CS3481

Project 2 Report

Group 15

Names:

DE Ankolika : 55350200

NAUTIYAL Rishabh: 55331640

RUDRARAJU Charan Kumar Raju: 55308163

VOBBILISETTY Sai Navyanth: 55357700

Abstract

Our study is based on a Kaggle Dataset containing certain information on payments, demographic factors, history of payment, credit data, and bill statements in Taiwan from April 2005 to September 2005. Through our research, we want to make better predictions for credit card defaulters. We believe that this is an important aspect of the financial industry because it acts as a strategic opportunity in terms of business for banks and other financial institutions. Moreover, the Wall Street Journal says that credit card debt is at an all-time high of $930 Billion. We have attempted to build models to analyse the important attributes, and make predictions which are as close as possible to the actual data. We used K-means, Decision Tree, Random Forest and XGboost to construct and compare training models and then evaluated the results that were obtained.

Introduction

Credit card default

Defaulting on a credit card is defined as failure to make a payment for your credit card for 180 days, unless specified differently by the company. If this happens, the card issuer is legally allowed to take action which includes selling off the account, closing your card, etc. This inability to pay credit card bills can actually be due to a variety of reasons, but the causative reasons for each individual case can often be traced back to some common attributes.

Why is it important to make a prediction for defaulters?

Credit cards are one of the most popular forms of transactional options, as they are very attractive based on the convenience they offer. This comes from the fact that they offer ancillary benefits and don't require reserves of money.

In a well-developed financial system, risk prediction is essential for predicting business performance or individual customers' credit risk and to reduce the damage and uncertainty in banks and other financial institutions(Meriikoski, Vittala, 2018). To manage and analyse this risk, it is important to handle large amounts of data that represent a versatile array of

attributes. For the bank, it is important to analyse and identify the potential defaulters because the consequence of credit card defaulters coming into play, can result in huge losses. Moreover, the defaulters who may be able to pay but are constantly exceeding the credit limit can also impact the bank such that it results in very large losses. If and when such situations are noted, the predictions can help in the prevention of enormous financial losses. The point to note here is that banks would always want to maximize the number of credit card users, but the defaulters offer drawbacks that supersede the gains. Hence, if there was a simple and precise way to manage and analyse this data, by using a variety of factors such as age, sex, transaction history, etc, the banks could gradually increase the number of credit card holders, and optimize their profit levels based on that.

How have we attempted to make the predictions?

We have used a variety of models to classify the data for the purpose of predicting the class values. In this study, the class value 0, represents the prediction of the user not being a defaulter, while the class value 1 represents the prediction of the user having more chances of being a defaulter.

The following are the attributes of the dataset

ID: ID of each client

LIMIT_BAL: Amount of given credit in NT dollars (includes individual and family/supplementary credit

SEX: Gender (1=male, 2=female)

EDUCATION: (1=graduate school, 2=university, 3=high school, 4=others, 5=unknown, 6=unknown)

MARRIAGE: Marital status (1=married, 2=single, 3=others)

AGE: Age in years

PAY_0: Repayment status in September, 2005 (-1=pay duly, 1=payment delay for one month, 2=payment delay for two months, ... 8=payment delay for eight months, 9=payment delay for nine months and above)

PAY_2: Repayment status in August, 2005 (scale same as above)

PAY_3: Repayment status in July, 2005 (scale same as above)

PAY_4: Repayment status in June, 2005 (scale same as above)

PAY_5: Repayment status in May, 2005 (scale same as above)

PAY_6: Repayment status in April, 2005 (scale same as above)

BILL_AMT1: Amount of bill statement in September, 2005 (NT dollar)

BILL_AMT2: Amount of bill statement in August, 2005 (NT dollar)

BILL_AMT3: Amount of bill statement in July, 2005 (NT dollar)

BILL_AMT4: Amount of bill statement in June, 2005 (NT dollar)

BILL_AMT5: Amount of bill statement in May, 2005 (NT dollar)

BILL_AMT6: Amount of bill statement in April, 2005 (NT dollar)

PAY_AMT1: Amount of previous payment in September, 2005 (NT dollar)

PAY_AMT2: Amount of previous payment in August, 2005 (NT dollar)

PAY_AMT3: Amount of previous payment in July, 2005 (NT dollar)

PAY_AMT4: Amount of previous payment in June, 2005 (NT dollar)

PAY_AMT5: Amount of previous payment in May, 2005 (NT dollar)

PAY_AMT6: Amount of previous payment in April, 2005 (NT dollar)

Our study follows the following approach: Preprocessing, training and evaluation. Each step has been explained in detail in the next sections. Many pieces of literature have criticized the use of a single method for the purpose of classifying a dataset such as this and hence, we decided to use ensemble methods to better use the strength of multiple classifiers. Finally, after getting the different results, we compared them and devised the most optimum way to make the predictions.

Preprocessing

Step 1: Exploring the data.

- Using the df.info command in python we found out that there are 25 attributes (including default.payment.next.month) in the dataset out of which 12 are int type while 13 are float type.
- Using the command df.isnull().sum() command we figured out that there are no null values in the data set and that it is complete.
- In the dataset, the chances of a person defaulting the payment next month is 0.2212 (default.payment.next.month =1) and chances of not defaulting the payment next month is 0.7788 (default.payment.next.month =1)

Step 2: Dropping Attributes.

From the data set we dropped the attributes ID and default.payment.next.month. We dropped the ID because ID is unique to each person and from heuristics and common sense doesn't tell if a person will default a payment or not. We dropped the default.payment.next.month because it is the class value we are trying to predict.

Step 3: Splitting the data set.

We split the dataset into train and test sets.

Train Set: 70%, Test Set: 30%

Step 4: Feature Scaling.

The standard deviation of the non categorical features is high as shown in the image, thus we need to scale them to the level of other features.

- Categorical Features: `['EDUCATION','SEX','MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6']`

	0	1	2	3	4	5	6	7	8	9	10
count	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000
mean	-0.004206	0.006028	0.008777	0.008189	-0.003292	-0.001855	0.007425	0.006171	0.008565	-0.022296	-0.011118
std	0.985820	0.995975	0.986415	0.986157	0.962622	0.973559	0.979241	0.986081	1.006068	0.647997	0.860730
min	-1.209901	-1.567756	-2.929712	-1.630499	-2.914023	-1.448360	-1.523207	-4.143852	-0.339993	-0.240261	-0.288366
25%	-0.902914	-0.809277	-0.639975	-0.642506	-0.633744	-0.630985	-0.626925	-0.626640	-0.282880	-0.209019	-0.267094
50%	-0.212195	-0.159152	-0.378199	-0.389066	-0.387163	-0.375920	-0.364473	-0.366976	-0.212974	-0.160973	-0.189723
75%	0.555271	0.707682	0.232682	0.230241	0.186464	0.169522	0.170533	0.182375	-0.037371	-0.043026	-0.036672
max	4.469349	4.716785	9.410755	9.724082	8.341633	7.440354	7.606017	7.285634	30.204989	22.657320	22.488660

	SEX	PAY_0	AGE	LIMIT_BAL	BILL_AMT1	PAY_AMT1
count	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000	9000.000000
mean	1.603444	-0.004556	35.524444	167100.666667	51677.578667	5762.700000
std	0.489209	1.126750	9.191851	128451.270711	72930.704129	16633.319818
min	1.000000	-2.000000	21.000000	10000.000000	-165580.000000	0.000000
25%	1.000000	-1.000000	28.000000	50000.000000	3712.000000	944.250000
50%	2.000000	0.000000	34.000000	140000.000000	23066.500000	2100.000000
75%	2.000000	0.000000	42.000000	240000.000000	68232.000000	5003.250000
max	2.000000	8.000000	79.000000	750000.000000	746814.000000	505000.000000

  - The above image shows similar standard deviations, among attributes after scaling.
- Scale the remaining features which are numerical using StandardScaler()

```
x_train_s = scaler.fit_transform( x_train.drop(cate_features, axis=1))
x_test_s = scaler.transform( x_test.drop(cate_features, axis=1))
```

- Finally we combine the scaled features with the categorical and ordinal features using np.hstack(). As seen above, after scaling, the standard deviation, for all the attributes become similar.

Training:

We are using different types of models to figure out the best option and will then compare the results of each type. We have used the following 4 methods:

- **K Means:** this algorithm clusters data by trying to separate samples in n groups(2 in our case) of equal variance, minimizing a criterion known as the inertia or within-cluster sum-of-squares. It scales well to a large number of samples and has been used across a large range of application areas in many different fields.
- **XGBoost:** this is an implementation of a gradient boosted decision tree focused on more speed and better performance. Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
- **Decision Tree:** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.
- **Random Forest:** Random forests is an ensemble learning method that operates by creating multiple decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forests corrects for decision trees' habit of overfitting to their training set.

Training K Means Model:

For the K Means Model we preprocessed the data separately.
We did this by importing the MinMaxScaler from the sklearn.preprocessing.

```
from sklearn.preprocessing import MinMaxScaler
```

Next we fit the dataset to the MinMaxScaler using the fit_transform. We also dropped the ID and default.payment.next.month attributes from the dataset for the same reasons as above:

```
import numpy as np
minmax_norm = MinMaxScaler()
X_std = minmax_norm.fit_transform(df.drop(
['ID','default.payment.next.month'], axis=1))
np.min(X_std,axis=0), np.max(X_std,axis=0)
```

We also have our own implementation of the KMeans algorithm in the defined class.
We use the following functions defined by us in Class Kmeans:

- initalize_centroids(self,X): initializes the values.
- compute_centroids(self,X,labels): calculates the centroids of the clusters
- compute_distance(self,X,centroids): calculates the distance of each point from the centroids of the clusters
- find_closest_cluster(self,distance): finds the centroid which is closest to a point
- compute_sse(self,X,labels,centroids): computes the sum of squared errors
- fit(self,X): trains the model using the above helper functions
- predict(self,X): predicts which the cluster to which a point belongs to.

```
from sklearn.datasets.samples_generator import (make_blobs,make_circles,
make_moons)

from sklearn.cluster import KMeans, SpectralClustering|
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import silhouette_samples, silhouette_score

kmeans_clr = Kmeans(n_clusters=2)#only two clusters for 0 and 1
kmeans_clr.fit(X_std)
```

Training XGBoost Model:

- For using XGBoost Model in python we first had to install the xgboost component using the following command:

```
!pip install xgboost
```

And imported it using the code

```
from xgboost import XGBClassifier
```

❖ We defined the models and the hyperparameters and then fitted the data to mode:

```
model = XGBClassifier(objective = 'binary:logistic', max_depth = 4)
model.fit(x_train_s, y_train)
```

❖ Finally we made the predictions using the functions predict and predict_proba on both x_train_s and x_test_s data sets (scaled sets) for comparison and calculation of features like pr curve etc.

```
y_train_pred = model.predict(x_train_s)
y_train_pred_prob = model.predict_proba(x_train_s)
y_test_pred =  model.predict(x_test_s)
y_test_pred_prob = model.predict_proba(x_test_s)
```

❖ We limited the max_depth of the tree to 4 to prevent overfitting.

Training Decision Tree Model:

Training the decision tree model was a simple task as we used the DecisionTreeClassifier from the sklearn library. We used the .fit(x_train_s, y_train) to fit the training data to the model and then predicted the values of the x_test_s set using the predict function.

```
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier().fit(x_train_s,y_train)
y_pred_dt=dt.predict(x_test_s)
print(accuracy_score(y_test,y_pred_dt))
print(confusion_matrix(y_test,y_pred_dt))
```

Training Random Forest Model:

This model was again imported from the sklearn library in python.

❖ We defined the parameters for the function as following:

```
RFC_METRIC = 'gini'  #metric used for RandomForrestClassifier
NUM_ESTIMATORS = 100 #number of estimators used for
RandomForrestClassifier
NO_JOBS = 4 #number of parallel jobs used for RandomForrestClassifier
RANDOM_STATE = 2018
```

❖ And fitted the x_train_s dataset to the model using the following command:

```
rf=RandomForestClassifier(n_jobs=NO_JOBS,
random_state=RANDOM_STATE,
criterion=RFC_METRIC,n_estimators=NUM_ESTIMATORS,
verbose=False).fit(x_train_s,y_train)
```

❖ Finally we predicted the results using the rf.predict(x_test_s) function on the scaled test data set.

```
y_pred_rf=rf.predict(x_test_s)
```

Training Random Forest Model with Cross Validation:

This was done by importing the cross_val_score from model_selection in sklearn library in python:

```
from sklearn.model_selection import cross_val_score
```

We defined some parameters for Cross Validation:

```
#CROSS-VALIDATION
      NUMBER_KFOLDS = 5 #number of KFolds for cross-validation
      RANDOM_STATE = 2020
      MAX_ROUNDS = 1000 #lgb iteration
      EARLY_STOP = 50 #lgb early stop
      OPT_ROUNDS = 1000  #To be adjusted based on best validation rounds
      VERBOSE_EVAL = 50 #Print out metric result
```

We used this to get the score of the cross validation as follows:

```
rfc_cv_score = cross_val_score(rf,
      df.drop(['ID','default.payment.next.month'],axis=1),
                  df['default.payment.next.month'],   cv=10,
scoring='precision')
```

Training XGBoost Model with Cross Validation:

To do this we required to preprocess the data separately because we had to convert the df to DMatrix:

```
cate_features = ['EDUCATION','SEX','MARRIAGE', 'PAY_0', 'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6'] #categorical and ordinal features

### START CODE HERE ### (3~4 lines)

scaler = StandardScaler() #initiate the StandardScaler

df_copy = df.drop(['ID','default.payment.next.month'], axis=1)

dfcopy = scaler.fit_transform( df_copy.drop(cate_features,axis=1) ) #fit the scaler with train set numerical data and then transform it
```

Here we first scaled the numerical values of the dataset and then adding the categorical features to the scaled dataset using np.hstack:

```
df_xgb = np.hstack( [dfcopy, df_copy[cate_features].values] )
```

For training we import cross_val_score from sklearn and also classification report and confusion metrics from sklearn.metrics

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

To run the model (the previous xgb model) with cross validation and get the cross validation score we do used the following command:

```
xgb_cv_score = cross_val_score(model, df_xgb,
df['default.payment.next.month'], cv=10, scoring='precision')
```

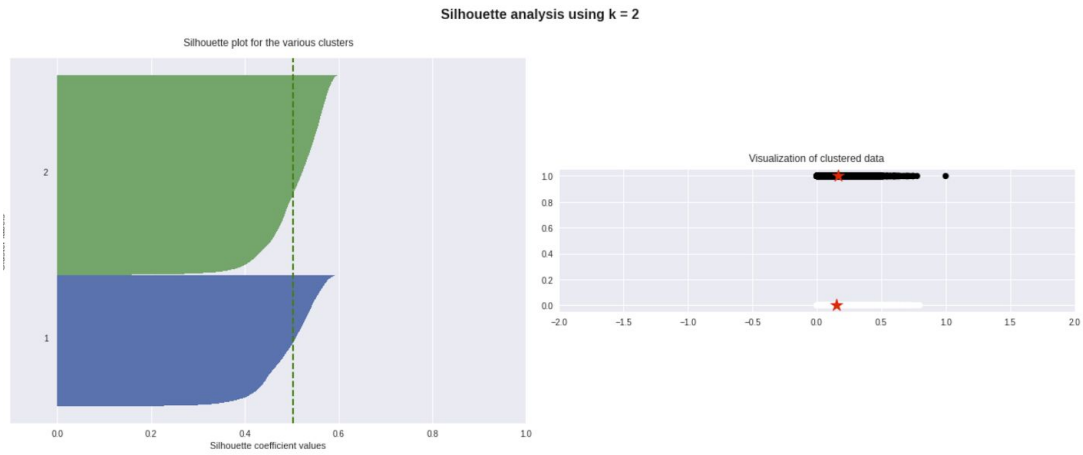
This gives us xgb_cv_score which is an array of the Cross Validation score of ten rounds:

```
array([0.62068966, 0.63817664, 0.65263158, 0.63610315,
0.67329545,0.66974596, 0.73246753, 0.75241158, 0.71471471, 0.7191358])
```

To prevent overfitting we defined the max_depth to 4 and also used early stopping defined to 50 as can be seen in the defined variables above.

Evaluation:

KMeans

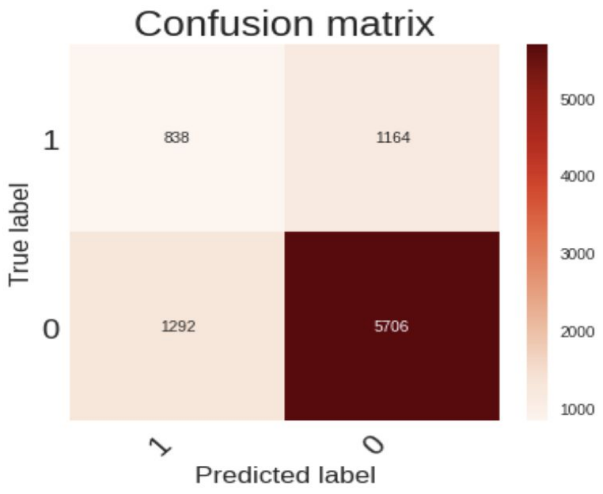


For K-means, we have decided to calculate the Silhouette coefficient as it is a good measure to analyse the number of clusters that are to be formed. It is calculated in the following way, for each corresponding instance:

(x-y)/ Max(x,y)

As we can see, the Silhouette Coefficient we get is 0.5. So, we know that there must be only 2 clusters, which are basically, class 0 and class 1, so a better clustering solution should result in a very high silhouette coefficient for k=2, but k-means provides only a value of 0.5 which is not high, that is why we didn't even evaluate this model with other extrinsic methods of evaluation.

The evaluation process that follows mainly consists of studying the outputs of the ROC curves and the confusion matrices for the classification models.



Decision Tree Classifier:

For the decision tree classifier, the following were the outputs of the confusion matrix and Roc curve.

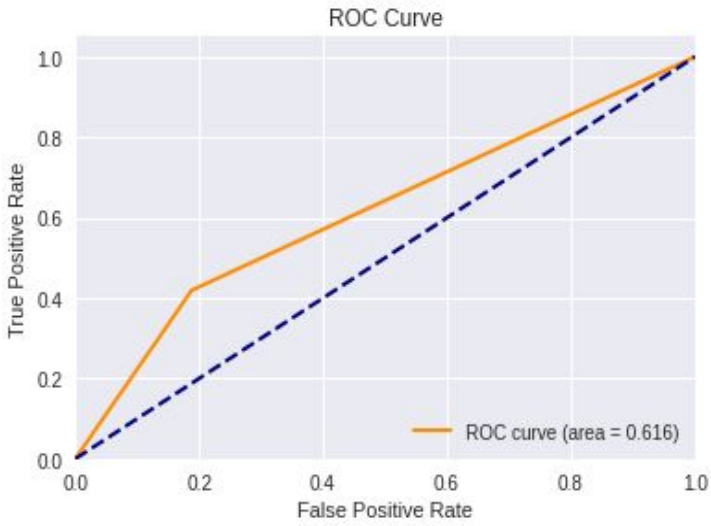
```
Accuracy: 0.731
Precision: 0.403
Recall: 0.430
F1 Score: 0.416
```

As can be seen, from the above values, the accuracy was 0.731 and the precision was 0.403. This concludes to the fact that though the value of accuracy is high, precision which is in fact a better evaluator has a low value. Therefore, accuracy is not a good evaluation metric.

Unfortunately, precision and recall are often in tension, i.e, improving precision typically reduces recall and vice versa. So, here we can see that precision is 0.403 which is very low and the recall is 0.430 which is comparatively high. So, instead of facing difficulty in evaluating with precision and recall, we use another evaluation metric called F1 score.

F1 score actually conveys the balance between precision and recall, which is basically the harmonic mean of precision and recall. So, we can easily compare models using the f1 score. If the f1 score is high, it indicates that the model is good.bIn our case we have an f1 score of 0.406, which we shall attempt to compare with the other classification models.

Next, we can evaluate the ROC curve. The ROC curve represents a plot between the True Positive rate on the y-axis and the False Positive rate on the x-axis. The area under this curve, represents the measure of how well a parameter can differentiate among two groups.



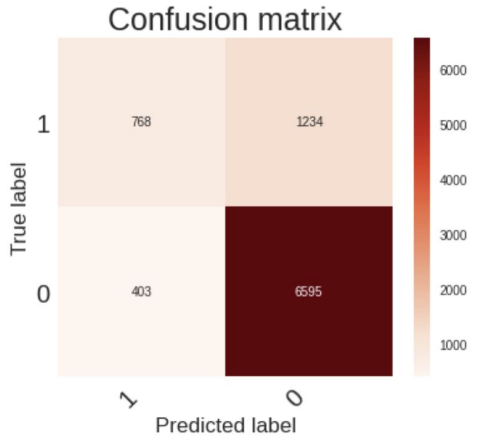
First, we attempt to analyse the ROC curve of the Decision tree classifier.

The area under the ROC curve is 0.616. In the following paragraphs, we will attempt to compare this value to the values generated by other classifiers.

Random Forest:

Now we output the confusion matrix and ROC curve of the Random Forest classifier.

```
Accuracy: 0.818
Precision: 0.656
Recall: 0.384
F1 Score: 0.484
```

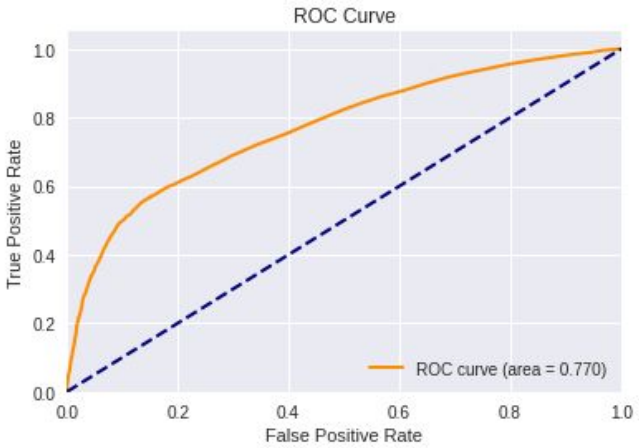


In this case, the value of accuracy is 0.818 and the value of precision is 0.656. As we can see, both the values of accuracy and the precision are high.

As we have mentioned before, precision and recall are often in tension. That is, improving precision typically reduces recall and vice versa. So, here we can see that precision is 0.656 which is higher than that of decision tree classifier and the recall is 0.384 which is lower when compared to decision tree. So, instead of facing difficulty in evaluating with precision and recall, we again turn to f1 score.

Here, the f1 score is 0.484 which is better than that of the decision tree classifier (0.416). So, we can say that random forest is a better classifier when compared to the decision tree classifier.

Now, let's have a look at the ROC curve for this classifier.



As we can see, the area under the ROC curve is 0.770. This value is higher than that of the decision tree classifier (0.616). By this, we can say that the Random Forest classifier is better than the decision tree classifier as it evidently points towards better(higher values) in terms of our evaluation metrics.

Random Forest with Cross Validation:

We first list out the mean AUCs, mean precision, mean recall and mean f1 score for this classification of Random Forest with Cross Validation.

Avg Precision - 0.64959970
Avg Recall - 0.37146050
Avg f1 score - 0.47377706
Avg AUC_ROC - 0.7651234633314586

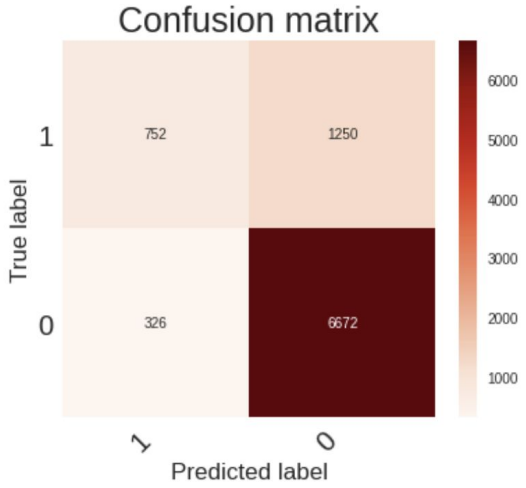
Comparing to other Classifiers before:

- This precision is less than the Random Forest classifier without cross validation (0.656), but it is higher than the decision tree classifier (0.403).
- This recall is still less than the Random Forest Classifier without cross validation (0.384), and also less than the decision tree classifier (0.430).
- This f1 score is more than the decision tree classifier (0.416) but slightly less than the Random Forest Classifier without cross-validation (0.484).
- This AUC_ROC is more than the decision tree classifier (0.616) but less than Random Forest Classifier without cross validation (0.770)

XGBoost:

Let's output the confusion matrix for this classifier first:

```
Accuracy: 0.825
Precision: 0.698
Recall: 0.376
F1 Score: 0.488
```

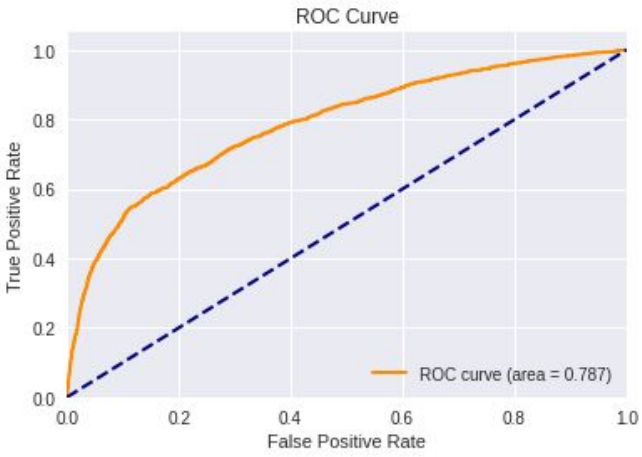


In this case, the precision is 0.698. Comparing it with all other classifiers we mentioned above, this value is the highest precision value that we have been able to achieve.

Here, the recall is 0.376. As the precision is the highest, the recall here should be the lowest. But, strangely it is slightly higher than the recall of the Random Forest with cross validation classifier.

Here, the F1 score is 0.488. The F1 score is the highest corresponding score achieved so far.

Let's now output the ROC curve.



As we can see, the area under the ROC Curve is 0.787 which is the highest that we have obtained till now.

XGBoost with Cross Validation

We first list out the mean AUCs, mean precision, mean recall and mean f1 score for this classification of Random XGBoost with Cross Validation.

Avg Precision - 0.6809372
Avg Recall - 0.3652844
Avg f1 score -0.4745614
Avg AUC_ROC -0.78189844

Comparing to other Classifiers before:

- This precision is higher than the Random Forest classifier without cross validation (0.656), the decision tree classifier (0.393) and Random Forest classifier with cross validation (0.64959970).
- This recall is still less than the Random Forest Classifier without cross validation (0.384), and also less than the decision tree classifier (0.419).
- This f1 score is more than the decision tree classifier (0.406) and Random Forest with cross validation (0.47377706) but slightly less than the Random Forest Classifier without cross-validation (0.484).

❖ This AUC_ROC is more than the decision tree classifier (0.616) and also the Random Forest Classifier without cross validation (0.770) and Random Forest Classifier with Cross Validation (0.76512346)

Conclusion:

Classification Model	*F1 Score*	*AUC value*
Decision tree	0.406	0.616
Random forest **without** cross validation	0.484	0.770
Random Forest **with** cross validation	0.47377706 (Mean)	0.7651235 (Mean)
XGboost **without** cross validation	0.488	0.787
XGboost **with** cross validation	0.4745614 (Mean)	0.78189844 (Mean)

The above table summarises the values that have been generated, for the ease of comparison. Here, we can see that XGboost without cross validation gives the best results and hence can be used as a mode of prediction for credit card defaulters. The average f1 score for that particular model is the highest and the AUC value is also the highest that is achieved so far. This allows us to form the conclusion of our report as XGboost without cross validation being the best way to predict credit card defaulters.

References:

1. Noeske, F. (2007). The 2- and 3-modular characters of the sporadic simple Fischer group Fi22 and its cover. Journal of Algebra, 309(2), 723–743. doi: 10.1016/j.jalgebra.2006.06.020
2. Sharma, Sunakshi & Mehra, Vipul. (2018). Default Payment Analysis of Credit Card Clients.10.13140/RG.2.2.31307.28967.
3. Figure 2f from: Irimia R, Gottschling M (2016) Taxonomic revision of Rochefort Sw. (Ehretiaceae,Boraginales).Biodiversity Data Journal 4: e7720. https://doi.org/10.3897/BDJ.4.e7720. (n.d.). doi: 10.3897/bdj.4.e7720.figure2f
4. Figure 2f from: Irimia R, Gottschling M (2016) Taxonomic revision of Rochefort Sw. (Ehretiaceae, Boraginales). Biodiversity Data Journal 4: e7720. https://doi.org/10.3897/BDJ.4.e7720. (n.d.). doi: 10.3897/bdj.4.e7720.figure2f
5. https://www.kaggle.com/uciml/default-of-credit-card-clients-dataset
6. L.C. Thomas, A survey of credit and behavioral scoring: Forecasting financial risk of lending to consumers International Journal of Forecasting, 16 (2) (2000), pp. 163-167
7. sklearn.tree.DecisionTreeClassifier — scikit-learn 0.22.2 documentation. (2020). Scikit-learn.org. Retrieved 1 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
8. sklearn.ensemble.RandomForestClassifier — scikit-learn 0.22.2 documentation. (2020). Scikit-learn.org. Retrieved 1 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.ens
9. XGBoost Documentation — xgboost 1.1.0-SNAPSHOT documentation. (2020). Xgboost.readthedocs.io. Retrieved 1 May 2020, from https://xgboost.readthedocs.io/en/latest/

10. sklearn.model_selection.cross_val_score — scikit-learn 0.22.2 documentation. (2020). Scikit-learn.org. Retrieved 1 May 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.

11. API Reference — scikit-learn 0.22.2 documentation. (2020). Scikit-learn.org. Retrieved 1 May 2020, from https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics