DecisionTreeClassifier(max\_depth=3,random\_state=100,criterion=**'entropy'**,min\_samples\_leaf=5)  
modelXG.fit(X\_train, Y\_train)  
  
Y\_predXG = modelXG.predict(X\_valid)  
  
print(**"Train Accuracy: "**, modelXG.score(X\_train, Y\_train))  
print(**"Validation Accuracy: "**, modelXG.score(X\_valid, Y\_valid))  
  
print(**"AUROC Score of decision = "**, roc\_auc\_score(Y\_valid, Y\_predXG))  
  
**from** sklearn.ensemble **import** RandomForestClassifier  
  
modelRF = RandomForestClassifier(max\_depth=3,random\_state=500)  
modelRF.fit(X\_train, Y\_train)  
  
Y\_predRF = modelRF.predict(X\_valid)  
  
print(**"Train Accuracy: "**, modelRF.score(X\_train, Y\_train))  
print(**"Validation Accuracy: "**, modelRF.score(X\_valid, Y\_valid))  
  
print(**"AUROC Score of Random Forest = "**, roc\_auc\_score(Y\_valid, Y\_predRF))  
  
  
  
modelAB = LogisticRegression()  
modelAB.fit(X\_train, Y\_train)  
  
Y\_predAB = modelAB.predict(X\_valid)  
  
print(**"Train Accuracy: "**, modelAB.score(X\_train, Y\_train))  
print(**"Validation Accuracy: "**, modelAB.score(X\_valid, Y\_valid))  
  
print(**"AUROC Score of logistic = "**, roc\_auc\_score(Y\_valid, Y\_predAB))  
  
**from** sklearn.ensemble **import** GradientBoostingClassifier  
  
modelGB = GradientBoostingClassifier()  
modelGB.fit(X\_train, Y\_train)  
  
Y\_predGB = modelGB.predict(X\_valid)  
  
print(**"Training Accuracy: "**, modelGB.score(X\_train, Y\_train))  
print(**'Testing Accuarcy: '**, modelGB.score(X\_valid, Y\_valid))  
  
print(**"AUROC Score of Gradient Boosting = "**, roc\_auc\_score(Y\_valid, Y\_predGB))  
  
  
test\_Y\_RF = modelRF.predict(test\_X)  
test\_Y\_XG = modelXG.predict(test\_X)  
test\_Y\_AB = modelAB.predict(test\_X)  
test\_Y\_GB = modelGB.predict(test\_X)  
test\_Y\_pred = []  
*#***for** i **in** range(len(test\_Y\_RF)):  
 k = 0.25 \* test\_Y\_RF[i] + 0.175 \* test\_Y\_GB[i] + 0.125 \* test\_Y\_XG[i] + 0.1 \* test\_Y\_AB[i] *# weighted averaging* test\_Y\_pred.append(k)