MACHINE LEARNING ENGINEER NANODEGREE

CAPSTONE PROJECT REPORT

Prediction Of Consumer Credit Risk

By :- KARTIK GUPTA

- # DEFINITION
  - ### Project Overview

Because of the mortgage crisis that U.S economy faced in 2007 which caused panic and turmoil around the world, many banks and credit companies have developed new tools to assess credit risk of individuals more carefully and accurately. People also realized that the main cause of the crisis was that loans were given to people whose credit risk profile was high.

Nowadays due to increasing number of companies created in the field of credit and peer to peer lending, it has become difficult for an individual to get a loan may be because of their banking history. This is why in this project, I tried to build a model for lending managers or banks so that they can easily and accurately identify the default risk of their clients. The main goal of this project is to predict if a consumer will experience a delinquency (90 days or worse) during the next two years.

The dataset for this project is picked from a Kaggle competition **"Give Me Some Credit"** which consists of historical data of 1,50,000 borrowers characterized by 11 variables.

Dataset link :- https://www.kaggle.com/c/GiveMeSomeCredit/data

○ Problem Statement

Develop a supervised learning model using classification algorithms to predict whether a consumer will experience a serious delinquency (90 days or worse) during the next two years. It is a binary classification problem. The output will be either 0 or 1, 0 indicates that the individual **will not** experience a serious delinquency (90 days or worse) during the next two years and 1 indicates that the individual **will** experience a serious delinquency (90 days or worse) during the next two years.

○ Metrics

The metric chosen to measure the performance of the models is area under the Receiver Operating Characteristic (ROC) curve (AUC).

The receiver operating characteristic curve (ROC) is a two-dimensional graphical illustration of the trade-off between the true positive rate (sensitivity) and false positive rate (specificity). The ROC curve illustrates the behaviour of a classifier without having to take into consideration the class distribution or misclassification cost. In order to compare the ROC curves of different classifiers, the area under the receiver operating characteristic curve (AUC) must be computed.

Sensitivity (tpr)= true positives/(true positive + false negative)

Specificity (fpr)=true negatives/(true negative + false positives)

For this project, I preferred AUC over accuracy because :-

1. The accuracy depends on the threshold chosen, whereas the AUC considers all possible thresholds. AUC provides a broader view of the performance of the classifier.
2. Accuracy is biased on the size of training and testing data while AUC is not.

- # ANALYSIS
  - ## Data Exploration

Descriptive statistics of the data:

	SeriousDlqin2yrs (Target)	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome
count	150000.00000	150000.00000	150000.00000	150000.0000	150000.00000	1.202690e+05
mean	0.066840	6.048438	52.295207	0.421033	353.005076	6.670221e+03
std	0.249746	249.755371	14.771866	4.192781	2037.818523	1.438467e+04
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00
25%	0.000000	0.029867	41.000000	0.000000	0.175074	3.400000e+03
50%	0.000000	0.154181	52.000000	0.000000	0.366508	5.400000e+03
75%	0.000000	0.559046	63.000000	0.000000	0.868254	8.249000e+03
max	1.000000	50708.000000	109.000000	98.000000	329664.00000	3.008750e+06

	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents
count	150000.00000	150000.00000	150000.00000	150000.0000	146076.000000
mean	8.452760	0.265973	1.018240	0.240387	0.757222
std	5.145951	4.169304	1.129771	4.155179	1.115086
min	0.000000	0.000000	0.000000	0.000000	0.000000
25%	5.000000	0.000000	0.000000	0.000000	0.000000
50%	8.000000	0.000000	1.000000	0.000000	0.000000
75%	11.000000	0.000000	2.000000	0.000000	1.000000
max	58.000000	98.000000	54.000000	98.000000	20.000000

Null Values :

ATTRIBUTE	NULL VALUES
SeriousDlqin2yrs (Target)	0
RevolvingUtilizationOfUnsecuredLines	0
age	0
NumberOfTime30-59DaysPastDueNotWorse	0
DebtRatio	0
MonthlyIncome	29731
NumberOfOpenCreditLinesAndLoans	0
NumberOfTimes90DaysLate	0
NumberRealEstateLoansOrLines	0
NumberOfTime60-89DaysPastDueNotWorse	0
NumberOfDependents	3924

Outliers :

ATTRIBUTE	Outliers detected by Tukey's method	Outliers actually present
RevolvingUtilizationOfUnsecuredLines	528	3321 (values > 1.0)
age	2	1
NumberOfTime30-59DaysPastDueNot Worse	23982	269
DebtRatio	30815	30815
MonthlyIncome	0	0
NumberOfOpenCreditLinesAndLoans	1898	17 (values > 49)
NumberOfTimes90DaysLate	8338	269
NumberRealEstateLoansOrLines	473	24 (values > 15)

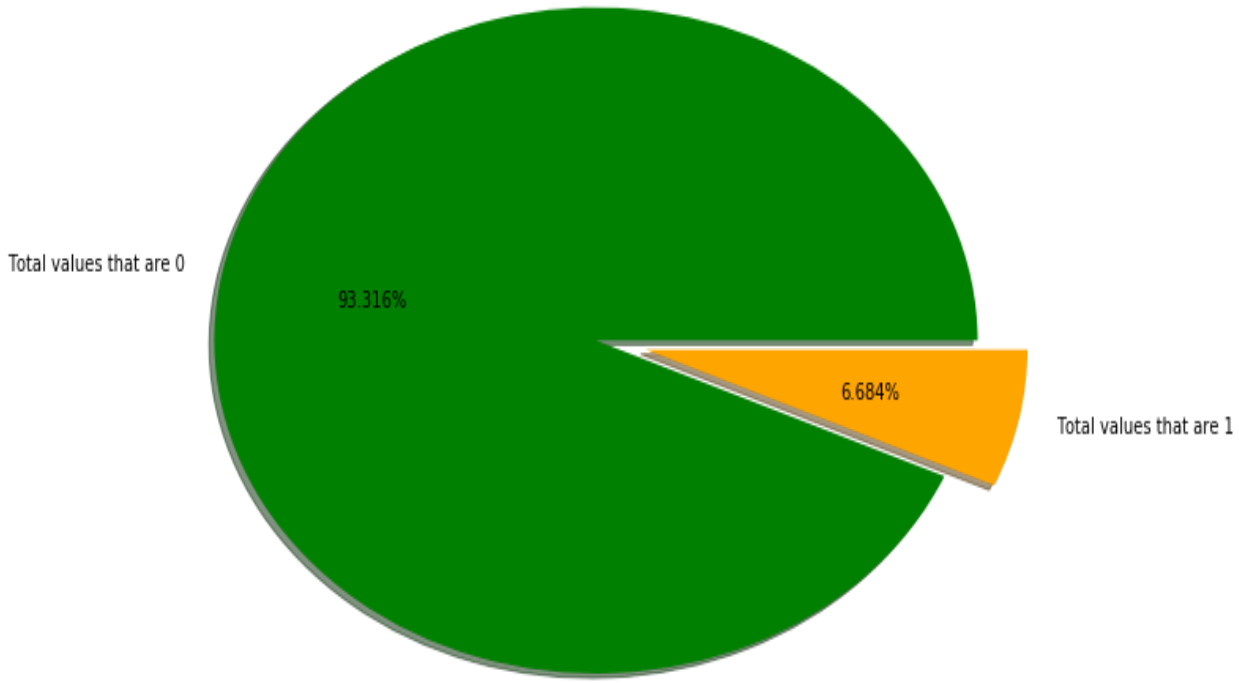
NumberOfTime60-89DaysPastDueNot Worse	7604	269
NumberOfDependents	0	2

Observations :-

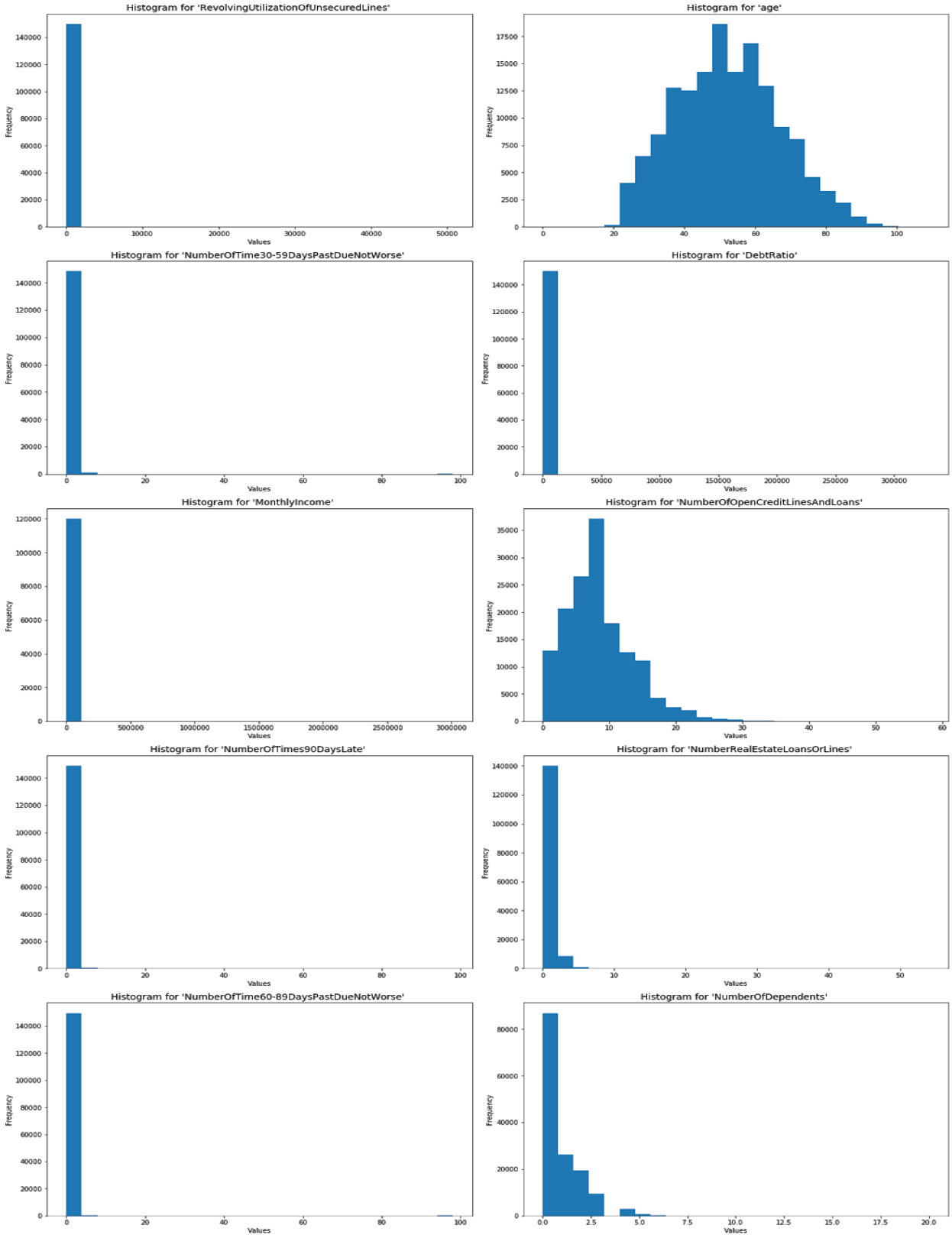
1. According to the statistical records obtained from first two tables, it is clear that some of the features have mean >>> median which indicates that most of the features are skewed to the left while last 3-4 features have mean closer to median but not equal which means that there is less skewness in those features.
2. Third table shows that two of the features namely **'MonthlyIncome'** and **'NumberOfDependents'** have null values while others do not.
3. According to fourth table, some features like **'RevolvingUtilizationOfUnsecuredLine', 'NumberOfTime30-59DaysPastDueNotWorse', 'DebtRatio'** and **'NumberOfTime60-89DaysPastDueNotWorse'** contains many outliers while the remaining features have very less outliers (approx 10-20). Presence of these outliers is the clear reason for the presence of skewness in the data.
4. Therefore, I conclude that almost all features **(except age)** are neither normalized nor balanced because of the presence of outliers and null points.

- **Exploratory Visualization**

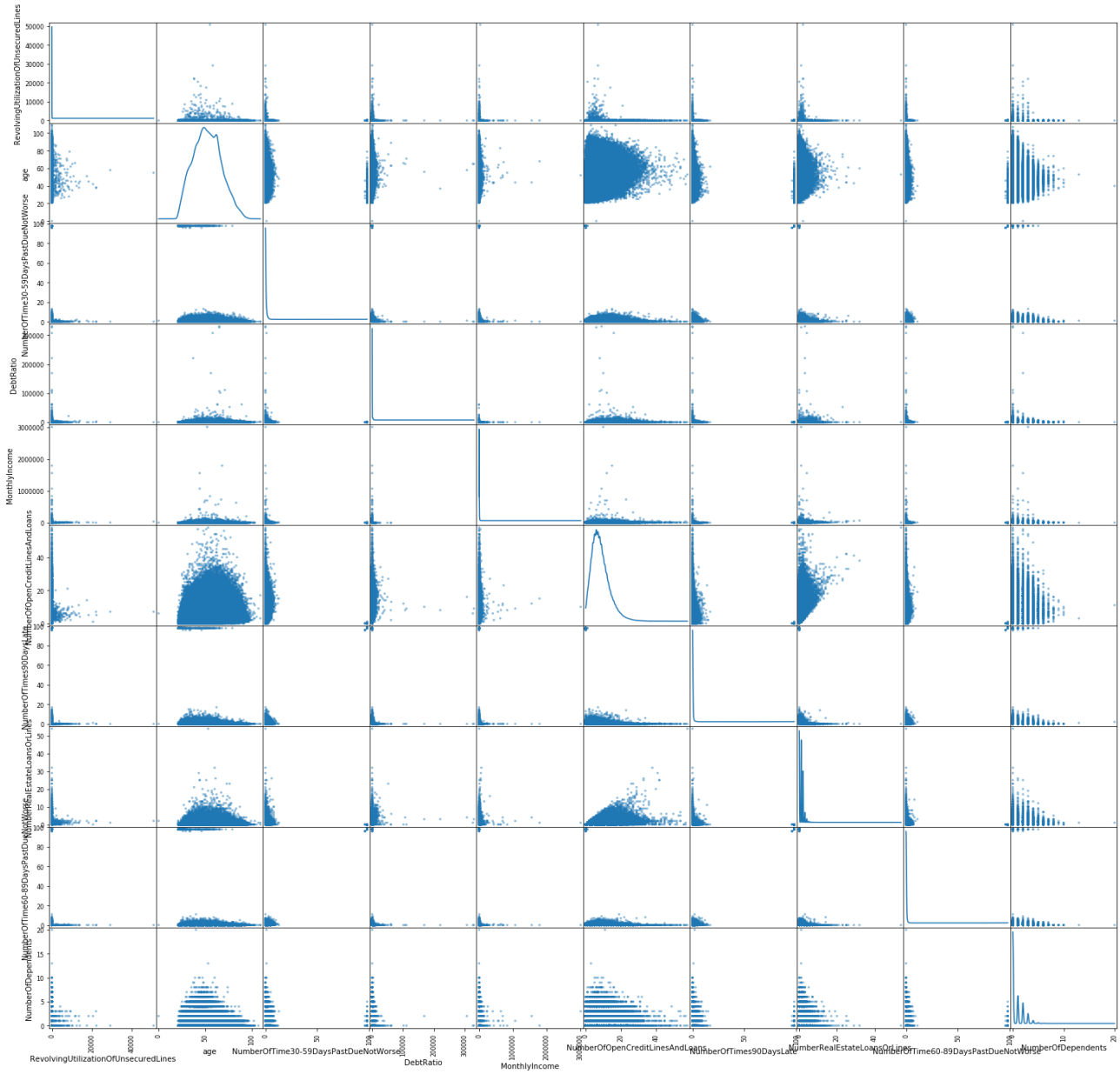
Pie chart for target variable (SeriousDlqin2yrs):



Histogram of all features:



Scatterplot:



3 most correlated features :

```
# Print highly correlated features
col = ['NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTime60-89DaysPastDueNotWorse', 'NumberOfTimes90DaysLate']
for i in range(len(col)):
    for j in range(len(col)):
        if i < j:
            print(col[i], col[j], np.corrcoef(features[col[i]], features[col[j]])[0][1])
```

```
('NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTime60-89DaysPastDueNotWorse', 0.987005447479946)
('NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTimes90DaysLate', 0.9836026812820766)
('NumberOfTime60-89DaysPastDueNotWorse', 'NumberOfTimes90DaysLate', 0.9927961825915981)
```

Observation:

1. Target variable is heavily biased i.e it contains only 6.684% (10026) values that are 0.
2. Most of the features are not normalized.
3. Three features namely "NumberOfTime30-59DaysPastDueNotWorse", "NumberOfTime60-89DaysPastDueNotWorse" and "NumberOfTimes90DaysLate" are highly correlated with correlation coefficient greater than 0.95.

○ Algorithms And Techniques:

I used the following classification algorithms in this project :-

1. **Logistic Regression :-** It is the most basic and easy to implement model and is always the first choice in most of the classification problems because of its simplicity. It requires lesser amount of time to train the model as compared to other classification models.

   Cost function for logistic regression can be written as :-

   $J(\theta) = -(1/m) \sum[y(i) * \log(h\theta(x(i))) + (1-y(i)) * \log(1-h\theta(x(i)))]$

2. **Ensemble methods :-** Ensemble algorithms are the next best choice. They are more robust to noise and outliers. Ensemble method builds multiple weak models using subset of same training data and then combine them to obtain a stronger model. There are two types of ensemble techniques - bagging and boosting.

   In bagging several subsets of data is created from training sample chosen randomly with replacement. Then each collection of subset data is used to train on decision trees. As a result, we end up with an ensemble of different models. Finally average of all the predictions from different trees are used which is more robust than a single decision tree.

   In boosting, we create a collection of predictors. In this technique, learners are learned sequentially with early learners fitting simple models to the data and then analyzing data for errors. In other words, we fit consecutive trees (random sample) and at every step, the goal is to solve for net error from the prior tree.

I used the following ensemble models :-

1. Random Forest Classifier
2. Gradient Boosting Classifier
3. Adaboost Classifier
4. Extra Trees Classifier

3. The next algorithm that I used is **XGBoost classifier** which is basically just an implementation of gradient boosting trees mainly designed for speed and performance. It further helps to reduce overfitting or variance in the data. The main purpose of using xgboost is that it is very fast and efficient.

4. Finally, I used **Multi-Layered Perceptron** for checking non-linearity in the data. It is basically an artificial feedforward neural network consisting of at least 3 layers of nodes and except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

○ Benchmark

I used Logistic regression as benchmark model on original dataset without any pre-processing or feature engineering. I only replaced all the null values with median because median is less sensitive to outliers as compared to mean.

Observations :-

1. AUC score obtained on training data = 0.69648388494
2. AUC score obtained on testing data = 0.708517391115
3. Difference in AUC value (train - test) = -0.0120335061752
4. Time taken to train the model = 1.56961679459 sec

● METHODOLOGY

○ Data Preprocessing

First I divided the dataset into training and testing subset using sklearn.model_selection.train_test_split. Training set consists of 80% of the data while testing set consists of 20% of the data.

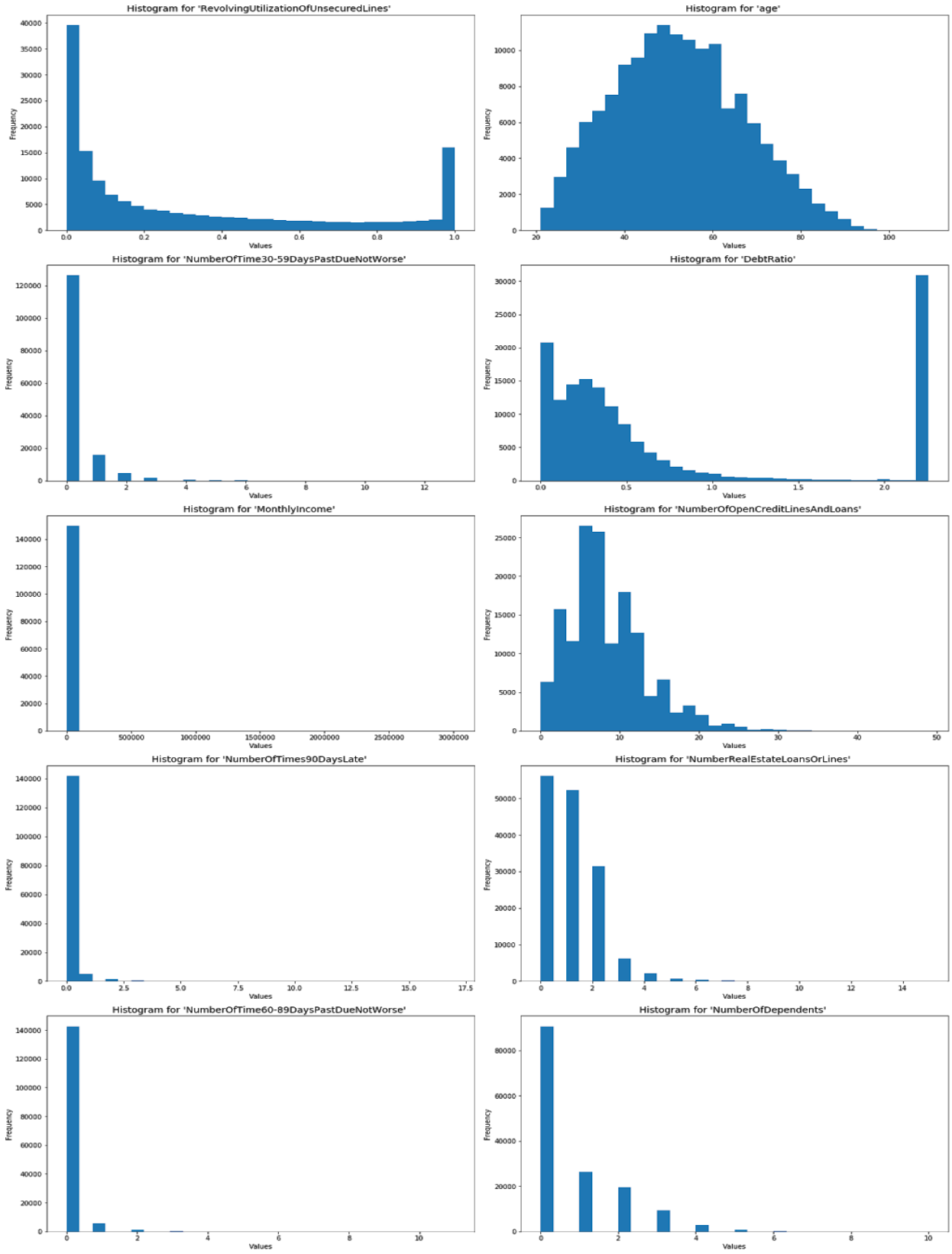
As mentioned above in Data Exploration section, there are outliers and null values in the data which needs to be processed. I cannot remove any row in the data as the target variable is heavily biased and removing any row might make it more biased. Therefore, the only option I had was to replace these values with some other value, preferably median.

So, except for the feature “DebtRatio”, I replaced all outlier values with the median of all the values of that feature or with highest value smaller than the smallest outlier. For “DebtRatio”, I followed Tukey’s method to detect and replace outlier values.

For null values:

1. NumberOfDependents :- I replaced all the null values with median of the data.
2. MonthlyIncome :- For this feature, I used regression algorithms to predict values to replace null values. I did this because I think that Monthly Income is a very important feature in predicting the credit risk of a person and hence replacing null values with any other constant value (say median) could make the data more biased. I experimented with many models and based on their R2 scores RandomForest proved to be the best model for prediction. I then tuned it and used it to predict all null values.

Histograms for all features after preprocessing is given below :-



○ Implementation

I implemented and tested the following algorithms :-

1. sklearn.linear_model.LogisticRegression
2. sklearn.ensemble.RandomForestClassifier
3. sklearn.ensemble.GradientBoostingClassifier
4. sklearn.ensemble.AdaBoostClassifier
5. sklearn.ensemble.ExtraTreesClassifier
6. xgboost.XGBClassifier
7. sklearn.neural_network.MLPClassifier

Steps performed in model implementation are as follows :-

1. I manually trained each model using some default and some self-defined parameters and also stored training time in a dictionary with model name as key and training time as value.
2. Then, I created another dictionary named as **clf_dict** that stores model name as keys and model object as value. This dictionary is passed as a parameter in a method described in step 3.
3. After this I defined a method **fun()** which takes a dictionary clf_dict (described in step 2), training subset (X_train and y_train) and testing subset (X_test and y_test) as parameters and it returns a dictionary having model name as key and a list of training_auc score, testing_auc score and their difference as value. Method fun() basically calculates the training and testing AUC scores obtained from each model using model_name.predict_proba() method.
4. Training_time and benchmark model score was also added to this dictionary for later analyzation and visualization.
5. The dictionary was then converted to dataframe.

Performance metric used = AUC as previously mentioned in **metrics** under **Definition** section.

Observations :-

Algorithm used	Train_AUC score	Test_AUC score	Difference	Training Time
LogisticRegression	0.803182	0.796517	0.006665	1.010580
RandomForestClassifier	0.999736	0.792652	0.207084	4.801014
GradientBoostingClassifier	0.870229	0.855535	0.014695	16.908223
AdaBoostClassifier	0.864172	0.849406	0.014766	7.560262
ExtraTreesClassifier	0.999983	0.790497	0.209486	2.314877
XGBClassifier	0.869793	0.855729	0.014064	5.342705
MLPClassifier	0.729234	0.725479	0.003755	13.878448
LogisticRegression (benchmark)	0.696484	0.708517	-0.012034	1.569617

This table indicates that the following 3 models scored an AUC score near or equal to 0.85 which is quite satisfactory as the author of the research paper also scored an AUC score of 0.85 before applying model tuning :-

1. GradientBoosting Classifier
2. AdaBoost Classifier
3. XGB Classifier

Also **MLP Classifier** and **Logistic Regression** did not perform well and **Random Forest** and **Extra Trees** may have overfitted the data which can clearly be seen by the difference in training and testing score.

○ Refinement

For refinement process, I used sklearn.model_selection.GridSearchCV.

As described above in section **Implementation** in **Methodology,** I used parameter tuning on these 3 algorithms :-

1. **AdaBoost Classifier :-** For this, I only used 2 parameters to tune the model :-
   - **n_estimators :-** The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

- **Learning Rate :-** It shrinks the contribution of each classifier by learning_rate.

```
from sklearn.model_selection import GridSearchCV

ab_param = {'n_estimators' : [100, 150, 200, 250], 'learning_rate' : [0.1, 0.3, 0.5, 0.7]}

ab_grid = GridSearchCV(estimator = ab_clf, param_grid = ab_param, scoring = 'roc_auc',
                       n_jobs = 1, cv = 3, verbose = 0)

ab_grid.fit(X_train, y_train)
```

Observations :-

1. AUC score before tuning the model :- 0.849406
2. AUC score after tuning the model :- 0.850970
3. Gain in AUC score :- 0.850970 - 0.849406 = 0.001564

2. **Gradient Boosting Classifier :-** I used the following parameters to tune the model.
   - **Loss:-** It is basically a loss function which needs to be optimized.
   - **Learning Rate :-** It is used to shrink the learning rate of each tree.
   - **n_estimators :-** It defines the number of boosting stages to perform.
   - **Min_samples_leaf :-** The minimum number of samples required to be at a leaf node.

```
gb_param = {'loss' : ['deviance', 'exponential'],
            'learning_rate' : [0.1, 0.15, 0.2], 'n_estimators' : [100, 150, 200],
            'min_samples_leaf' : [1, 3, 5]}

gb_grid = GridSearchCV(estimator = gb_clf, param_grid = gb_param, scoring = 'roc_auc',
                       n_jobs = 1, cv = 3, verbose = 0)

gb_grid.fit(X_train, y_train)
```

Observations :-

- AUC score before tuning the model :- 0.855535
- AUC score after tuning the model :- 0.856054
- Gain in AUC score :- 0.856054 - 0.855535= 0.000519

3. **XGBoost Classifier :-** I used the following parameters to tune the model :-
   - **n_estimators :-** Number of boosted trees to fit.
   - **Learning Rate :-** Boosting learning rate.

```
xgb_param = {'n_estimators' : [150, 180, 210, 240], 'learning_rate' : [0.1, 0.15, 0.2]}

xgb_grid = GridSearchCV(estimator = xgb_clf, param_grid = xgb_param, scoring = 'roc_auc',
                        n_jobs = 1, cv = 3, verbose = 0)

xgb_grid.fit(X_train, y_train)
```

Observations :-

- AUC score before tuning the model :- 0.855729
- AUC score after tuning the model :- 0.856454
- Gain in AUC score :- 0.856454- 0.855729 = 0.000725

● Results

○ Model Evaluation and Validation

XGBoost Classifier has proved to be the best model with AUC score of 0.856454.

Parameters of untuned model :-

1. N_estimators = 100
2. Learning rate = 0.1

Parameters of tuned model :-

1. N-estimators :- 150
2. Learning rate = 0.1

For checking **robustness** of the model, I plotted feature_importance graph of tuned xgboost model and noticed that 3 of the features namely **'NumberOfTime60-89PastDueNotWorse', 'NumberRealEstateLoansOrLines'** and **'NumberOfDependents'** are the least important features with F1 score less than 100 while the other features have F1 score above 100.

So, I trained and tested my tuned model on 7 out of 10 features, thus ignoring the above mentioned features.

Observations :-

Features used	Train AUC	Test AUC	Difference	Training Time
10	0.872339	0.856454	0.015885	7.993341 sec
7	0.845669	0.827017	0.018652	6.660305 sec

It can clearly be seen that after reducing 3 features out of 10 features, testing AUC score only reduced by 0.029437 and is above 0.80 which is quite satisfactory.

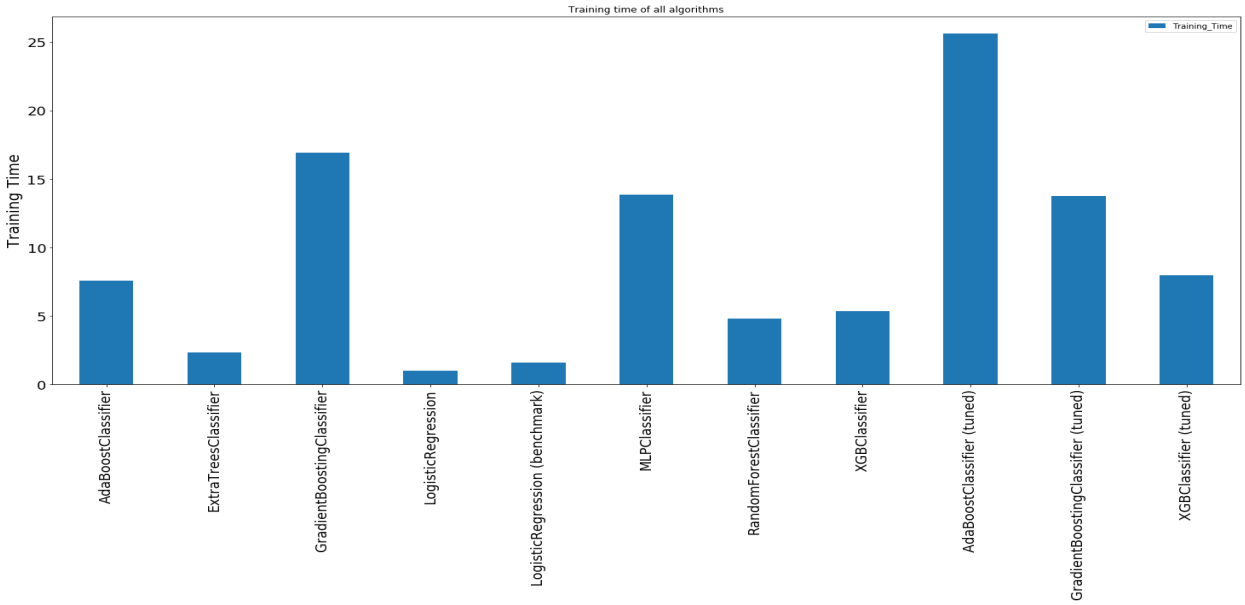
- **Justification**

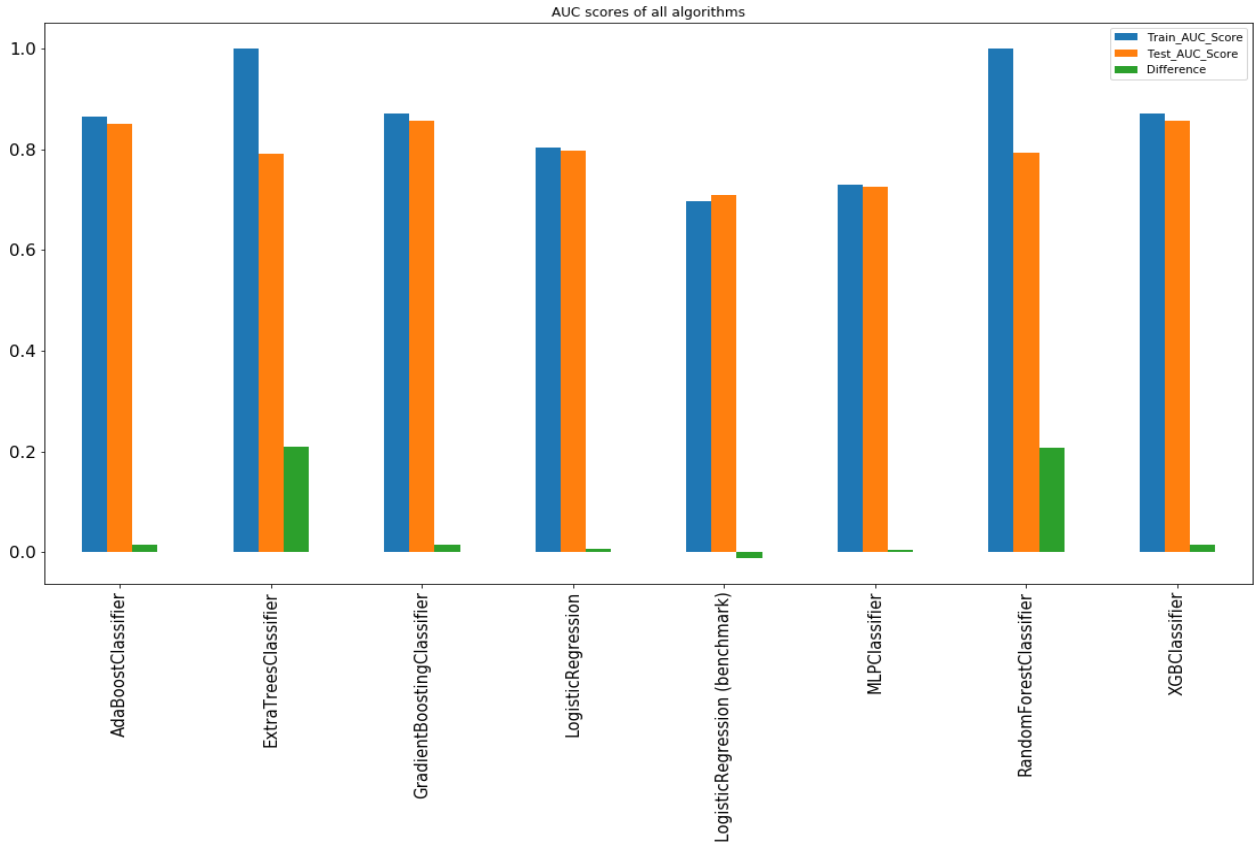
	Benchmark Model (Logistic Regression)	Final Model (XGBoost Classifier)	Difference
Training AUC score	0.696484	0.872340	0.175856
Testing AUC score	0.708517	0.856454	0.147937
Training time	1.569617 sec	7.993342 sec	6.423725 sec

The above table clearly indicates that result of final model is stronger than the benchmark model and therefore the final model has solved the problem posed in this project.

- # Conclusion
    - ## Free-Form Visualization

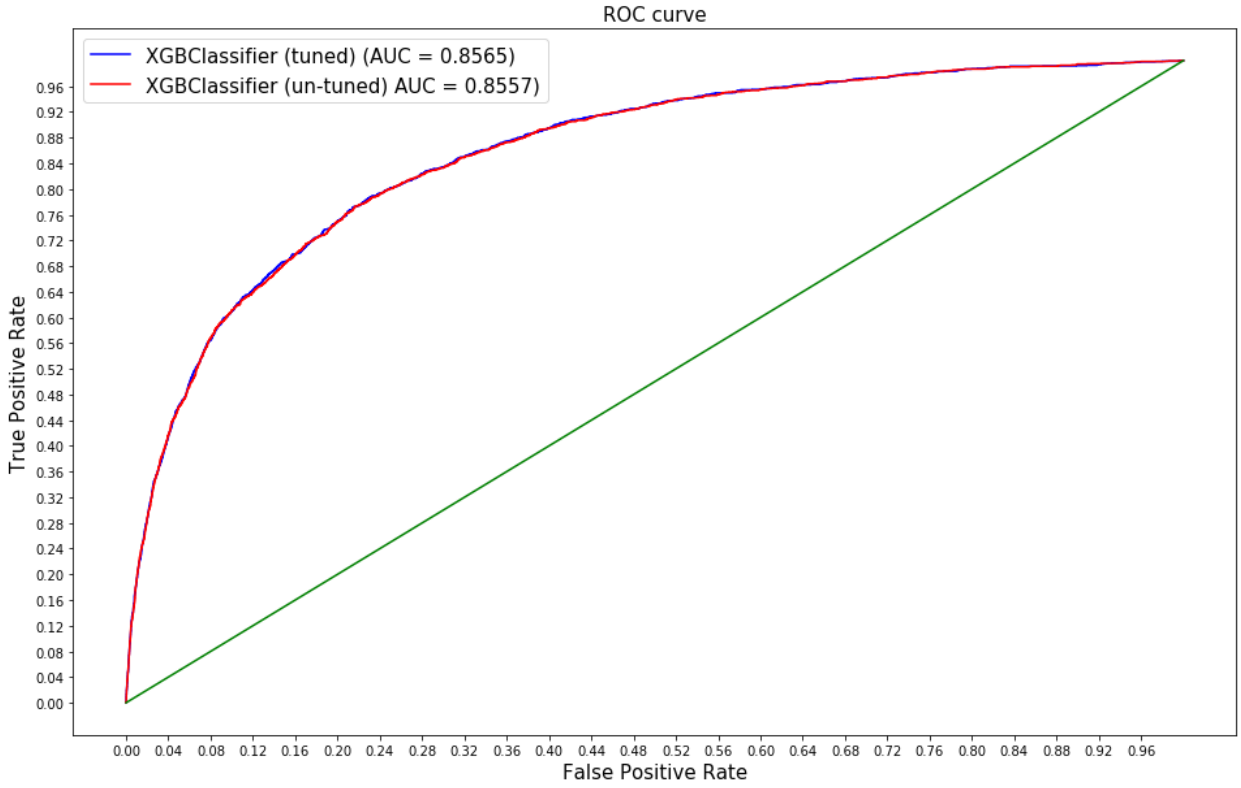
A plot comparing each algorithm with respect to training time and AUC scores is given below :-





According to the these two graphs and as mentioned above in **Results** section, XGBoost classifier has proved to be the best model because it gave the best AUC score on test data and difference in training and testing score is also less which clearly indicates that it does not overfit the data. Its training time is also less and comparable to other models. Although Gradient Boosting also performed well but it scored slightly lesser than XGBoost in terms of AUC score but it also took more training time than XGBoost.

FPR vs TPR graph of XGBoost Classifier :-

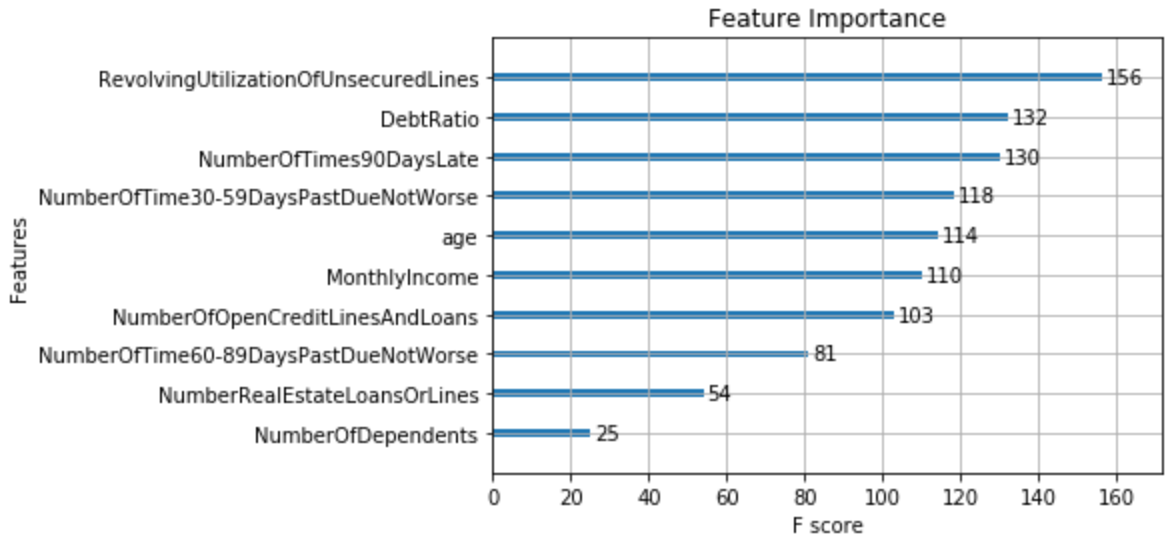


Feature importance :-

According to the best model (XGBoost Classifier), the feature importance is given below :-

1. RevolvingUtilizationOfUnsecuredLines
2. DebtRatio
3. NumberOfTimes90DaysLate
4. NumberOfTime30-59DaysPastDueNotWorse

5. Age
6. MonthlyIncome
7. NumberOfOpenCreditLinesAndLoans
8. NumberOfTime60-89DaysPastDueNotWorse
9. NumberRealEstateLoansOrLines
10. NumberOfDependents



It tells us that an individual's total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits is the most deciding criteria of whether the individual will experience financial distress or not during the next two years and number of dependents the individual have in the family excluding himself/herself is the least deciding factor of the person's financial level.

- **Reflection**

In this project, I tried to build a model for peer-to-peer lending managers or banks or credit companies so that they can easily access the default risk of their clients. The project is a binary classification problem in which the goal is to predict whether an individual will face financial distress during the next two years or not.

This project can be summarized in the following steps as given below :-

1. Searching and selecting an interesting project on Kaggle related to finance or credit system and deciding between classification or regression problems.
2. Downloading the dataset and then analysing and visualising various patterns in the dataset.
3. Pre processing the data and applying feature engineering like PCA.
4. Selecting suitable algorithms according to nature of the problem and patterns in the data.
5. Defining a benchmark model.
6. Applying all the selected algorithms.
7. Hyper-parameter tuning on the best model and then testing the tuned model on testing data.
8. Visualizing all the results and applying feature importance.

During the entire process, data pre-processing and selection of best algorithms were both the most difficult and interesting steps. It was difficult to select the range of outliers from the data as removing even a single wrong value from the data could greatly affect the final result. I tried many approaches, tried to remove a different range of outliers from the data and each try gave me a different result. The approach that I have discussed above gave me the best score.

It was also very interesting that 3 out of 10 features were highly correlated to each other with correlation coefficient more than 0.95. But when I removed 2 out of 3 correlated features, my overall AUC score reduced by approximately 0.045. Therefore, I decided not to remove any feature.

Another challenging task for me was to decide appropriate parameters for the models. Each different value of parameter also resulted in a different score.

But what I found most interesting is the result obtained from my benchmark model (logistic regression). I trained my benchmark model on original dataset without any pre-processing or feature engineering and got an AUC score of 0.69 while my best model gave me the AUC score of 0.85. This is really interesting to note that after applying simple pre-processing techniques on data set, I was able to increase my score by 0.16.

○ Improvement

Few of the ways by which the performance of the model can be increased are :-

1. Using more robust outlier detection and removal methods.
2. Using more sophisticated feature engineering techniques.
3. Selecting and experimenting with other algorithms.
4. Adding more number of parameters in our final model.

● References

1. https://www.sciencedirect.com/science/article/pii/S095741741101342X?via%3Dihub
2. http://cs229.stanford.edu/proj2014/Marie-Laure%20Charpignon,%20Enguerrand%20Horel,%20Flora%20Tixier,%20Prediction%20of%20consumer%20credit%20risk.pdf
3. https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
4. https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/
5. https://xgboost.readthedocs.io/en/latest/python/python_api.html
6. http://datapigtechnologies.com/blog/index.php/highlighting-outliers-in-your-data-with-the-tukey-method/
7. https://www.investopedia.com/terms/c/creditrisk.asp