Machine Learning Engineer Nanodegree

Capstone Project

Rabin Aryal

July 11th, 2018

I. Definition

Project Overview

Financial distress is the term used in corporate finance to indicate a condition when business, household, or individual runs into a tight cash situation and cannot pay the owed amount to creditor on a due date. Financial institutions use a credit scoring algorithm, which determines what is the probability of credit default.

Machine learning techniques can be an important tool for financial risk management. [1] Predictive machine learning models are able to predict the probability of credit default could reduce the risk for credit originators. Better credit score systems could lead to lower overall borrowing costs. This could benefit both lenders and borrowers.

The data that is being used to predict the somebody will experience financial distress will be used from Kaggle competition website Give Me Some Credit - data.

Problem Statement

Banks play a crucial role in market economies. They decide who can get finance and on what terms and can make or break investment decisions.[2] For markets and society to function, individuals and companies need access to credit. Credit scoring algorithms, which make a guess at the probability of default, are the method banks use to determine whether or not a loan should be granted. The credit scoring algorithm can be improved using a machine learning that will predict the probability that somebody will experience financial distress in the next two years. The proposed model will be supervised classification [3] problem as the output is credit score (probability) that the customer will experience financial distress next two years based based on the features in the dataset as input. Based on the probability score, creditor as well as a borrower can make a wise decision whether to give/take loan or not.

Metrics

The model prediction for this problem can be evaluated in several ways. Since the official evaluation of this project is done by Kaggle using AUC, area under curve (higher it is, better the model), same will be used for evaluation of models.

An **ROC curve** [4] (**receiver operating characteristic curve**) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

True Positive Rate (**TPR**) is a synonym for recall and is therefore defined as follows:

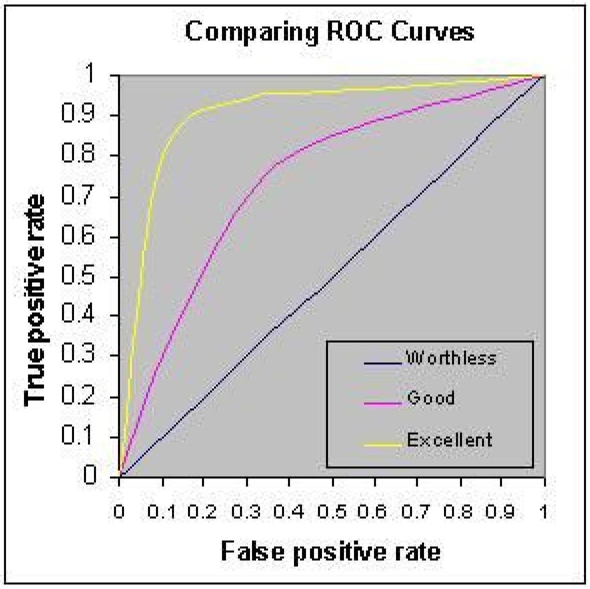
$$TPR = TP / (TP + FN)$$

False Positive Rate (**FPR**) is defined as follows:

$$FPR = FP / (FP + TN)$$

An ROC curve plots TPR vs. FPR at different classification thresholds. Lowering the classification threshold classifies more items as positive, thus increasing both False Positives and True Positives.

AUC [5] stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1).



AUC ranges in value from 0 to 1. A model whose predictions are 100% wrong has an AUC of 0.0; one whose predictions are 100% correct has an AUC of 1.0.

AUC is desirable for the following two reasons:

- AUC is **scale-invariant**. It measures how well predictions are ranked, rather than their absolute values.
- AUC is **classification-threshold-invariant**. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

II. Analysis

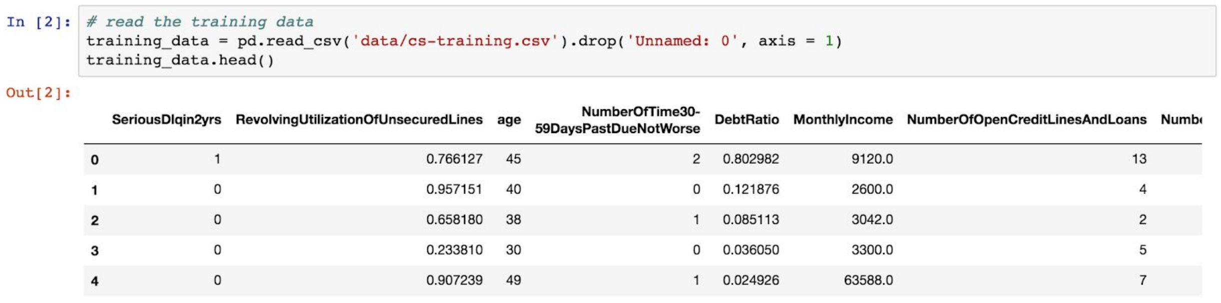
Data Exploration

The data is available from the Kaggle competition website Give Me Some Credit - data. The data includes

1. cs-training.csv (7.21 mb)
2. cs-test.csv (4.75 mb)
3. Data Dictionary.xls(14.50 kb)
4. sampleEntry.csv (1.82 mb)

The training and test dataset contains eleven features that is used to determine the probability that the customer will default the loan. Some of those features are as follows:

- Serious delinquency in 2 years
- Revolving utilization of unsecured line
- Age
- Debt ratio
- Monthly Income
- Number of open credit lines and loans
- Number of dependent



```
In [2]: # read the training data
        training_data = pd.read_csv('data/cs-training.csv').drop('Unnamed: 0', axis = 1)
        training_data.head()
```

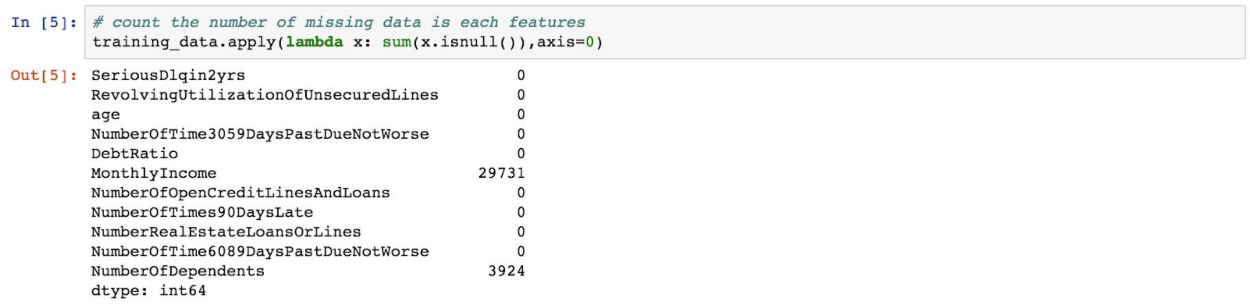
Out[2]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	Numb
0	1	0.766127	45	2	0.802982	9120.0	13	
1	0	0.957151	40	0	0.121876	2600.0	4	
2	0	0.658180	38	1	0.085113	3042.0	2	
3	0	0.233810	30	0	0.036050	3300.0	5	
4	0	0.907239	49	1	0.024926	63588.0	7	

The training dataset consist of 150,000 borrowers, and the test dataset contains 101,503 observations, for a sum of over 250,000 borrowers gathered from historical data. The outcome will be the probability that the customer will default the loan in the next two years. The 30% of the training dataset will be used as validation dataset to test the model.

Exploratory Visualization

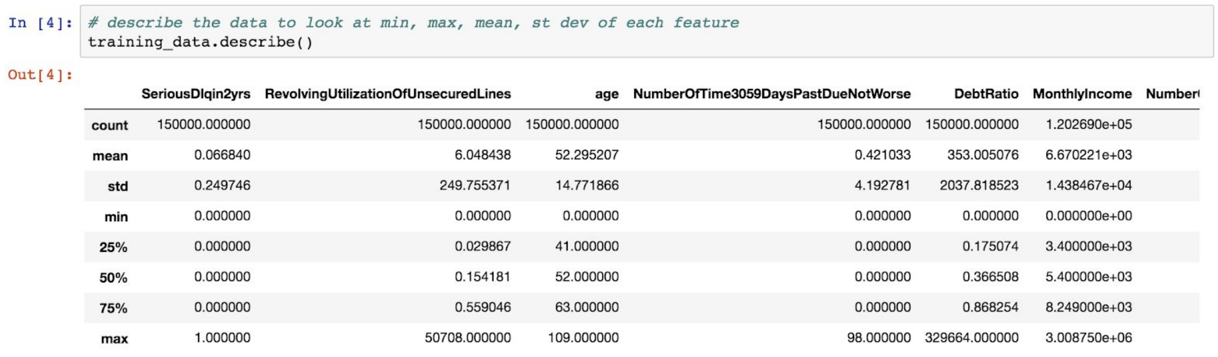
At quickly glancing the data, I notice that some of the feature have a missing or incorrect data. To be specific, “Monthly Income” and “Number of dependent” have missing data. The two column seems to be very important feature on determining a probability score, so they can not be discarded. The option here will be to replaced the data with a median value for each feature. which may be discarded while implementing the model.



```
In [5]: # count the number of missing data is each features
        training_data.apply(lambda x: sum(x.isnull()),axis=0)

Out[5]: SeriousDlqin2yrs                                0
        RevolvingUtilizationOfUnsecuredLines            0
        age                                             0
        NumberOfTime3059DaysPastDueNotWorse             0
        DebtRatio                                       0
        MonthlyIncome                               29731
        NumberOfOpenCreditLinesAndLoans                 0
        NumberOfTimes90DaysLate                         0
        NumberRealEstateLoansOrLines                    0
        NumberOfTime6089DaysPastDueNotWorse             0
        NumberOfDependents                           3924
        dtype: int64
```

The feature “Number of times borrower has been 90 days or more past due” have some incorrect data of value 98, which seems not to be true. The median for this feature seems to be 0 and does not make replace the incorrect data with median. I was thinking to discard this feature on my model initially but on further analysis replacing outlier with the value that is max value after discarding the outlier value would not change the result of the model .

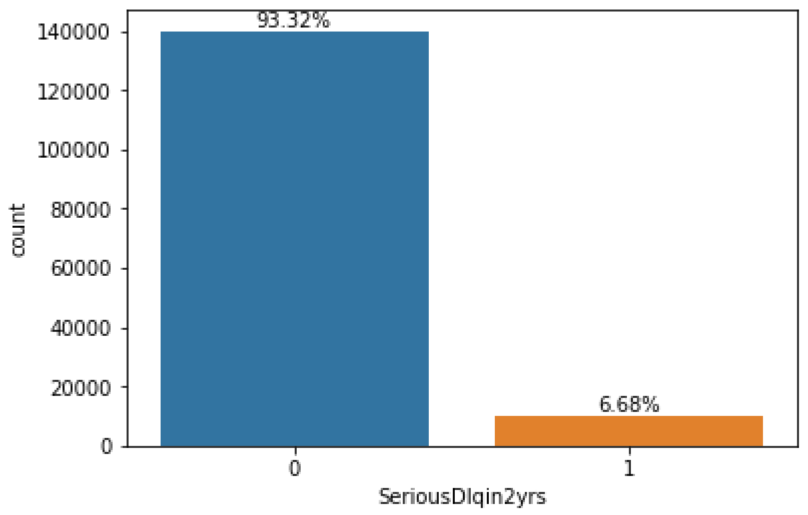
Below is some statistic on the data



```
In [4]: # describe the data to look at min, max, mean, st dev of each feature
        training_data.describe()

Out[4]:
```

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime3059DaysPastDueNotWorse	DebtRatio	MonthlyIncome	Number
count	150000.000000	150000.000000	150000.000000	150000.000000	150000.000000	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	329664.000000	3.008750e+06	

The feature “Serious delinquency in 2 years” will be target variable and will be discarded from the feature that determines the probability score. At the end, the probability score can be compared with the “Serious delinquency in 2 years”. Out of 150,000 training dataset, “Serious delinquency in 2 years” consists 139,974 (93.3%) records that are predicted to not have any financial distress in next 2 year and 10,026 (6.7%) records that are predicted to have any financial distress in next 2 year. The dataset is highly unbalanced and “Serious delinquency in 2 years” cannot be used to evaluate the performance.



Algorithms and Techniques

Due to the nature in machine learning, a prediction model would be developed using several machine learning algorithms on this dataset to see which will give the best performance and will be judged against a predefined evaluation metric. Binary classification [6] algorithms including random forests, bayesian networks, and logistic regression will be evaluated. XGBoost [7](trusted) and LightGBM [8](fast) algorithm will be evaluated in order to compare with prediction model using binary classification.

Random Forest:

Ensemble Methods combine the results of many base models (weak learners), can handle categorical variables and outliers very well, and can handle non-linear relationship between features. Although its relatively slower to train compared to Naive Bayes or Logistic Regression.

In this project I choose Random Forest, which combines the results of many decision trees. Decision trees area 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.

Decision trees [9] tend to overfit the data. In the extreme case of a decision tree having one leaf for every data point, the training precision will be 100%, but the result won't be able to generalize. Random forest combines multiple such decision trees and optimize the combined model. As discussed above, the major problem for decision trees and random forest is overfitting. There are several parameters to tune in random forest implementation in scikit-learn: n_estimators (number of trees to grow), min_samples_split (minimum number of data points to split a tree), min_samples_leaf (minimum number of data points for leaf). In addition, I also plan to tune max_features (maximum number of features to use to split tree), bootstrap (whether bootstrap samples are used when building trees).

Gaussian Naive Bayes:

Naive Bayes [10] methods are a set of supervised learning algorithms based on applying Bayes' theorem with the "naive" assumption of independence between every pair of features. In spite of their apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many real-world situations, famously document classification and spam filtering.

In this project I intend to use Gaussian Naive Bayes classification method implemented in scikit-learn.Gaussian Naive Bayes assumes the likelihood of the features to be Gaussian. There's only one parameter for the algorithm: the class prior. The default value for class prior is the probability of each class, which is a reasonable assumption here.

Logistic Regression:

Logistic Regression [11] measures the relationship between the dependent variable (our label, what we want to predict) and the one or more independent variables (our features), by estimating probabilities using it's underlying logistic function.

One of the problem with logistic regression is that coefficients are unbound and can take extreme values, making other features useless and model unstable. There are several ways to introduce regularization. Here I use the logistic regression method implemented in scikit-learn, in which and penalty are the regularization parameters. Penalty takes two choices: L1 and L2 regularization with the default value being L2 regularization.

XGBoost:

XGBoost [7] is short for eXtreme gradient boosting. It is a library designed and optimized for boosted tree algorithms. Boosting is an ensemble technique where new models are added to correct the errors made by existing models. Models are added sequentially until no further improvements can be made. XGBoost is also being widely-used in many winning solutions of machine learning competitions.

It's main goal is to push the extreme of the computation limits of machines to provide a scalable, portable and accurate for large scale tree boosting. In this project I choose XGBoost because of its performance and technique where new models are added to correct the errors made by existing models.

LightGBM:

LightGBM [8] is a gradient boosting framework that uses tree based learning algorithms. It is designed to be distributed and efficient with the following advantages:

- Faster training speed and higher efficiency
- Lower memory usage
- Better accuracy
- Parallel and GPU learning supported
- Capable of handling large-scale data

Benefit from these advantages, LightGBM is also being widely-used in many winning solutions of machine learning competitions.

Comparison Experiments [12] on public datasets show that LightGBM can outperform existing boosting frameworks on both efficiency and accuracy, with significantly lower memory consumption. What's more, the parallel experiments [13] show that LightGBM can achieve a linear speed-up by using multiple machines for training in specific settings. In this project I choose LightGBM because of its advantages and also to compare with other trusted boosting framework XGBoost.

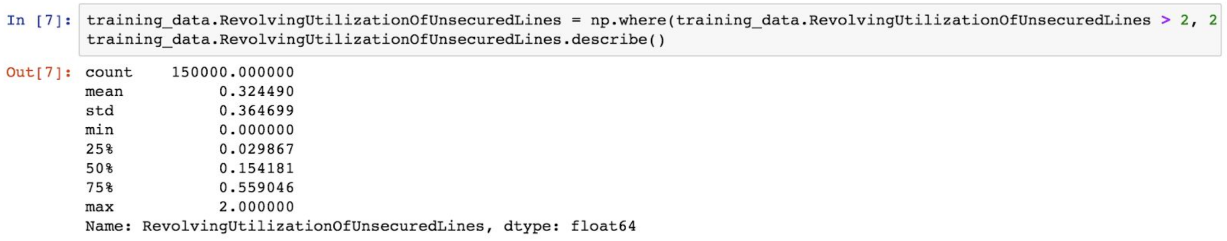
Benchmark

As this is a Kaggle competition a benchmark model is evaluated for area under curve [5], which is a metric for binary classifier. I will run the random forest classifier to get a base AUC. Then, I can compare our next model with it to see of it can beat it and by what extent. I will take the best model and for satisfying the curiosity, run it on test set provided by Kaggle as test dataset. A personal goal would be to be get a AUC of 86% (0.86000) or greater.

III. Methodology

Data Preprocessing

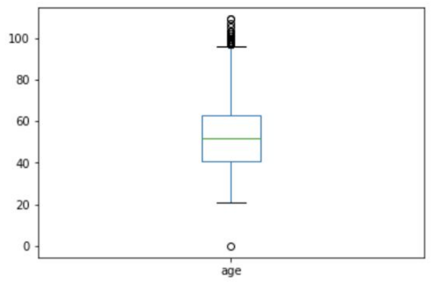
The max value for “Revolving utilization of unsecured line” seems to be outlier as it is off from min, 25%, 50% and 75% by a much. Replacing outlier value greater than 2 with 2 will not change the data and won’t affect the final outcome.



```
In [7]: training_data.RevolvingUtilizationOfUnsecuredLines = np.where(training_data.RevolvingUtilizationOfUnsecuredLines > 2, 2
        training_data.RevolvingUtilizationOfUnsecuredLines.describe()

Out[7]: count    150000.000000
        mean          0.324490
        std           0.364699
        min           0.000000
        25%           0.029867
        50%           0.154181
        75%           0.559046
        max           2.000000
        Name: RevolvingUtilizationOfUnsecuredLines, dtype: float64
```

The age (Age of borrower in years) seems to have a lot of 0 value. It seems to be an outlier and will need to be replaced with next minimum after ignoring 0.

```
In [8]: training_data.age.plot.box()

Out[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1a0b236d30>
```



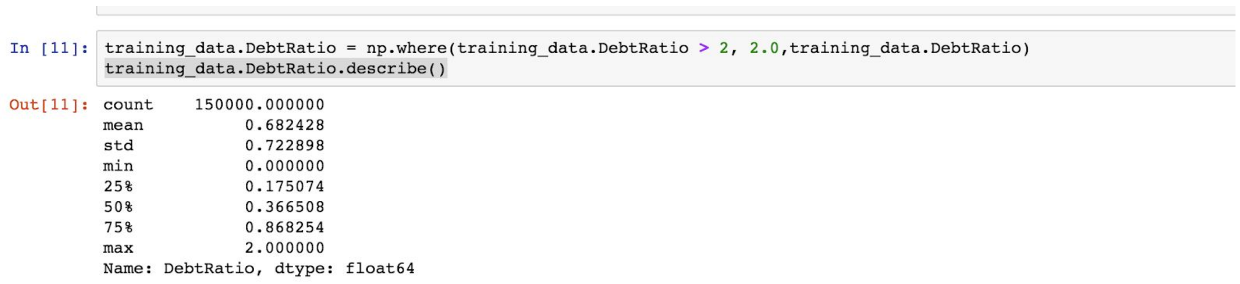
```
In [9]: training_data.age = np.where(training_data.age == 0, 21, training_data.age)
        training_data.age.describe()

Out[9]: count    150000.000000
        mean         52.295347
        std          14.771470
        min          21.000000
        25%          41.000000
        50%          52.000000
        75%          63.000000
        max         109.000000
        Name: age, dtype: float64
```

Similarly, all the features were visualized and the outlier were fixed and final value are below.

There seems to be few outlier for “Number Of Time 30-59 Days Past Due Not Worse” . The min, 25%, 50% and 75% data is 0 and so is the median. Replacing the outlier with the max value after outlier is ignored will not have any impact on the final prediction.

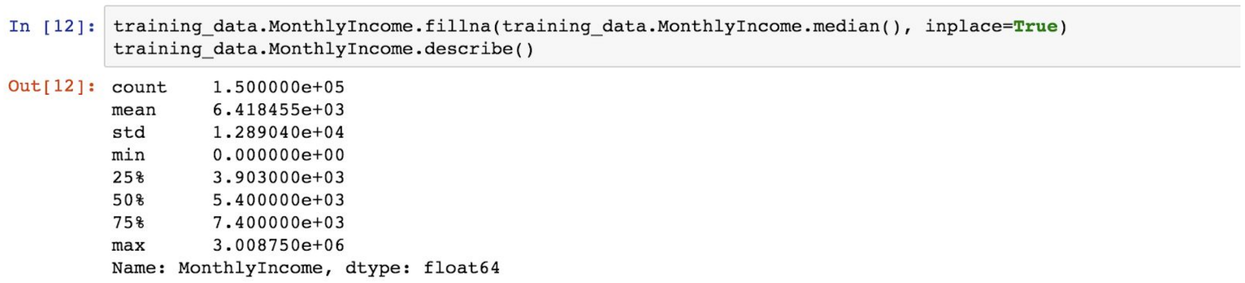
```
In [29]: training_data.NumberOfTime3059DaysPastDueNotWorse = np.where(training_data.NumberOfTime3059DaysPastDueNotWorse > 13 , 1
         training_data.NumberOfTime3059DaysPastDueNotWorse.describe()

Out[29]: count    150000.000000
         mean          0.268667
         std           0.881603
         min           0.000000
         25%           0.000000
         50%           0.000000
         75%           0.000000
         max          13.000000
         Name: NumberOfTime3059DaysPastDueNotWorse, dtype: float64
```

There seems to be few outlier for “Debt Ratio“ as the max is 329664.00 which is much greater than median and 75%.Replacing outlier value that is any value greater than 2 with 2 will not change the data.



```
In [11]: training_data.DebtRatio = np.where(training_data.DebtRatio > 2, 2.0,training_data.DebtRatio)
         training_data.DebtRatio.describe()

Out[11]: count    150000.000000
         mean          0.682428
         std           0.722898
         min           0.000000
         25%           0.175074
         50%           0.366508
         75%           0.868254
         max           2.000000
         Name: DebtRatio, dtype: float64
```

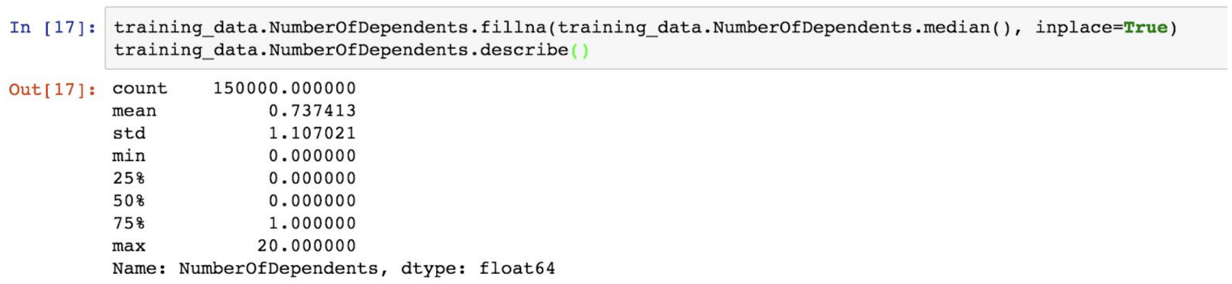
There are 29731 missing value for “monthly income”. So, replacing them with median.



```
In [12]: training_data.MonthlyIncome.fillna(training_data.MonthlyIncome.median(), inplace=True)
         training_data.MonthlyIncome.describe()

Out[12]: count    1.500000e+05
         mean     6.418455e+03
         std      1.289040e+04
         min      0.000000e+00
         25%      3.903000e+03
         50%      5.400000e+03
         75%      7.400000e+03
         max      3.008750e+06
         Name: MonthlyIncome, dtype: float64
```

Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards), Number of times borrower has been 90 days or more past due, Number of mortgage and real estate loans including home equity lines of credit and Number of times borrower has been 60-89 days past due but no worse in the last 2 years all had a outlier and was replaced with a value that was max after ignoring the outlier.

“Number of dependents” in family excluding themselves (spouse, children etc.) had 3924 missing vale and was replaced by median.

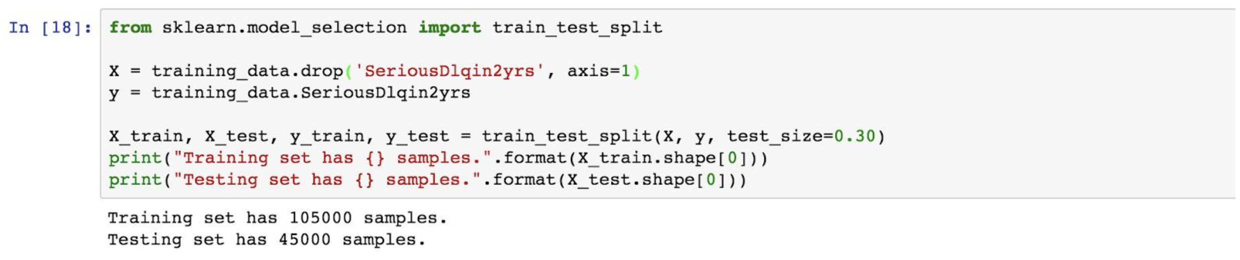


```
In [17]: training_data.NumberOfDependents.fillna(training_data.NumberOfDependents.median(), inplace=True)
         training_data.NumberOfDependents.describe()

Out[17]: count    150000.000000
         mean          0.737413
         std           1.107021
         min           0.000000
         25%           0.000000
         50%           0.000000
         75%           1.000000
         max          20.000000
         Name: NumberOfDependents, dtype: float64
```

Implementation

The implementation process consists of

Splitting training data into training set and validation set (70% of the data will be used for training and 30% for validation)



```
In [18]: from sklearn.model_selection import train_test_split

         X = training_data.drop('SeriousDlqin2yrs', axis=1)
         y = training_data.SeriousDlqin2yrs

         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30)
         print("Training set has {} samples.".format(X_train.shape[0]))
         print("Testing set has {} samples.".format(X_test.shape[0]))

         Training set has 105000 samples.
         Testing set has 45000 samples.
```

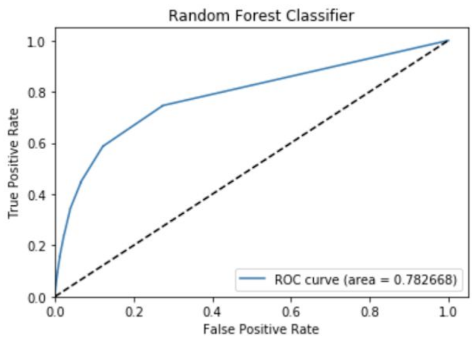
Perform machine learning with default parameter for the below algorithms and record training and testing scores

Random Forest:

The scikit-learn random forest is used with default parameter and produced the AUC score of 0.782688.

```
In [21]: rf_Model = RandomForestClassifier()
         rf_Model.fit(X_train, y_train)
         rf_Model.score(X_test, y_test)

         display_accuracy_report_plot("Random Forest Classifier", rf_Model,  X_train, X_test, y_train, y_test)
```

```
Train Accuracy :  0.989971428571
Test Accuracy  :  0.933044444444
             precision    recall  f1-score   support

          0       0.94      0.99      0.96     42032
          1       0.48      0.16      0.24      2968

avg / total       0.91      0.93      0.92     45000
```

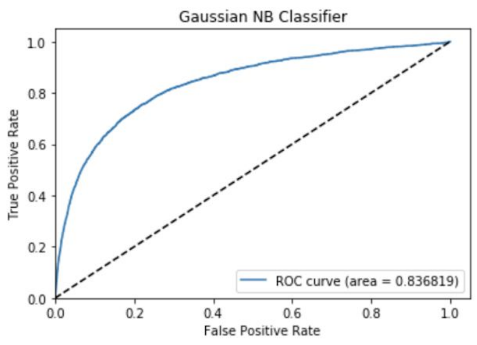


Gaussian Naive Bayes:

The scikit-learn GaussianNB is used with default parameter and produced the AUC score of 0.836819

```
In [22]: nb_Model = GaussianNB()
         nb_Model.fit(X_train, y_train)
         nb_Model.score(X_test, y_test)


         display_accuracy_report_plot("Gaussian NB Classifier", nb_Model,  X_train, X_test, y_train, y_test)
```

```
Train Accuracy :  0.930066666667
Test Accuracy  :  0.930622222222
             precision    recall  f1-score   support

          0       0.95      0.98      0.96     42032
          1       0.46      0.29      0.35      2968

avg / total       0.92      0.93      0.92     45000
```

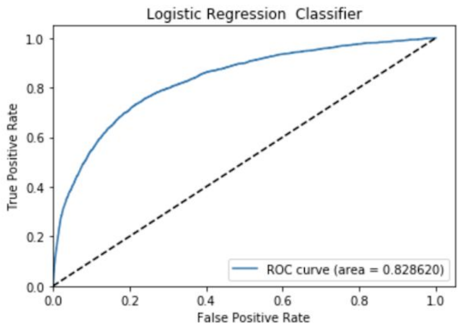


Logistic Regression

The scikit-learn Logistic Regression is used with default parameter and produced the AUC score of 0.828620

```
In [23]: lr_Model  =  LogisticRegression()
         lr_Model.fit(X_train, y_train)
         lr_Model.score(X_test, y_test)


         display_accuracy_report_plot("Logistic Regression  Classifier", lr_Model,  X_train, X_test, y_train, y_test)
```

```
Train Accuracy :  0.934828571429
Test Accuracy  :  0.935933333333
             precision    recall  f1-score   support

          0       0.94      1.00      0.97     42032
          1       0.60      0.09      0.15      2968

avg / total       0.92      0.94      0.91     45000
```



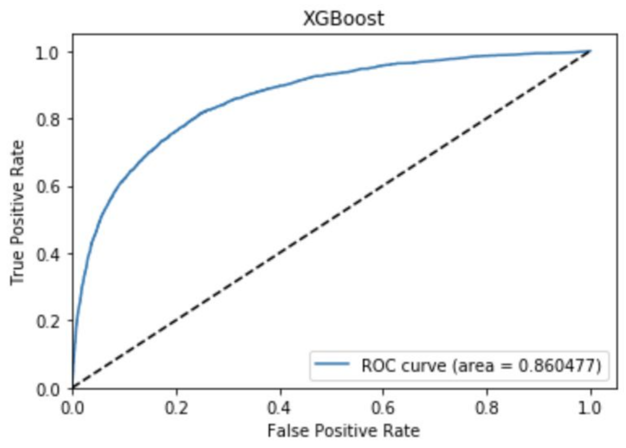
XGBoost

The XGBoost is used with ‘eval_metric’ parameter for ‘auc’ and produced the AUC score of 0.860447. It took 2.50 sec to train the model.

```
In [23]: params = {
                 'eval_metric': 'auc'
             }
         num_boost_round = 50
         early_stopping_rounds = 20
         test_size = 0.1

         dtrain = xgb.DMatrix(X_train, y_train)
         dtest = xgb.DMatrix(X_test, y_test)

         watchlist = [(dtrain, 'train'), (dtest, 'eval')]

         start_time = dt.datetime.now()
         print("Start time: ",start_time)
         xgb_gbm = xgb.train(params, dtrain, num_boost_round, evals=watchlist,
                             early_stopping_rounds=early_stopping_rounds,
                             verbose_eval=True)
         print("Time took to train: ", dt.datetime.now()-start_time)

         prediction = xgb_gbm.predict(xgb.DMatrix(X_test), ntree_limit=xgb_gbm.best_iteration+1)


         print ("Test Accuracy  : ", accuracy_score(y_test, prediction.round()))

         display_classification_report(y_test, prediction.round())
         plot_roc ("XGBoost", y_test, prediction)
```

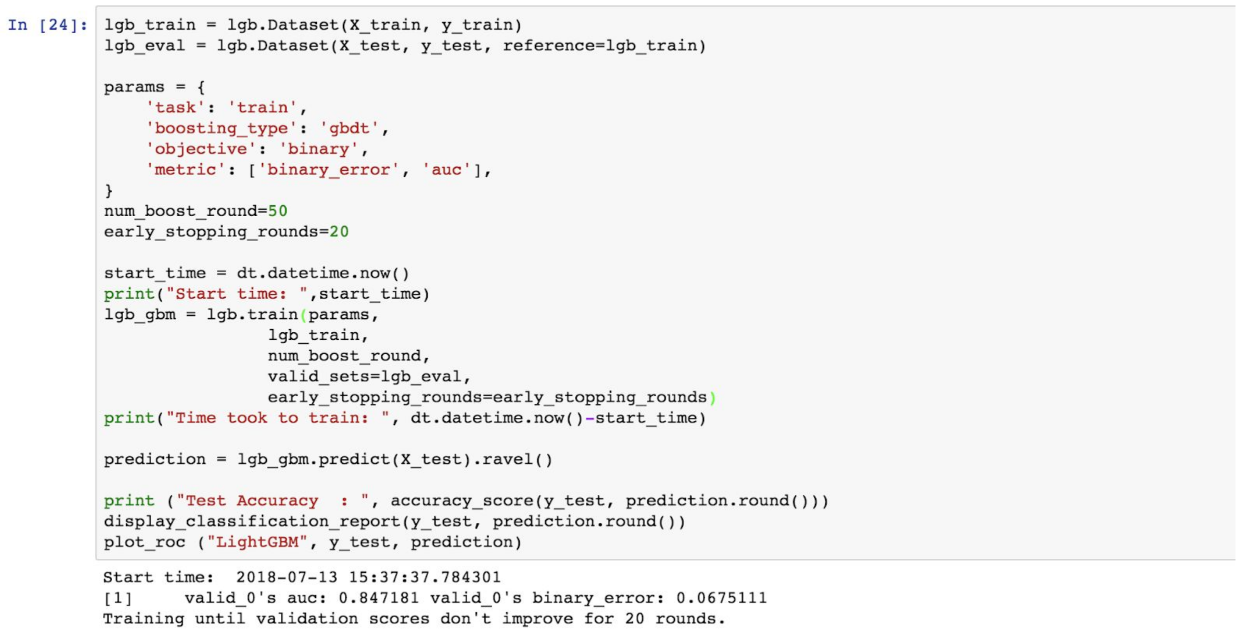
```
Start time:  2018-07-13 15:37:35.108781
[0]     train-auc:0.848631      eval-auc:0.846728
Multiple eval metrics have been passed: 'eval-auc' will be used for early stopping.
```

```
Time took to train:  0:00:02.497283
Test Accuracy  :  0.937177777778
             precision    recall  f1-score   support

          0       0.94      0.99      0.97     41962
          1       0.61      0.19      0.29      3038

avg / total       0.92      0.94      0.92     45000
```



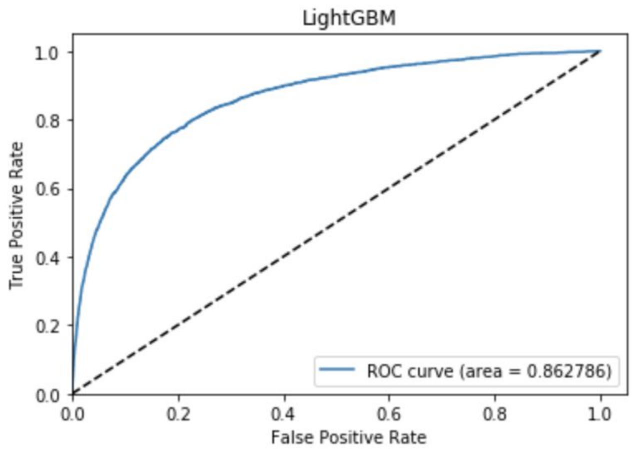
LightGBM

The LightGBM is used to train ‘eval_metric’ param for ‘auc’ and produced the AUC score of 0.862786 It took 0.62 sec to train the model.



```
In [24]: lgb_train = lgb.Dataset(X_train, y_train)
         lgb_eval = lgb.Dataset(X_test, y_test, reference=lgb_train)

         params = {
             'task': 'train',
             'boosting_type': 'gbdt',
             'objective': 'binary',
             'metric': ['binary_error', 'auc'],
         }
         num_boost_round=50
         early_stopping_rounds=20

         start_time = dt.datetime.now()
         print("Start time: ",start_time)
         lgb_gbm = lgb.train(params,
                         lgb_train,
                         num_boost_round,
                         valid_sets=lgb_eval,
                         early_stopping_rounds=early_stopping_rounds)
         print("Time took to train: ", dt.datetime.now()-start_time)

         prediction = lgb_gbm.predict(X_test).ravel()

         print ("Test Accuracy  : ", accuracy_score(y_test, prediction.round()))
         display_classification_report(y_test, prediction.round())
         plot_roc ("LightGBM", y_test, prediction)

         Start time:  2018-07-13 15:37:37.784301
         [1]     valid_0's auc: 0.847181 valid_0's binary_error: 0.0675111
         Training until validation scores don't improve for 20 rounds.
```

```
Time took to train:  0:00:00.626864
Test Accuracy  :  0.938577777778
             precision    recall  f1-score   support

          0       0.95      0.99      0.97     42029
          1       0.61      0.20      0.30      2971

avg / total       0.92      0.94      0.92     45000
```



Refinement

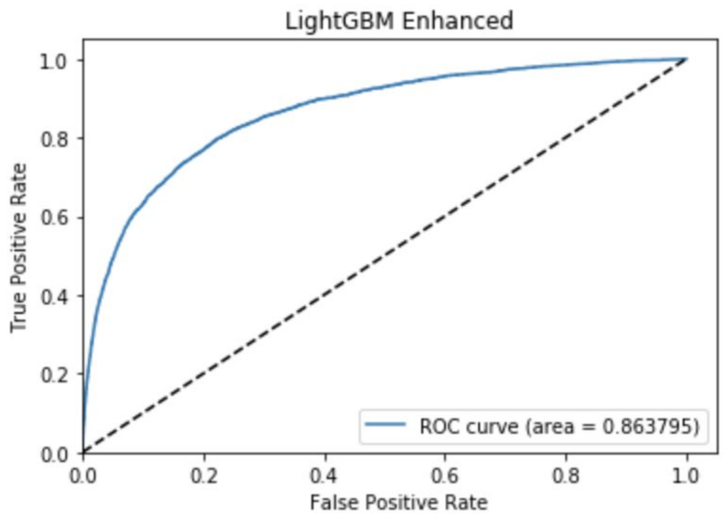
The LightGBM was the chosen model, as it AUC score was better than others. The XGBoost came close but the time LightGBM took to train the model was way faster than XGBoost. Therefore, the following parameter for the LightGBM was modified to achieve a greater AUC score of 0.863795 in 0.59 sec.

- 'num_leaves': 31,
- 'learning_rate': 0.15,
- 'feature_fraction': 0.8,
- 'lambda_l1': 5,
- 'lambda_l2': 10,
- 'min_gain_to_split': 0.5,
- 'min_child_weight': 1,
- 'min_child_samples': 5,
- 'max_depth' : 14

```
In [25]: params = {
             'task': 'train',
             'boosting_type': 'gbdt',
             'objective': 'binary',
             'metric':  'auc',
             'learning_rate': 0.15,
             'feature_fraction': 0.8,
             'lambda_l1': 5,
             'lambda_l2': 10,
             'min_gain_to_split': 0.5,
             'min_child_weight': 1,
             'min_child_samples': 5,
             'max_depth' : 14
             }
         num_boost_round=500
         early_stopping_rounds=20

         start_time = dt.datetime.now()
         print("Start time: ",start_time)
         lgb_gbm = lgb.train(params,
                         lgb_train,
                         num_boost_round,
                         valid_sets=lgb_eval,
                         early_stopping_rounds=early_stopping_rounds)
         print("Time took to train: ", dt.datetime.now()-start_time)

         prediction = lgb_gbm.predict(X_test).ravel()

         print ("Test Accuracy  : ", accuracy_score(y_test, prediction.round()))
         display_classification_report(y_test, prediction.round())
         plot_roc ("LightGBM Enhanced", y_test, prediction)
```

```
Start time:  2018-07-13 17:00:52.449774
[1]     valid_0's auc: 0.8489
Training until validation scores don't improve for 20 rounds.
[2]     valid_0's auc: 0.850706
```

```
Time took to train:  0:00:00.593726
Test Accuracy  :  0.938733333333
             precision    recall  f1-score   support

          0       0.95      0.99      0.97     42029
          1       0.61      0.20      0.30      2971

avg / total       0.92      0.94      0.92     45000
```



IV. Results

Model Evaluation and Validation

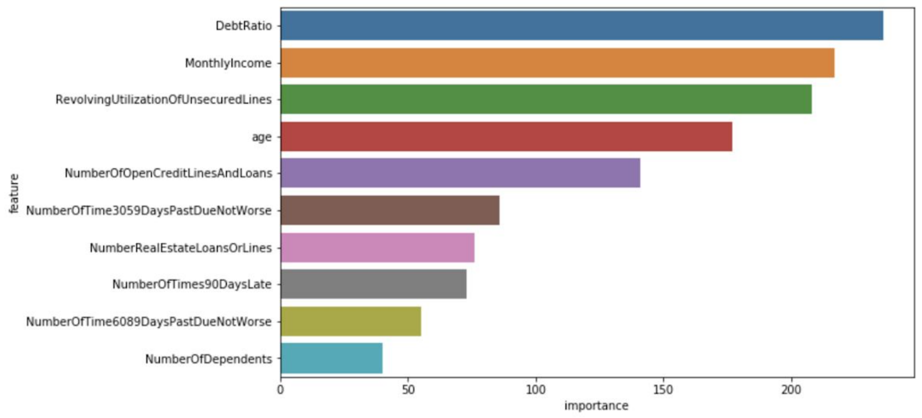
During the optimization/refinement stage of the project, the LightGBM was tested with on a cross validation set and got a AUC score of 0.863795. To verify the robustness of the model, the model was used to predict whether a person will have “Serious delinquency in 2 years” or not with an accuracy of 0.92 and recall of 0.94.

A recall score of 0.94 implies that the final model predicted 94% of the “Serious delinquency in 2 years” in the dataset correctly.

Feature Importance

```
In [26]: imp = pd.DataFrame()
         imp['feature'] = X_train.columns
         imp['importance'] = lgb_gbm.feature_importance()
         imp = imp.sort_values(by='importance', ascending=False)

         plt.figure(figsize=(10, 6))
         sns.barplot(x="importance", y="feature",
                     data=imp)
         plt.show();
```

```
/Users/rabin/anaconda3/lib/python3.6/site-packages/seaborn/categorical.py:1460: FutureWarning: remove_na is deprecate
d and is a private function. Do not use.
  stat_data = remove_na(group_data)
```



The most important 3 features are mostly a creditor financial related features such as debt ratio, monthly income, revolving utilization of unsecured lines (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). The age and Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g. credit cards) were also important. The feature “Number of dependent” that I thought would be important feature was not important at all.

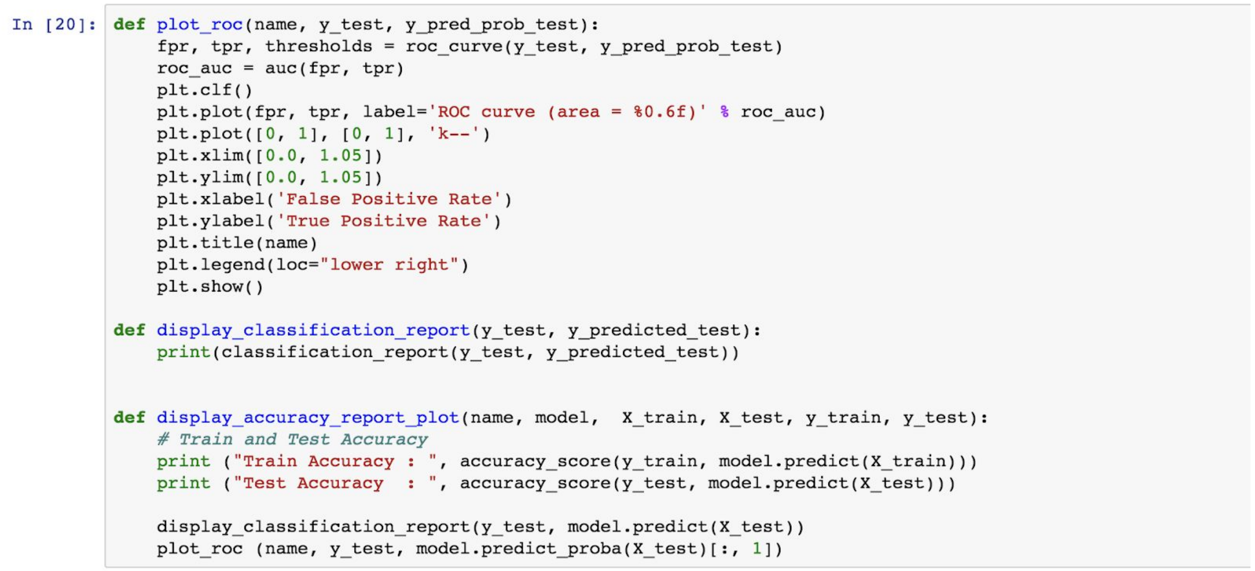
Justification

As this is a Kaggle competition a benchmark model is evaluated for area under curve [5], which is a metric for binary classifier. I ran LightGBM, the best model on the test set provided by Kaggle as test dataset. The submission file was uploaded on kaggle website and I received the score of 0.860394, which is more than I initially anticipated (my personal goal was to get 0.860000 or more). The reason for the score may be because that I did not discard any features when training my model. I will try discarding the less important features from the above section and will run the data through LightGBM and submit the result again later when I have time.

V. Conclusion

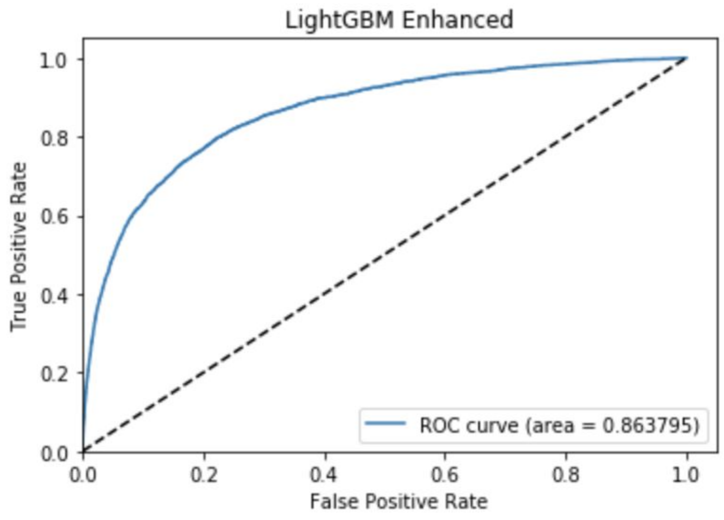
Free-Form Visualization

I visualize the ROC curve for the final model. The x-axis for the ROC curve is FP (False Positive) rate and the y-axis is TP (True positive) rate. The plot shows that the model prediction is better than random guess, which is the diagonal line running from (0, 0) to (1, 1).

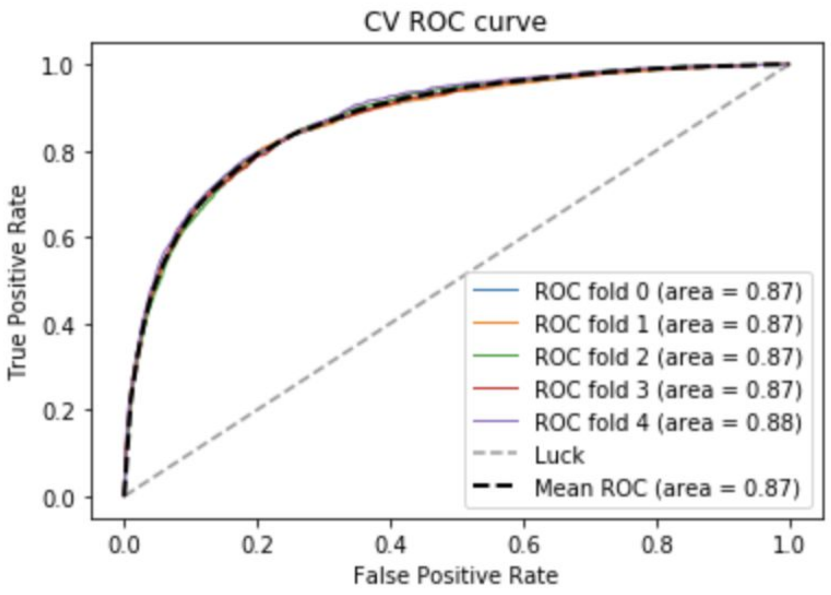


```
In [20]: def plot_roc(name, y_test, y_pred_prob_test):
             fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob_test)
             roc_auc = auc(fpr, tpr)
             plt.clf()
             plt.plot(fpr, tpr, label='ROC curve (area = %0.6f)' % roc_auc)
             plt.plot([0, 1], [0, 1], 'k--')
             plt.xlim([0.0, 1.05])
             plt.ylim([0.0, 1.05])
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title(name)
             plt.legend(loc="lower right")
             plt.show()

         def display_classification_report(y_test, y_predicted_test):
             print(classification_report(y_test, y_predicted_test))


         def display_accuracy_report_plot(name, model,  X_train, X_test, y_train, y_test):
             # Train and Test Accuracy
             print ("Train Accuracy : ", accuracy_score(y_train, model.predict(X_train)))
             print ("Test Accuracy  : ", accuracy_score(y_test, model.predict(X_test)))

             display_classification_report(y_test, model.predict(X_test))
             plot_roc (name, y_test, model.predict_proba(X_test)[:, 1])
```

```
Time took to train:  0:00:00.593726
Test Accuracy  :  0.938733333333
             precision    recall  f1-score   support

          0       0.95      0.99      0.97     42029
          1       0.61      0.20      0.30      2971

avg / total       0.92      0.94      0.92     45000
```

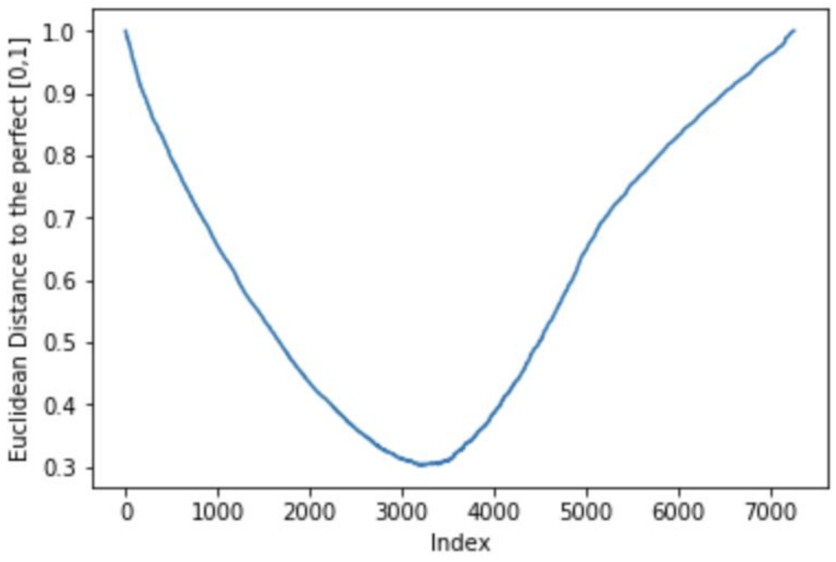


Cross validation ROC Curve will plot the CV ROC curve with nfolds



ROC Zero One will compute the best cut-off point for the classifiers

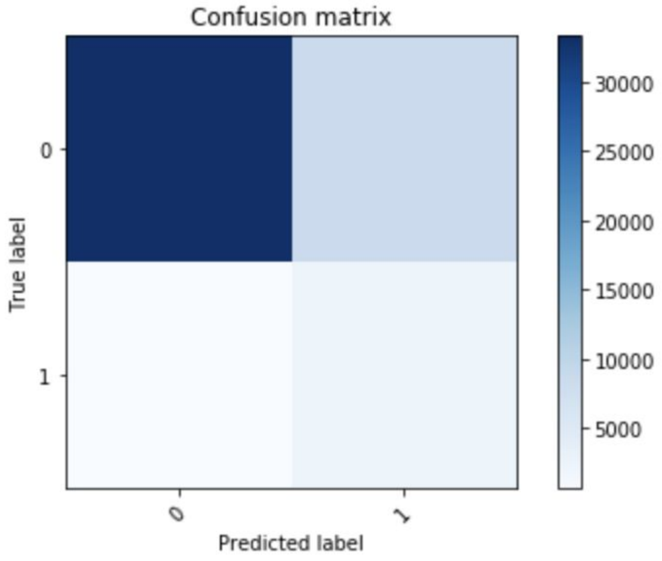
```
Best point on the ROC: TPR = 77.603%, FPR = 20.306%
Best Cut-Off point: 0.0648
```

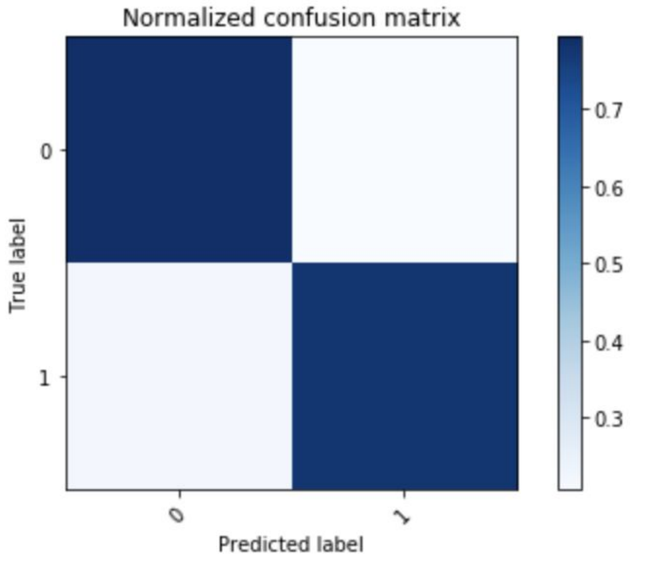


Confusion matrix

The final model predicted 33315 delinquency correctly, 8698 delinquency incorrectly, 2324 delinquency correctly and 663 not delinquency incorrectly.

```
 Confusion matrix, without normalization:

[[33315  8698]
 [  663  2324]]

 Normalized confusion matrix

[[ 0.79296884  0.20703116]
 [ 0.22196183  0.77803817]]
```





Reflection

This project can be broken down into phases based on my encounter for end-to-end problem solution;

Getting the dataset from kaggle : This was the easiest phase for me as the data prepared by Kaggle.

Exploratory data analysis and pre-processing : All the data were numeric value. There were some important principle features that were missing the value and some had outlier. The missing value and outlier needed to be replaced with a median or max value after removing outlier.

Split the data : The training data was split into training set and test set to test the model.

Model Selection and Optimization : After the five algorithm was initially tested with default parameters, choosing between XGBoost and LightGBM was bit confusing because AUC score for both algorithm was almost same. The LightGBM was however faster than XGBoost because it utilizes multi processor of computer.

The final model and solution doesn't fit my expectation for the problem, and I expect the model to perform better. When I submitted my output to Kaggle, I did not performed as I expected initially. I may have to go back and look at the testing data and clean/process it again. I believe it may still have some outliers.

Finally, embarking on this project has taught me important lessons about problems that may arise in real life. Such lessons include, being able to work with data at an abstracted level, there is no perfect algorithm we can only find a very good one that suits the problem and sometimes the simple model is the best option.

Improvement

In the global context of loan delinquency detection this model might not be a silver bullet but it is definitely a great starting point to a larger project that seeks to tackle loan delinquency globally.

Improvement on solving the problem of loan delinquency would be industry/environment specific but some general rules apply, like a proper and detailed exploratory data analysis of variables that might explain the outcome of a transaction. One possible improvement to the final auc score can be achieved by dropping some features. The date set only had eleven features.

Also more data (and features) could be collected, which would put the complexity and strength of advanced deep learning techniques to good use in building a more robust and accurate algorithm.

Further improvement to the project is to predict ROI, since ROI is what eventually matters.

References

1. Accurate Prediction of Financial Distress of Companies with Machine Learning Algorithms (https://www.researchgate.net/publication/225150865_Accurate_Prediction_of_Financial_Distress_of_Companies_with_Machine_Learning_Algorithms)
2. Give Me Some Credit (https://www.kaggle.com/c/GiveMeSomeCredit)
3. Classification Analysis(https://en.wikipedia.org/wiki/Statistical_classification)
4. ROC Curve(https://www.medcalc.org/manual/roc-curves.php)
5. Area under curve (http://fastml.com/what-you-wanted-to-know-about-auc/)
6. Binary Classification (https://en.wikipedia.org/wiki/Binary_classification)
7. XGBoost (https://en.wikipedia.org/wiki/Xgboost)
8. LightGBM (https://github.com/Microsoft/LightGBM)
9. Decision Tree (http://scikit-learn.org/stable/modules/tree.html)
10. Naives Bayes (http://scikit-learn.org/stable/modules/naive_bayes.html)
11. Logistic Regression (http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
12. Comparison Experiment (https://github.com/Microsoft/LightGBM/blob/master/docs/Experiments.rst#comparison-experiment)

13. Parallel Experiment(https://github.com/Microsoft/LightGBM/blob/master/docs/Experiments.rst#parallel-experiment)