Machine-learning algorithms to model home loans credit default risk

Anand Choubey, Sebastian Wenger, Natarajan Coimbatore, Saurabh Duggal, Muraleetharan Kurundachalam

Class project - Group 4, Advance Business Analytics, Fall - DSBA 6211

Abstract

Typically, a bank uses application data and agency data (FICO score, etc.) to identify people with home loan credit worthiness and their risk towards default (probability of default). This study tries to include additional variety of alternative external data points - like telephone and other transactional information–to predict clients' credit worthiness and repayment abilities. The introduction of alternative data would make lending fairer by allowing a bank to not rely on just the application data, and thereby increasing the loan approval odds for customers with no credit history.

However, developing sufficiently accurate and robust models would be a complex effort for these institutions on this multi-dimensional dataset. The proposed study in this report is one of such efforts to develop efficient models for the home loan default risk problem. Specially, the study would focus on developing machine-learning based non-linear, nonparametric algorithms to predict default risk for home loan accounts. We use the data sourced from "Kaggle: home loan credit default risk" to pursue this study.

The report provides a systematic model development approach - starting from eliminating insignificant variables through feature selection from the predictor pool, building a benchmark model, continually improving the benchmark model by tuning the hyper-parameters, and then evaluating model on various performance measures - to arrive at the best estimation for the model to implement on the training data.

Finally, based on research findings, we provide insight into future research in using machine-learning based modeling approaches for addressing home loan default risk problem.

Keywords: Home loan, Credit risk, Probability of default, Machine learning, Nonlinear model, Nonparametric model

Project submitted to Advance Business Analytics, Fall - DSBA 6211 December 5, 2018

1. INTRODUCTION

Financial institutions try to estimate borrowers' credit worthiness for credit products such as home loans, for these reasons - 1) to determine if the loan can be extended to them or to determine interest rates, prior to lending; 2) to estimate losses and plan for sufficient reserves to absorb those losses, post lending; and 3) to meet regulatory requirements. To meet these requirements financial institutions are increasingly relying on models and algorithms to predict losses resulting from customers' defaults [1].

Typically, a bank uses application data and agency data (FICO score, etc.) to identify people with home loan worthiness and their risk towards default. This study tries to include additional variety of alternative external data points - like telephone and other transactional information–to predict clients' credit worthiness and repayment abilities. The introduction of alternative data would make lending fairer by allowing a bank to not rely on just the application data, and thereby increasing the loan approval odds for customers with no credit history. This inclusion of additional data might help institutions avoid rejecting any potential non-default clients thus avoiding loss of business. However, This task is complex and summarizing all of these various dimensions into one score is challenging [2], but models built on statistical and machine learning techniques help achieve this goal.

There are two most common approaches to model credit risk: traditional statistical models (Ex. Logistic Regressions, Linear Regressions, etc.) and machine learning based models (Ex. Random Forest, Boosting, etc.) [3]. Typically, statistical learning methods assume formal relationships between variables in the form of mathematical equations, while machine learning methods can learn from data without requiring any rule-based programming [2]. As a result of this flexibility, machine learning methods can better fit the patterns in data, are better equipped to capture the non-linear relationships common to credit risk modeling problems. However, at the same time, these models run into a risk of over-fitting, and the predictions made by ML approaches are sometimes difficult to explain due to their complex "black box" nature [2].

In this study, we try to construct nonlinear credit risk models based on machine-learning techniques to predict default risk for home loan accounts.

The report provides a systematic model development approach to machine learning alogorithms to model a practical credit risk problem to predict customers who would potentially default - starting from a feature selection approach to eliminating insignificant features from the predictor pool, continually improving a benchmark model by tuning the hyper-parameter, and then evaluating model on various performance metrics - to arrive at the best estimation for the model to implement on the training data.

The research applies and studies industries' most popular tool for such default probability prediction problems - Logistic Regression, and also applies other newer nonparametric, machine learning algorithms - Random Forest and Light GBM - to investigate their performance and applicability to the problem.

The rest of the report is organized as follows.

Section 2 provides background and the theory on the problem, various models used in this study, and model evaluation to establish the context for the readers and foundation leading to the development of these models for the problem.

Section 3 provides information on the data and an extensive exploratory data analyses to provide insight into the data.

Section 4 provides a two stage feature engineering approach to create an optimal predictor pool to be used in the subsequent model estimation process.

Section 5 proposes and implements a systematic model estimation approach, implements machine learning models on the home loan default problem and discusses performance results.

Section 6 summarizes the study, and proposes potential future study to address home loans credit default risk problem.

2. BACKGROUND

Home loan default occurs when a borrower is unable to meet the legal obligation of repayment. A bank may flag an account to have defaulted or charged-off if the account is delinquent for a period of time, or is flagged as deceased or bankrupt. A default results in a loss to the bank. Therefore, a bank wants to forecast if an account is going to default or not so that the bank can calculate projected losses to determine reserve amount and for various regulatory reports and business needs. As the bank is trying to determine for each account if it would default or not default, the problem falls in the category of binary classification problems.

To address the problem of modeling default, We plan to develop a logistic regression model and machine learning (ML) based algorithms, such as random forest and light GBM, for the credit default problem and study performances of these models on the given dataset.

2.1. ***Problem background and solution approach***

Our home loan default risk problem to predict if a customer is likely to default, given his/ her characteristics, falls in the category of supervised learning since the loans are labeled (in this case, default = 1 or 0).

In this report, we have implemented Logistic Regression, Random Forest and Light GBM models to the home loan default prediction problem. The following sections provide the theoretical background on these techniques.

2.2. ***Logistic regression***

Logistic regression is a widely implemented modeling approach for classification problem [4]. The independent variables vector (X) is linked to probability of outcomes (binary or multinomial) modeled by the dependent variable (y) by a logit function. The response probability, $p = Pr(Y = 1|x)$, is modeled with the logistic regression of the form [5]:

$$logit(p) = log(p/1 - p) = \alpha + \beta^{'} x \tag{1}$$

Where α is the intercept parameter and βs are slope parameters for independent variable vector, X. Once the $logit(p)$ is estimated, the response probability, p, probability that $Y = 1$, can be estimated by the following equation:

$$p = e^{\alpha+\beta^{'} x}/1 + e^{\alpha+\beta^{'} x} \tag{2}$$

The function above to calculate the value for p given x is a S-shaped, sigmoid function. This function ensures that output values for p will fall between 0 and 1, for any value of x. Figure 1 below show logistic and linear models fitted to a toy dataset. We can notice that a logistic function is better suited to model a binary response.

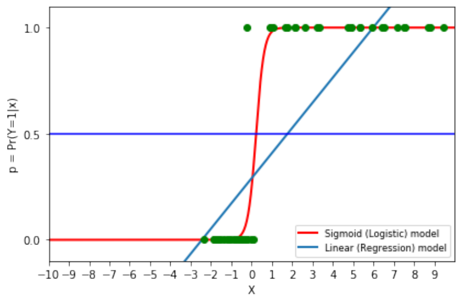


Figure 1: Logistic sigmoid model vs Linear regression model

Once we estimate the probability of default, we transform it to 1 or 0 using a limit to probability. If the default probability, $p > 0.5$ we would consider it as default $= 1$, and 0 otherwise. These predicted values for the target variable are then compared to a test data set to compute the performance of the model.

2.3. ***Random Forest***

Parametric models, such as logistic regression, make certain assumptions on the data that may not be entirely correct. That is why we wanted to implement other models to our dataset that are nonparametric, that is the model does not make any assumptions as to the form or parameters of a distribution fit to the data, to study performance.

The random forest is one of such nonparametric models. Random forest is an ensemble learning method for classification and regression that operates by constructing a lot of decision trees based on training data set and outputting the class that is the mode of the classes output by individual trees. The decision tree model predicts the value of a target variable by learning simple decision rules inferred from the training data. It creates a set of rules used to classify data into partitions. It evaluates variables in a data to determine which are the most important to partition the data, and then comes up with a tree of decisions (set of rules) which best partitions the data. In other words, the model tends to learn the training data instead of learning the patterns. Figure 2 shows a simple decision tree.

We know that a decision tree ealsy runs into "overfitting" situation. Hence, we typically use an ensemble random forest algorithm to address this issue.

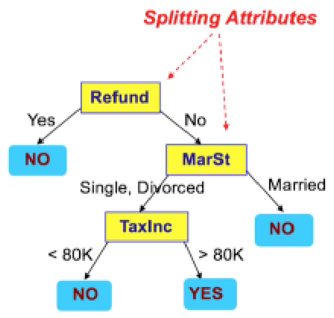


Figure 2: A simple Decision Tree(**Source:** DSBA 6211 class notes on Random Forest)

In the Random Forest process, we randomly select 'm' number of trees and randomly select 'n' number of variables from the given variables set for each tree, hence the word "random". These trees are then trained on different parts of the same training data. This approach ensures overcoming of over-fitting problem of an individual decision tree.

Figure 3 shows how ensemble approach of Random Forest for classification works. For the problem in our study, each tree gives a prediction or "votes" for a response 'default = 1' for instance; after the voting by all the m trees in the forest, we count the 'default = 1' votes. The percent 'default = 1' votes received is the predicted probability.

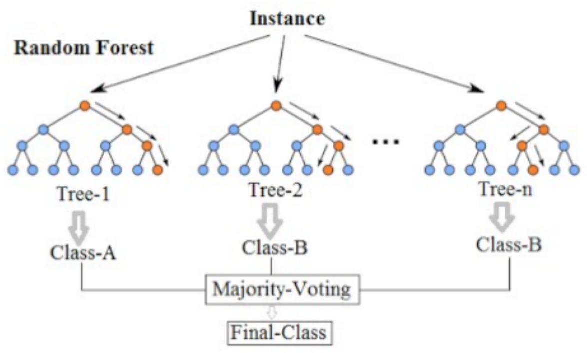


Figure 3: A simple Random Forest framework to model binary response
(**Source:** DSBA 6211 class notes on Random Forest)

Random forest implies different decision trees on randomly selected subsets of the data. These trees have small bias and high variance; thus, taking the aggregated prediction of all these models lowers the variance while keeping a small bias.

2.4. Light GBM

Light GBM is a gradient boosting framework that uses tree based learning algorithms [6]. Light GBM has demostrated several advantages, such as faster training speed, low memory usage, handling of large-scale data, over other algoirthms [7]. These advantages with this algorithm makes it particulary suitable for our problem.

Most decision tree learning algorithms grow trees by level (depth)-wise. Light GBM grows trees leaf-wise (best-first)[8] as shown in Figure 4. It will choose the leaf with max delta loss to grow. Leaf-wise algorithms tend to achieve lower loss than level-wise algorithms [8].

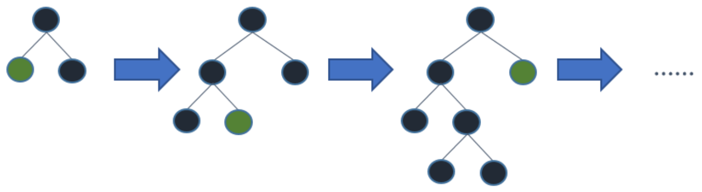


Figure 4: Leaf-wise tree growth in Light GBM
(**Source:** https://lightgbm.readthedocs.io/en/latest/Features.htmlreferences)

2.5. Model Evaluation

Model evaluation is a very important aspect of the estimation process. We need to evaluate the quality of predictions generated by the model at each step of model development. We also need to continually evaluate the model for robustness and applicability to the data as part of ongoing monitoring review process. There are two aspects of model evaluation: 1) the data we intend to use for evaluating the model, and 2) the appropriate metrics to measure the model's performance.

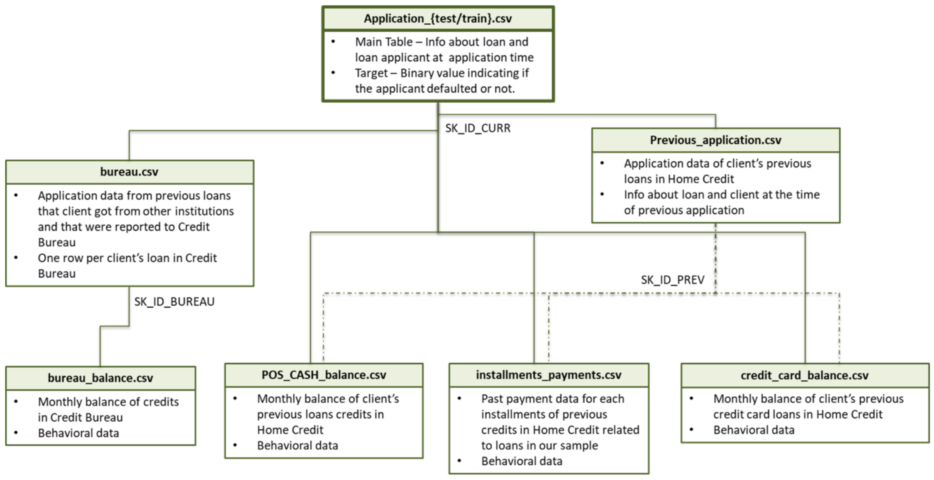
There are various ways to use data to evaluate a model. One approach, known as the "hold-out" method, is to randomly partition the given data into a training set and a testing set. Usually, we use about 60-70% of the data for training the model and rest is set aside for testing the model predictions.

We use few relevant metrics to measure model's prediction performance in this study, such as F1-score, ROC and AUC score.

3. DATA

This section is dedicated to provide a description on the data being used in the model and provide some insight into the data through exploratory data analysis. We use the data sourced from "Kaggle: home loan credit default risk" to pursue this study.

Figure 5 shows all the data components.



Application_{test/train}.csv
Main Table – Info about loan and loan applicant at application time
Target – Binary value indicating if the applicant defaulted or not.
SK_ID_CURR
Previous_application.csv
Application data of client's previous loans in Home Credit
Info about loan and client at the time of previous application
bureau.csv
Application data from previous loans that client got from other institutions and that were reported to Credit Bureau
One row per client's loan in Credit Bureau
SK_ID_BUREAU
SK_ID_PREV
bureau_balance.csv
Monthly balance of credits in Credit Bureau
Behavioral data
POS_CASH_balance.csv
Monthly balance of client's previous loans credits in Home Credit
Behavioral data
installments_payments.csv
Past payment data for each installments of previous credits in Home Credit related to loans in our sample
Behavioral data
credit_card_balance.csv
Monthly balance of client's previous credit card loans in Home Credit
Behavioral data

Figure 5: Data model
(**Source:** https://www.kaggle.com/c/home-credit-default-risk/data)

3.1. Model data

As stated earlier, this study includes data from multiple-sources to be used in the model development process. Each component of the data used in the model is explained below.

Application Data

Application data is the main dataset with each record representing a loan. It carries information about the client captured at the time of the application. It also contains the loan details for which the client was approved. Data elements include income, education, gender, employment status, family status, etc., Since this is not a traditional credit application that we have in USA the client information also includes Telecom, Residential details and Socio-economic data. Residential data is captured extensively to get a better understanding of their current living condition to calculate the expense and affordability. This is very critical to our solution, as most of the population does not have a credit history or banking footprint. Primary key in this dataset is the ID of the loan (SK ID CURR), which can be used to join to additional datasets like Bureau, Bureau Balance, POS Cash Balance, etc.,

Bureau

Bureau dataset carries the information about the client's previous credits provided by other lending institutions. It contains the annuity, credit limit, credit past due and also the days the clients are behind on the payment. In an ideal case the bureau data will have as many rows as the number of loans/credits the client had prior to the day the application was submitted. Current loan ID (SK ID CURR) can be used to join to the application dataset. Primary key on this dataset is (SK BUREAU ID), which can be used to join to Bureau balance dataset.

Bureau Balance

Bureau balance carries the balance details of the client's previous loans captured by the bureau. It is captured every month, with each row providing a snap shot of the loan's balance information as reported to the credit bureau. It also provides the status of the loan as of that month. Status includes if the loan is active, closed, unknown, etc., Primary key on this data set is (SK BUREAU ID), which can be used to join to Bureau dataset.

POS Cash Balance

POS Cash balance dataset provides the monthly balance snapshot of previous POS and Cash loans that the applicant has with Home Credit. The captured data elements include monthly balance, number of terms remaining, term details, etc., It contains one row for each month the history is captured for previous credit in Home Credit. Primary key is the (SK ID PREV). However, (SK ID CURR) can be used to join to the main application dataset.

Credit Card Balance

Credit card balance dataset carries the monthly balance snapshots of previous credit cards that the applicant has with Home Credit. The loan in this dataset includes consumer credit as well as the cash loans. For each month of every previous credit in Home Credit related to loans is captured as a row in this dataset. Data include the balance, interest, cash withdrawal information at ATM, POS, etc., Current (SK ID CURR) and previous loan ID (SK ID PREV) are present in this dataset so it can be joined to both main/current and previous applications datasets.

Previous Application

Previous application dataset carries information about the client captured at the time of the previous application. It also contains the loan details for which the client was approved. There is some key metrics like Days First Due, Days Last Due, Days First Drawing, etc., derived with relevant to the date of current application. Primary key in this dataset is the ID of the previous loan (SK ID PEV). It also carries the current loan ID (SK ID CURR), which can be used to join to the current application dataset.

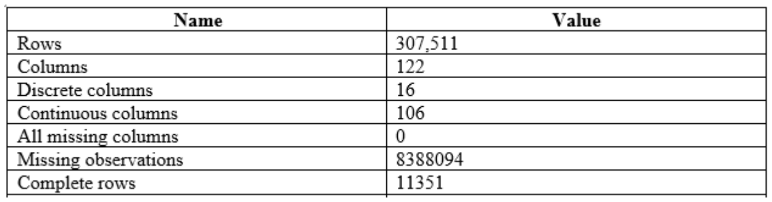
Installment Payments

Installment payments dataset carries the repayment history for the previously disbursed credits in Home Credit related to the loans in the data. There is one row for every payment that was made plus one row each for missed payment. Every payment towards one installment corresponding to one previous Home Credit loan is captured in a row. There is some key metrics like Days Installment and Days Entry Payment derived with relevant to the date of current application. Primary key in this dataset is the ID of the previous loan (SK ID PEV). It also carries the current loan ID (SK ID CURR), which can be used to join to the current application dataset.

3.2. ***Exploratory data analysis***

Basic statistics

Table 1 provides basic statistics on the data.



Name	Value
Rows	307,511
Columns	122
Discrete columns	16
Continuous columns	106
All missing columns	0
Missing observations	8388094
Complete rows	11351

Table 1: Basic Statistics

Correlation analysis

We now study correlation between the variables. Figure 6 below provides correlation within entire application data.

Based on the above correlation matrix, spooled few variables that showed higher correlation to get a clearer matrix as below:

Correlation Meter indicates the intensity of how the variables are correlated. Higher the number (indicated in brighter shades of red), higher the correlation between those variables.

Correlation indicates the relationship between 2 variables. It also indicates that as one variable changes in value, the other variable tends to change in a specific direction. Understanding such relationships is critical because we can use value of one variable to predict the value of the other variable. For example, in our data, we can use AMT-ANNUITY or AMT-GOODS-PRICE to predict the value of AMT-CREDIT. We can also get the

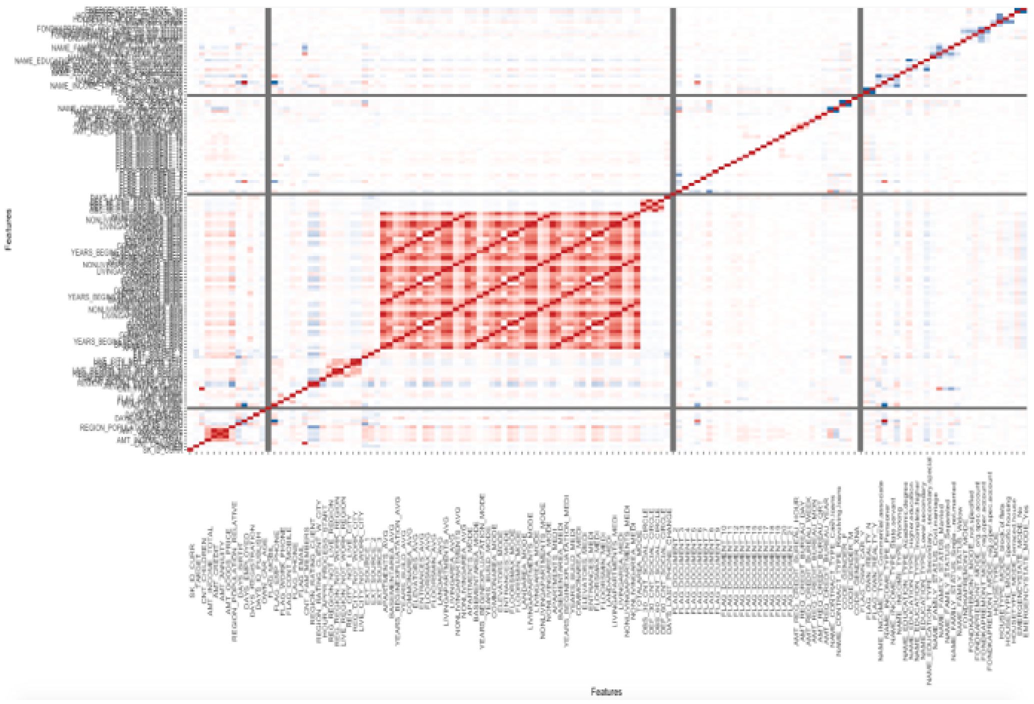


Figure 6: Correlation with entire application data set

individual correlation coefficients to get into depth of the variables linear or non-linear relationships.

$> cor(p1 - data\$TARGET, p1 - data\$AMT - CREDIT)$

$[1] - 0.03036929$

Which indicates that as the Credit amount of the loan increases, the applicant is more likely to be default.

Based on the correlation analysis presented in Figure 7, we infer that high correlation exists between:

- AMT-CREDIT with AMT-ANNUITY, AMT-GOODS-PRICE
- APARTMENTS-AVG with BASEMENTAREA-AVG, LANDAREA-AVG, LIVINGAPARTMENTS-AVG, APARTMENTS-MODE, BASEMENTAREA-MODE, YEARS-BUILD-MODE, COMMONAREA-MODE, FLOORSMAX-MODE, FLOORSMIN-MODE,
- LIVINGAPARTMENTS-MODE, LIVINGAREA-MODE

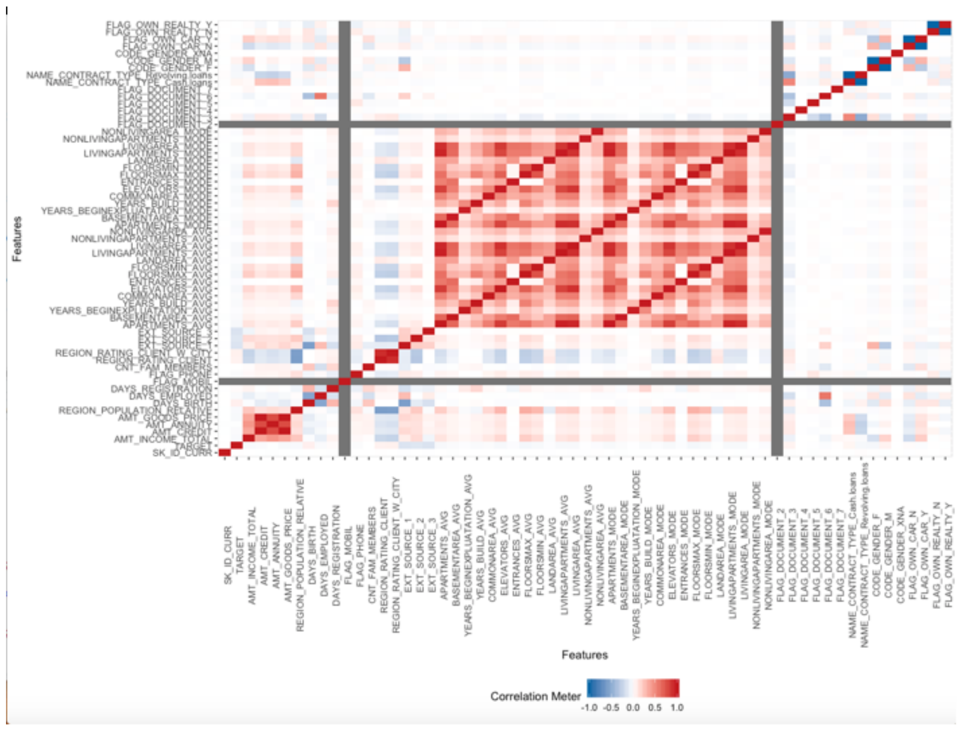


Figure 7: Correlation with selected fewer highly correlated variables

And, no correlation exists between:

- CONTRACT-TYPES
- GENDER
- OWNED CAR
- OWNED HOUSE or FLAT

Univariate distribution of categorical variables

Univariate analysis is the simplest form of analyzing data. This analysis does not deal with causes or relationships and its major purpose is to describe the data, summarize and find patterns in the data. We used bar plots to visually compare the data among different categories. The longer the bar, the greater its value in the dataset. The bar diagram made it easy to compare sets of data between different groups at a glance. The graph represents Frequency on one axis and the discrete variable in the other. The goal is to show the relationship between two axes.

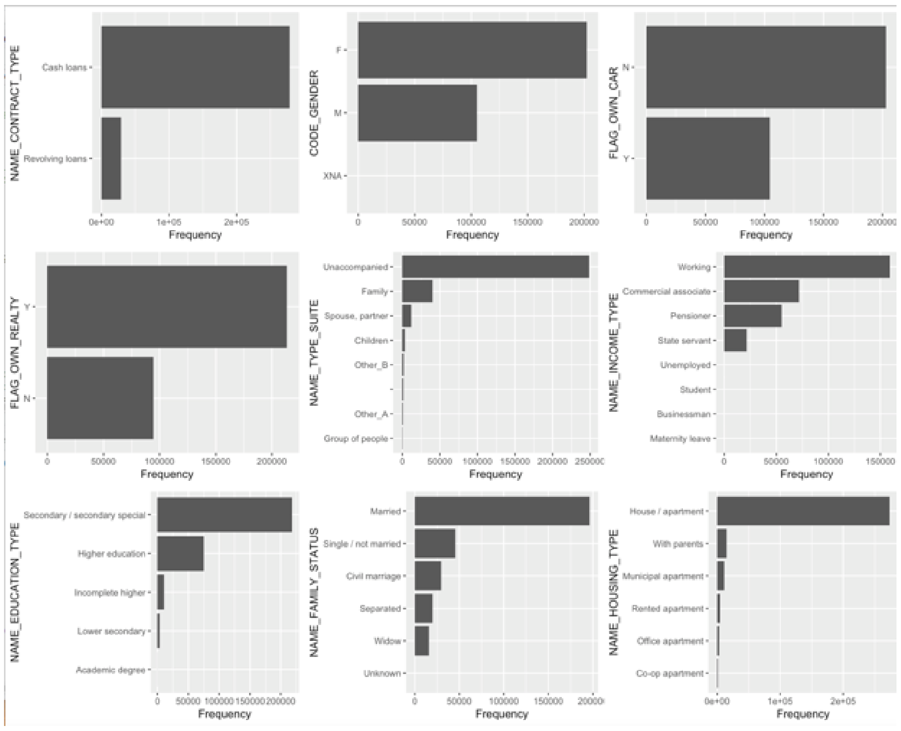


Figure 8: Univariate analysis

Applicants

- Majorly lived in House/Apartment
- Majorly from Secondary education type and higher educated
- Majorly owned a house or a flat
- Majorly applied cash loans than revolving loans
- Majorly are Females
- Majorly didn't own a Car
- Majorly unaccompanied, working, secondary level educated
- Majorly with family status of 'Married'

Applicants

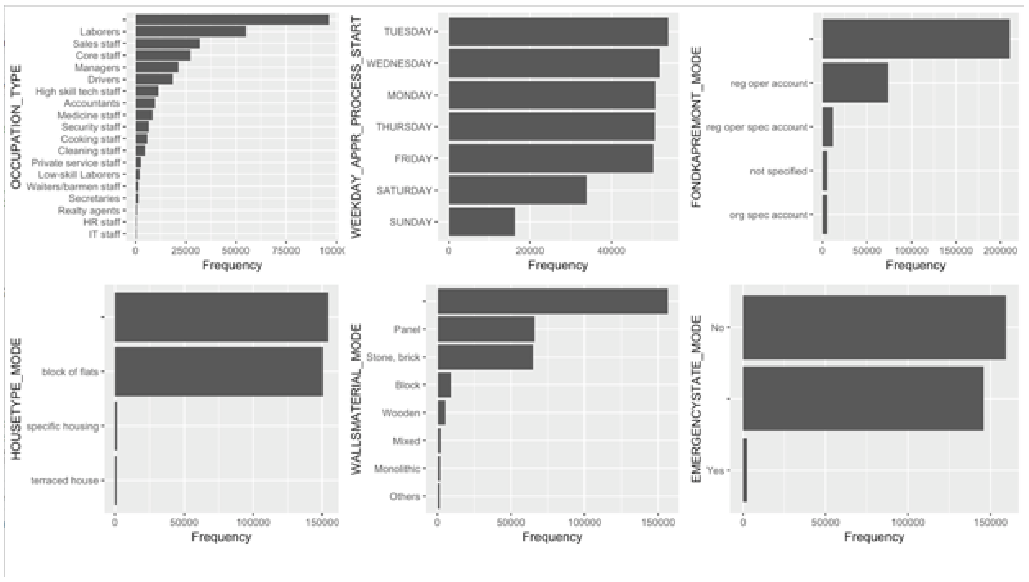


Figure 9: Univariate analysis

- Majorly are laborers and least from IT
- Majorly applied on Tuesday's and Wednesday's
- Majorly had regular Operating account

Ignored

- Organization-type had more than 58 categories.

Univariate distribution of continuous variables

Used Histogram to display the numeric data in bins. The area of each bin represents the frequency of data. The histogram of all the quantitative variables in the dataset is as displayed below.

If the left side of a histogram resembles a mirror image of the right side, then the data is symmetric which also indicates a mean or average is a good approximation for the center of the data. Statistical tools such as t-tests can be used to analyze such data. If the data is not symmetric, then they are either left-skewed or right-skewed. If the data are skewed, then the mean may not provide a good estimate for the center of the data and represent where most of the data fall. In such cases, we should consider using the median to evaluate the center of the data rather than the mean. Histogram also indicates outliers and data imbalances based on the distribution.

In our dataset, following variables are symmetrically and center values of those can be calculated using mean or t-test for using in the model. DAYS-BIRTH, DAYS-ID-PUBLISH, HOUR-APPR-PROCESS-START, EXT-SOURCE-1, EXT-SOURCE-3

Rest of the data fields that are skewed either left or right are:

AMT-CREDIT, AMT-ANNUITY, AMT-GOODS-PRICE, REGION-POPULATION-RELATIVE, DAYS-REGISTRATION, OWN-CAR-AGE, CNT-FAM-MEMBERS, EXT-SOURCE-2, APARTMENTS-AVG, BASEMENTAREA-AVG, YEARS-BEGINEXPLUATATION-AVG, YEARS-BUILD-AVG, COMMONAREA-AVG, ELEVATORS-AVG, ENTRANCES-AVG, FLOORSMAX-AVG, FLOORSMIN-AVG, LANDAREA-AVG, LIVINGAPARTMENTS-AVG, LIVINGAREA-AVG, NONLIVINGAPARTMENTS-AVG, NONLIVINGAREA-AVG, APARTMENTS-MODE, BASEMENTAREA-MODE

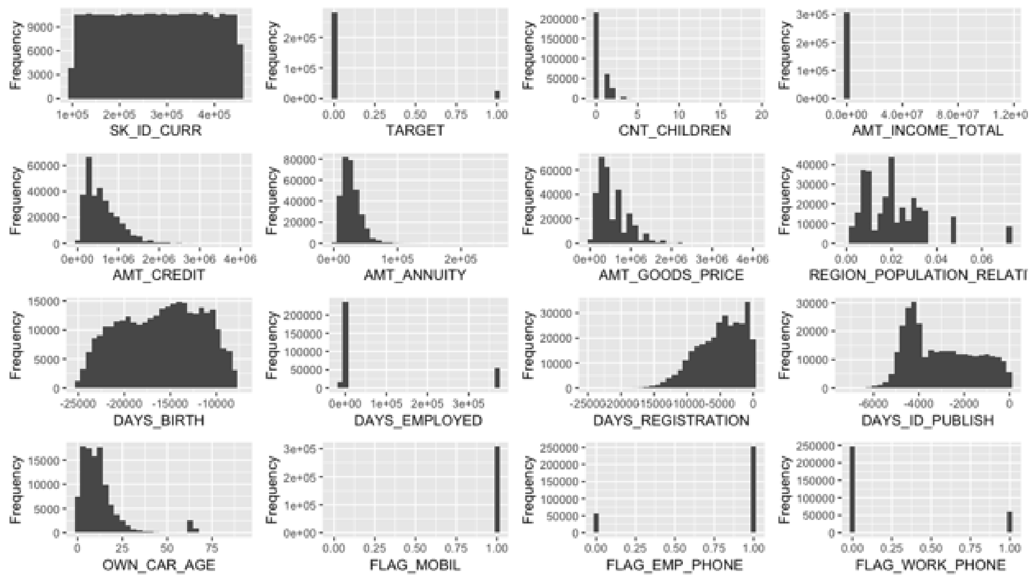


Figure 10: Univariate analysis

For the sake of brevity we are not showing plots for all the univariate analysis we data as part of this study.

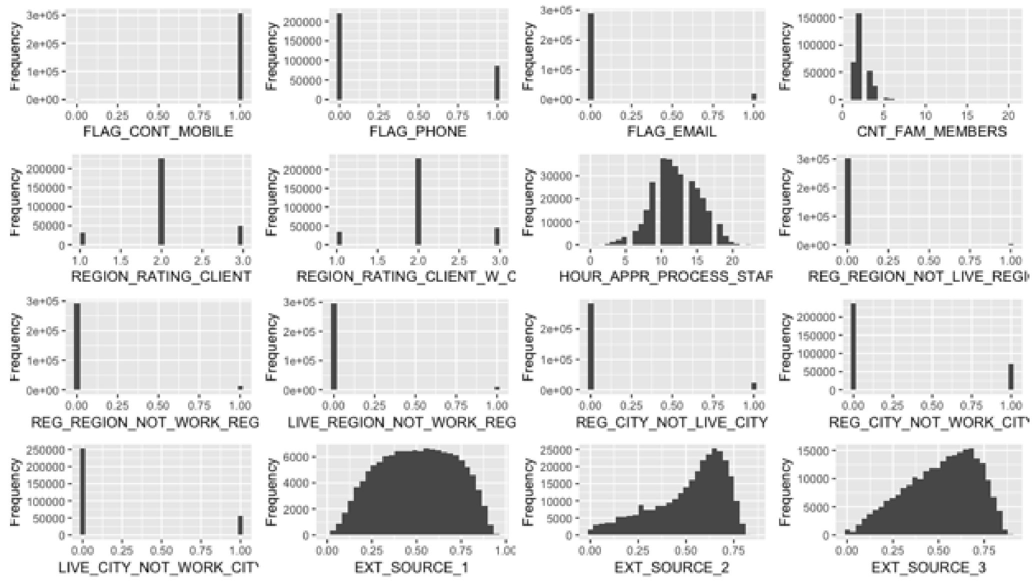


Figure 11: Univariate analysis

4. FEATURE ENGINEERING

In this study, we performed feature engineering in two stages - feature generation and feature reduction.

Feature generation

In this stage we regenarate variables based on two-way interactions amoung all variables. We used ”Scikit-learn” library’s (featuretools package) ”generate interactions” function to create all pairwise combinations of *,/,-,+ for all integer and float datatype columns. We followed the following steps to generate features.

1. Clean Data (replace bad data with ”NA”)
2. Generate all 2 way interactions (-,+,*,/) within each sub table’s numeric columns. This way, the aggregations can be applied on the interactions, which can be hard for the model to detect without doing so.
   On main table (application data), group by current loan ID on each subtable’s numeric columns calculating mean,median,max,min,standard deviation,skew,count.
3. On main table group by current loan ID on each subtable’s categorical columns, creating dummy variables for each category, representing the proportion of representation.
   This was much more effective versus aggregating by mode and number of unique categories.
4. Generate all 2-way (-,+,/,*) interactions within the main table’s numeric columns.
   Carefully examine these, and add in a few of the most important ones manually (totaled to 7). Most notable was amt-annuity/amt-credit (monthly payment/total loan amount).

After all the features are generated in the subsequent stage we try to reduce the features based on 1) correlation among the variables and 2) based on their significance in model estimation.

Feature reduction

We perform the feature reduction process in these two steps.

1. Drop enough columns on the combined main table so that no 2 columns have greater than .95 correlation (prioritize given features over created, and less NA over more NA). We were able to remove 2000 features in this step.

2. Run a **lgbmclassifier** model with default parameters 3 times and drop all features which consistently score at 0. We were able to remove 1000 features in this step.

Figure 12 shows the final features we selected for building model after going through the feature selections steps mentioned above.

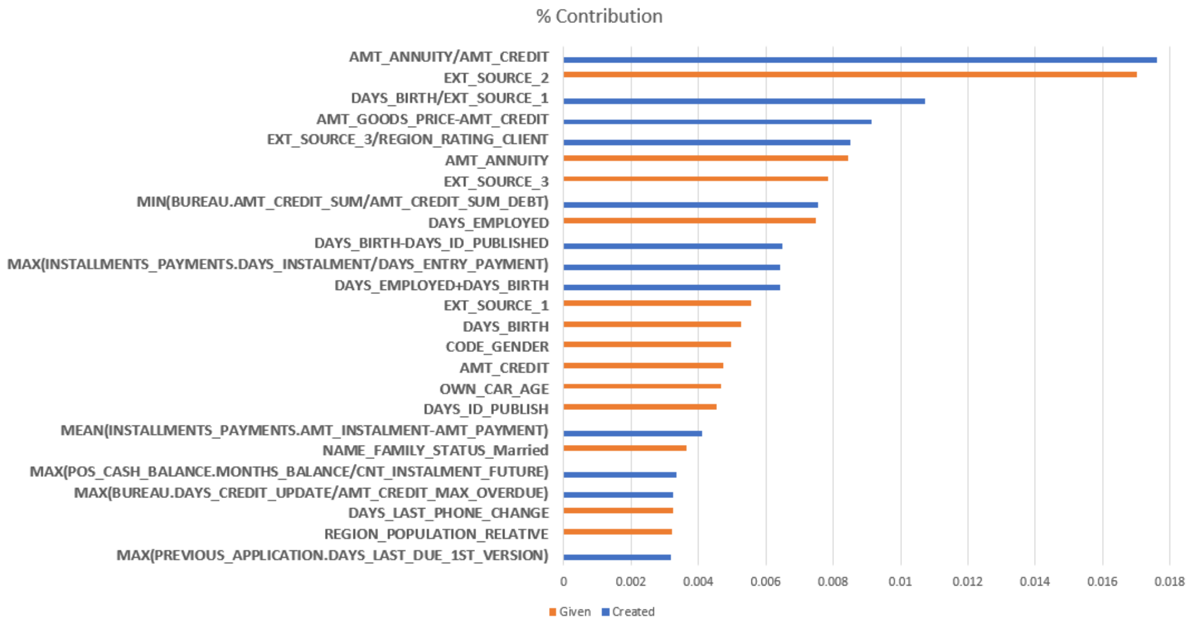


Figure 12: Feature importance

Figure 13 shows the cummulative importance of features to show the deminising returns in addiding more and more features in the predictor pool.

We use the features pool created after the feature engineering process to build our model. We would like to note here that we would further pursue dropping additional features in the estimation process of a specific model.

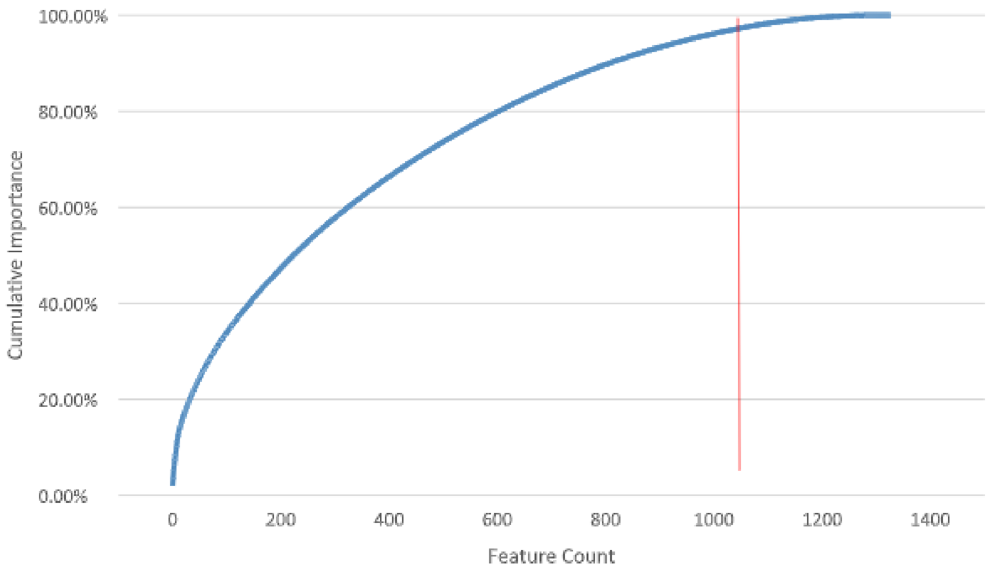


Figure 13: Cummulative feature importance

5. MODEL ESTIMATION AND RESULTS

This section is dedicated for describing the model estimation process, and analyzing prediction results. We develop and implement Logistic Regression, Random Forest and Light GBM on the balanced training data.

In this study we follow a systematic approach to model development that in general includes the following steps:

- Start with a benchmark model to the training set and review the results. A benchmark result provides a reference to any additional steps we take to improve the model performance.
- Apply a hyper-parameter optimization approach. Tune hyper-parameters either manually or through GridSearch, through Bayesian optimization to improve performance.
- Drop the least important features representing 5% of the feature imporantance. 400 removed.
- Run the hyper-parameter optimized models on the final training data
- Review model results using performance metrics such as F1-score, ROC Curve, AUC.

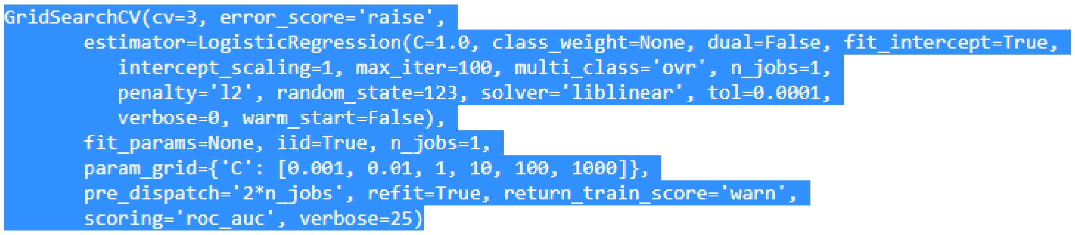
In this report we use machine learning libraries such as scikit-learn, and other open-source machine learning libraries, to build the models. We implemented these classification algorithms from these libraries by developing codes in Python and R programming languages. The code to implement EDA, feature selection, and model development is provided in Appendix section.

5.1. ***Logistic regression***

We followed these specific steps to estimate logistic regression model:

- We start with replacing any "NA" values (inf,-inf,blanks) with the columns mean. This is done because logistic regression cannot handle "NA" values.
- Randomly create train and test datasets.
- Run gridsearch to optimize C hyperparameter for AUC, with 3 fold cross validation.

  Figure 14 shows the best hyper-parameter for the logistic regression model.

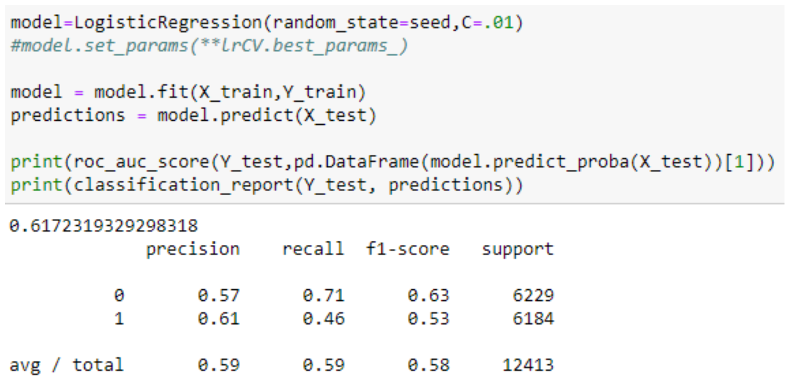


```
GridSearchCV(cv=3, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=123, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': [0.001, 0.01, 1, 10, 100, 1000]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring='roc_auc', verbose=25)
```

Figure 14: Best hyper-parameter for the logistic regression

- Finally, we reun the model on the final data.

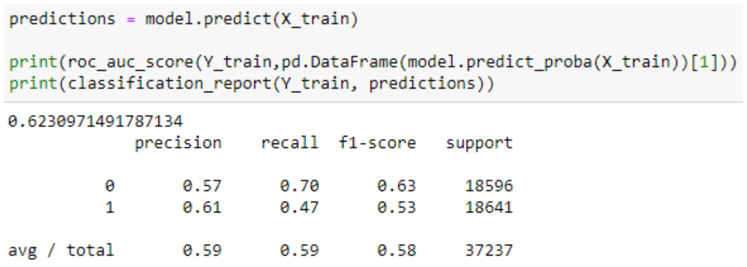
  Figure 15 and 16 show the model peformance on the testing data and the training data.



```
model=LogisticRegression(random_state=seed,C=.01)
#model.set_params(**lrCV.best_params_)

model = model.fit(X_train,Y_train)
predictions = model.predict(X_test)

print(roc_auc_score(Y_test,pd.DataFrame(model.predict_proba(X_test))[1]))
print(classification_report(Y_test, predictions))
```

```
0.6172319329298318
             precision    recall  f1-score   support

          0       0.57      0.71      0.63      6229
          1       0.61      0.46      0.53      6184

avg / total       0.59      0.59      0.58     12413
```

Figure 15: Logistic regression performance on Test data



```
predictions = model.predict(X_train)

print(roc_auc_score(Y_train,pd.DataFrame(model.predict_proba(X_train))[1]))
print(classification_report(Y_train, predictions))
```

```
0.6230971491787134
             precision    recall  f1-score   support

          0       0.57      0.70      0.63     18596
          1       0.61      0.47      0.53     18641

avg / total       0.59      0.59      0.58     37237
```

Figure 16: Logistic regression performance on Train data

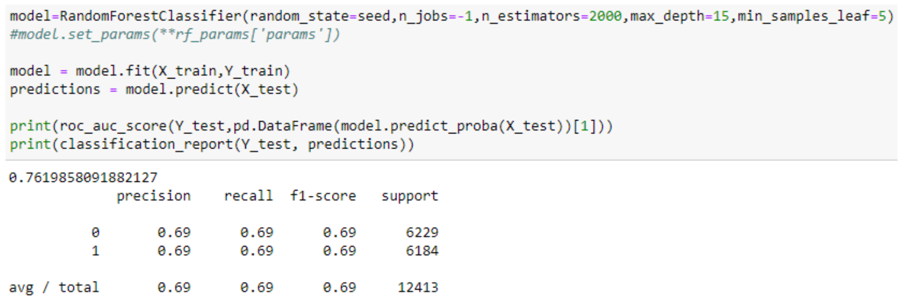
We noticed that both the hyper-parameter tuning and the optimal variable selection approaches resulted in improvement in the logistic regression model performance. However, the logistic regression performance on test data for AUC is only about 0.62.

5.2. ***Random forest***

We followed these specific steps to estimate random forest model:

- We start with replacing any "NA" values (inf,-inf,blanks) with the columns mean. This is done because radom forest cannot handle "NA" values.
- Randomly create train and test datasets.
- Run 30 iterations of bayesian optimization to find hyperparameters (less iterations for this model as there is a smaller solution space vs. light gbm).
- Then we run the model.

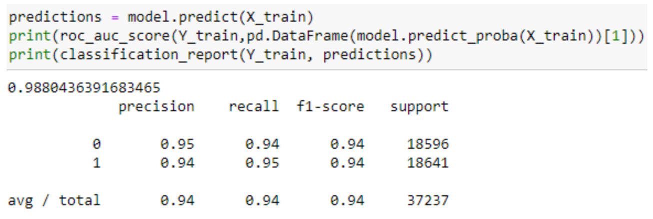
Figure 17 and 18 show the performance results on test and train data repectively.



```
model=RandomForestClassifier(random_state=seed,n_jobs=-1,n_estimators=2000,max_depth=15,min_samples_leaf=5)
#model.set_params(**rf_params['params'])

model = model.fit(X_train,Y_train)
predictions = model.predict(X_test)

print(roc_auc_score(Y_test,pd.DataFrame(model.predict_proba(X_test))[1]))
print(classification_report(Y_test, predictions))
```

```
0.7619858091882127
             precision    recall  f1-score   support

          0       0.69      0.69      0.69      6229
          1       0.69      0.69      0.69      6184

avg / total       0.69      0.69      0.69     12413
```

Figure 17: Random forest performance on Test data



```
predictions = model.predict(X_train)
print(roc_auc_score(Y_train,pd.DataFrame(model.predict_proba(X_train))[1]))
print(classification_report(Y_train, predictions))
```

```
0.9880436391683465
             precision    recall  f1-score   support

          0       0.95      0.94      0.94     18596
          1       0.94      0.95      0.94     18641

avg / total       0.94      0.94      0.94     37237
```

Figure 18: Random forest performance on Train data

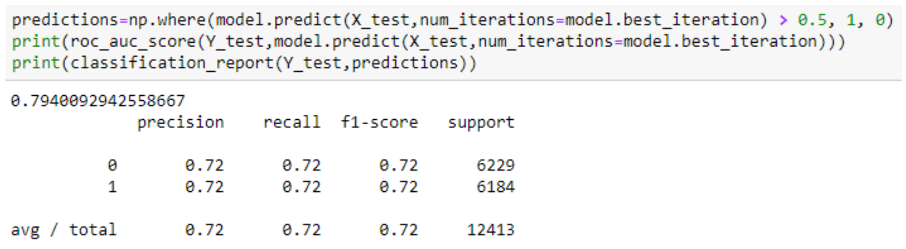
With random forest we see AUC score go up to 0.76, which is a significant improvement over our parametric logistic regression model.

5.3. ***Light GBM***

We followed these specific steps to estimate Light GBM model:

- Randomly create train and test datasets.
- Run 120 iterations of bayesian optimization to find hyperparameters.
- Perform 5 fold cross validation on train data.
- Finally, train the model to minimize binary log loss metric.

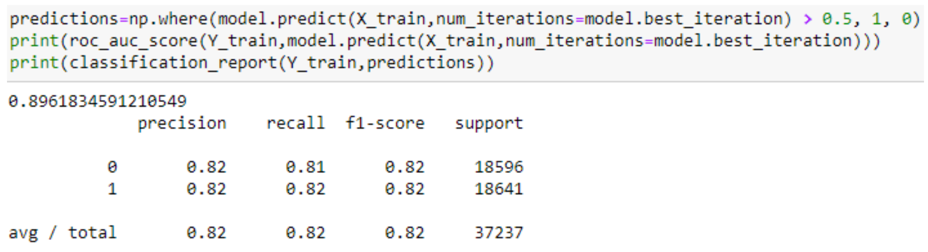
Figure 19 and 20 show the performance results on test and train data respectively.



```
predictions=np.where(model.predict(X_test,num_iterations=model.best_iteration) > 0.5, 1, 0)
print(roc_auc_score(Y_test,model.predict(X_test,num_iterations=model.best_iteration)))
print(classification_report(Y_test,predictions))
```

```
0.7940092942558667
             precision    recall  f1-score   support

          0       0.72      0.72      0.72      6229
          1       0.72      0.72      0.72      6184

avg / total       0.72      0.72      0.72     12413
```

Figure 19: Light GBM performance on Test data



```
predictions=np.where(model.predict(X_train,num_iterations=model.best_iteration) > 0.5, 1, 0)
print(roc_auc_score(Y_train,model.predict(X_train,num_iterations=model.best_iteration)))
print(classification_report(Y_train,predictions))
```

```
0.8961834591210549
             precision    recall  f1-score   support

          0       0.82      0.81      0.82     18596
          1       0.82      0.82      0.82     18641

avg / total       0.82      0.82      0.82     37237
```

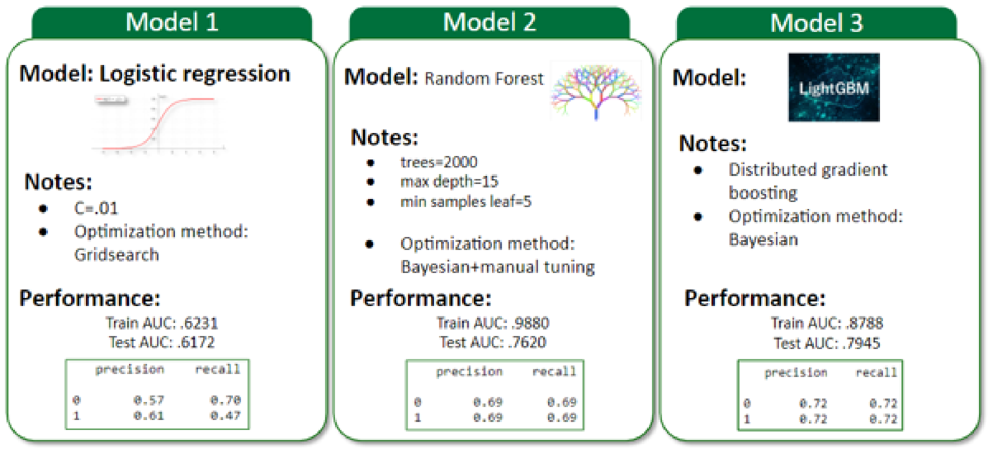
Figure 20: Light GBM performance on Train data

With light GBM we see AUC score go up to about 0.794 (on test data), which is an improvement over the random forest model.

5.4. *Models' performance review*

Figure 21 and Figure 22 show the results from different models. Light GBM seems to providing overall the best results on this data. Random forest results are very close; but we observed very stable results with Light GBM. Also, there were "NA" present in the data and to implement light GBM we did not have to do any specific treatments for "NA".

The results seem to be reasonable, we are able to achieve almost about 80% AUC score with Light GBM model. However, some aspects of predictions can to be improved to make the model more useful for a practical purpose.



Model 1

Model: Logistic regression

Notes:
- C=.01
- Optimization method: Gridsearch

Performance:
Train AUC: .6231
Test AUC: .6172

```
   precision   recall
0       0.57     0.70
1       0.61     0.47
```

Model 2

Model: Random Forest

Notes:
- trees=2000
- max depth=15
- min samples leaf=5
- Optimization method: Bayesian+manual tuning

Performance:
Train AUC: .9880
Test AUC: .7620

```
   precision   recall
0       0.69     0.69
1       0.69     0.69
```

Model 3

Model: LightGBM

Notes:
- Distributed gradient boosting
- Optimization method: Bayesian

Performance:
Train AUC: .8788
Test AUC: .7945

```
   precision   recall
0       0.72     0.72
1       0.72     0.72
```

Figure 21: Models' results

On the technical point of view, we provided a very systematic process to developing a model, starting from a basic model to continually improving it - following parameter optimization and predictor pool selection - to arrive at the final model. In practical sense, having a systematic process helps implement the model in an automated fashion.

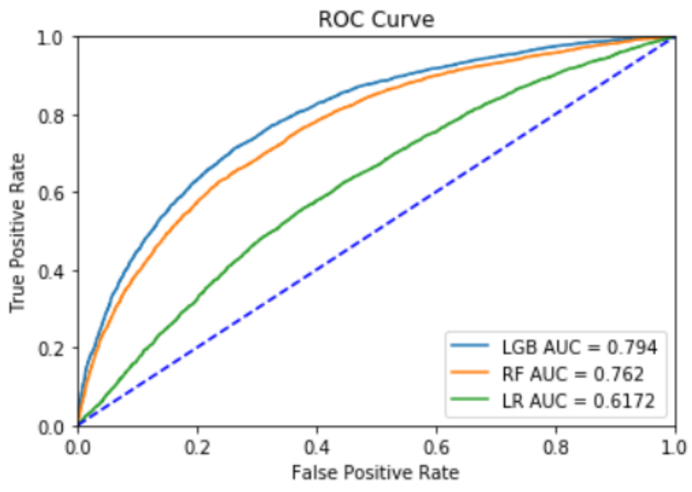


Figure 22: Models' ROC curve, AUC

6. CONCLUSION AND POTENTIAL FUTURE WORK

This report provides a systematic model development approach to machine learning models to model a practical home loan default risk problem to predict customers who would potentially default.

The introduction and background sections establishes the motivation and model development framework for various model development. We also provided the theory on various models, model evaluation to establish the context for the readers and foundation leading to the development of these models for the problem.

The report persents an extensive exploratory data analyses to provide insight into the data; the study included correlation analysis and univariate analyses on variables.

We estimated three models, Logistic Regression, Random Forest and Light GBM, and reviewed their results using performance metrics such as ROC curve/ AUC-Score and F1-Score. We found that overall Light GBM provides the best predictions.

We provided a very systematic process to developing the models, which makes the model easy to implement in a practical setting. However, we do think that certain aspects of the model estimation can be enhanced to improve model's performance be used for a practical purpose; and we would like a propose those enhancements as part of future study.

The potential future work may include:

- Considering inclusion of macroeconomic data to build such models. Inclusion of this data would allow for even more relevant predictions, and would enable creating forecasts for multiple different scenarios - most likely scenario and economically adverse scenarios.
- Developing a specific cost function, which takes into account the specific prediction goal, to evaluate the models.
- Analyzing variable selection by applying the home loan credit risk domain knowledge. Completely relying on the data and not applying business sense sometimes steer the estimation process in the wrong direction.

References

[1] P. Härle, A. Havas, and H. Samandari, "The future of bank risk management," *Mckinsey& Co*, 2016.

[2] D. Bacham and J. Zhao, "Machine Learning: challenges, lessons, and opportunities in credit risk modeling." https://www.moodysanalytics.com/risk-perspectives-magazine/managing-disruption/spotlight/machine-learning-challenges-lessons-and-opportunities-in-credit-risk-modeling. Accessed: 2018-11-3.

[3] J. Galindo and P. Tamayo, "Credit risk assessment using statistical and machine learning: basic methodology and risk modeling applications," *Computational Economics*, vol. 15, no. 1-2, pp. 107–143, 2000.

[4] S. Dreiseitl and L. Ohno-Machado, "Logistic regression and artificial neural network classification models: a methodology review," *Journal of biomedical informatics*, vol. 35, no. 5-6, pp. 352–359, 2002.

[5] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*, vol. 398. John Wiley & Sons, 2013.

[6] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems 30* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), pp. 3146–3154, Curran Associates, Inc., 2017.

[7] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "Lightgbm: A highly efficient gradient boosting decision tree," in *Advances in Neural Information Processing Systems*, pp. 3146–3154, 2017.

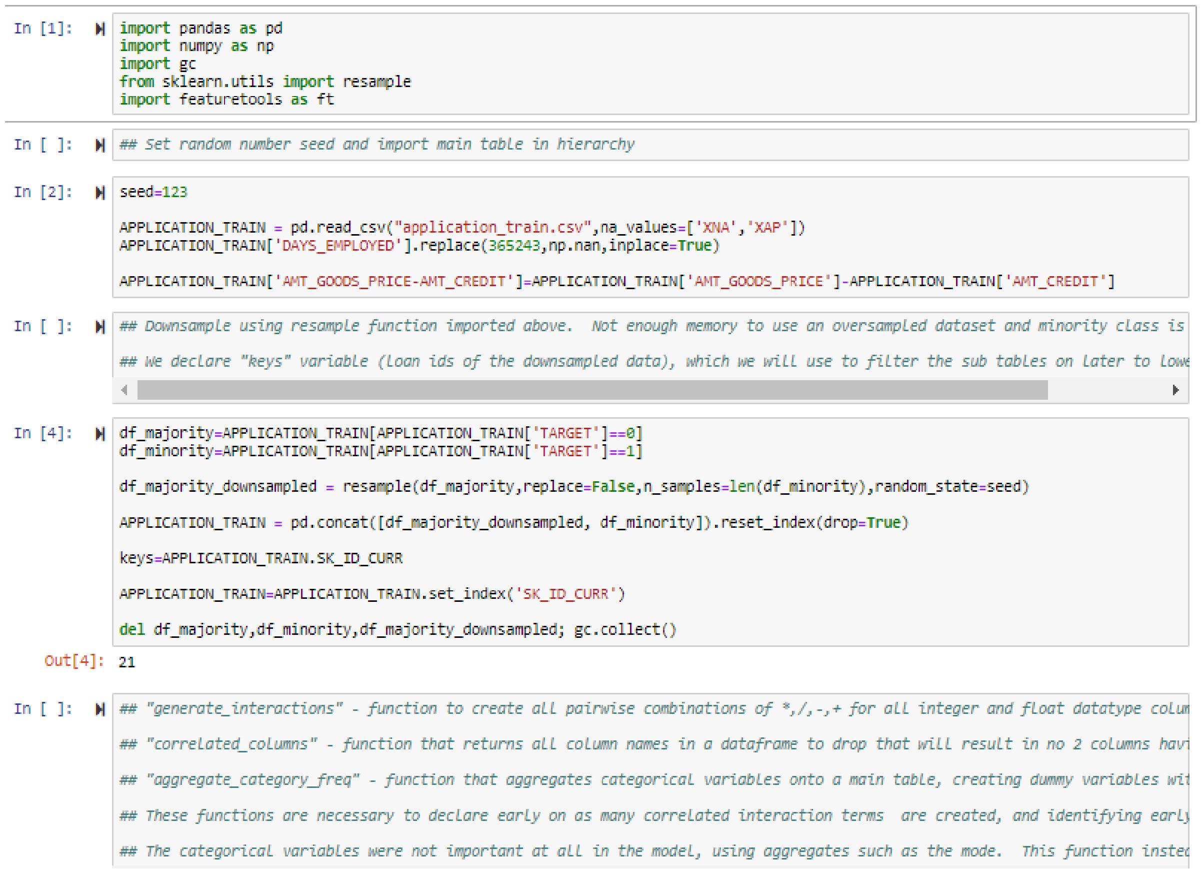
[8] H. Shi, *Best-first decision tree learning.* PhD thesis, The University of Waikato, 2007.

Appendix A. R code for exploratory data analysis

```
install.packages('DataExplorer')
library(DataExplorer)

library(data.table)
d_data <- fread("application_train.csv",select=c(1:21, 23:40))
p_data = fread("application_train.csv", select = c(1:6, 8:11, 17:20, 23,
27, 30:32, 42:72, 97:102))

p1_data = fread("application_train.csv", select = c(1:11, 17, 45:71))

plot_correlation(na.omit(p1_data), maxcat = 5L)

full_data = fread("application_train.csv")

summary(d_data)

cor(p1_data$TARGET, p1_data$AMT_CREDIT)

introduce(full_data)
plot_intro(full_data)
plot_missing(full_data)
missingdata = profile_missing(full_data)
missingdata


plot_bar(full_data)
plot_histogram(full_data)

ncol(d_data)
colnames(d_data)

plot_str(d_data)
plot_missing(d_data)
plot_histogram(d_data) ##- approximation of shape of distribution
plot_density(d_data) ## probability density - shows smooth rep of
distribution as funciton - prob density funciton  tell us observation is
found within a range of values...

plot_correlation(d_data, type = 'continuous')
plot_correlation(d_data, type = 'discrete')
plot_correlation(d_data, type = 'all')

create_report(d_data)
### Boruta package to pick important features
data_na <- read.csv("application_train.csv", sep=",", header=T,
strip.white = T, na.strings = c("NA","NaN","","?"))
colnames(data_na)

##categorical
categorical_na <- c(3,4,5,6,12,13,14,15,16,29,33,41,91,92,94,95) #
```

```
Select the categorical variables
data_na[,categorical] <- data.frame(apply(data_na[categorical_na], 2,
as.factor)) # Convert the categorical variables to categorical variables
in R
install.packages("Boruta", repos = "https://cran.r-project.org")
library(Boruta)
data_na = na.omit(data_na)
boruta <- Boruta(TARGET~., data = data_na, doTrace = 2)
print(boruta)
# Boruta performed 99 iterations in 13.07772 mins.
# 55 attributes confirmed important: AMT_ANNUITY, AMT_CREDIT,
AMT_GOODS_PRICE, AMT_INCOME_TOTAL, APARTMENTS_AVG
# and 50 more;
# 65 attributes confirmed unimportant: AMT_REQ_CREDIT_BUREAU_DAY,
AMT_REQ_CREDIT_BUREAU_HOUR,
# AMT_REQ_CREDIT_BUREAU_MON, AMT_REQ_CREDIT_BUREAU_QRT,
AMT_REQ_CREDIT_BUREAU_WEEK and 60 more;
#1 tentative attributes left: DAYS_ID_PUBLISH;

plot(boruta, xlab = "", xaxt = "n")
lz<-lapply(1:ncol(boruta$ImpHistory),function(i)
  boruta$ImpHistory[is.finite(boruta$ImpHistory[,i]),i])
names(lz) <- colnames(boruta$ImpHistory)
Labels <- sort(sapply(lz,median))
axis(side = 1,las=2,labels = names(Labels),
     at = 1:ncol(boruta$ImpHistory), cex.axis = 0.7)

boruta$finalDecision

boruta$ImpHistory
```

Appendix B. Python code for feature engineering



```
In [1]:   import pandas as pd
          import numpy as np
          import gc
          from sklearn.utils import resample
          import featuretools as ft

In [ ]:   ## Set random number seed and import main table in hierarchy

In [2]:   seed=123

          APPLICATION_TRAIN = pd.read_csv("application_train.csv",na_values=['XNA','XAP'])
          APPLICATION_TRAIN['DAYS_EMPLOYED'].replace(365243,np.nan,inplace=True)

          APPLICATION_TRAIN['AMT_GOODS_PRICE-AMT_CREDIT']=APPLICATION_TRAIN['AMT_GOODS_PRICE']-APPLICATION_TRAIN['AMT_CREDIT']

In [ ]:   ## Downsample using resample function imported above.  Not enough memory to use an oversampled dataset and minority class is

          ## We declare "keys" variable (loan ids of the downsampled data), which we will use to filter the sub tables on later to lowe

In [4]:   df_majority=APPLICATION_TRAIN[APPLICATION_TRAIN['TARGET']==0]
          df_minority=APPLICATION_TRAIN[APPLICATION_TRAIN['TARGET']==1]

          df_majority_downsampled = resample(df_majority,replace=False,n_samples=len(df_minority),random_state=seed)

          APPLICATION_TRAIN = pd.concat([df_majority_downsampled, df_minority]).reset_index(drop=True)

          keys=APPLICATION_TRAIN.SK_ID_CURR

          APPLICATION_TRAIN=APPLICATION_TRAIN.set_index('SK_ID_CURR')

          del df_majority,df_minority,df_majority_downsampled; gc.collect()

   Out[4]: 21

In [ ]:   ## "generate_interactions" - function to create all pairwise combinations of *,/,-,+ for all integer and float datatype colum

          ## "correlated_columns" - function that returns all column names in a dataframe to drop that will result in no 2 columns havi

          ## "aggregate_category_freq" - function that aggregates categorical variables onto a main table, creating dummy variables wit

          ## These functions are necessary to declare early on as many correlated interaction terms  are created, and identifying early

          ## The categorical variables were not important at all  in the model, using aggregates such as the mode.  This function instea
```

In [5]:
```
def generate_interactions(df,colstart):
    column_names=df[df.columns[colstart:]].select_dtypes(include=[np.int64,np.float64]).columns
    subcolumns_exclude=[]
    for col in column_names:
        subcolumns_exclude.append(col)
        for subcol in column_names:
            if subcol not in subcolumns_exclude:
                df[col+'*'+subcol]=df[col]*df[subcol]
                df[col+'/'+subcol]=df[col]/df[subcol]
                df[col+'+'+subcol]=df[col]+df[subcol]
                df[col+'-'+subcol]=df[col]-df[subcol]
    return df

def correlated_columns(df,threshold):

    corr_matrix = df.corr().abs()

    upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(np.bool))
    to_drop = [column for column in upper.columns if any(upper[column] > threshold)]
    print('There are %d total columns.' % (len(df.columns)))
    print('There are %d columns to remove.' % (len(to_drop)))

    return to_drop

def aggregate_category_freq(df,df_main,df_name,joinID):

    for col in df.select_dtypes(include=[object]).columns:
        for cat in df[col].drop_duplicates().dropna():
            catcol_aggs=pd.DataFrame((df[df[col]==cat].groupby(joinID)[col].agg('count')/df.groupby(joinID)[co
l].agg('count')).fillna(0))
            catcol_aggs.columns=[df_name+'.'+col+'_'+cat]
            df_main=df_main.join(catcol_aggs[df_name+'.'+col+'_'+cat],how='left')

    return df_main
```

In [ ]:
```
## Declare the entityset using featuretools package.  This is how all the subtables and main table will be com
bined to interact with one another when creating aggregate features.

## Variables coded as numbers, specifically here "0,1", needs to be specified as categorical so the correct ag
gregate columns are created.  For example, we should take the mode (maybe "1", instead of the mean (maybe .71
2) of a binary column of 0s and 1s.
```

In [8]:
```
es = ft.EntitySet('loan_data')
```

In [ ]:
```
## Read previous application dataset, filter, replace bad data ("365243" is an extreme outlier in all of these
 columns, concluding that this is probably how one of the data sources defines their NA fields)

## Use functions declared earlier to create interaction terms and drop intercorrelated ones.
```

In [9]:
```
PREVIOUS_APPLICATION = pd.read_csv("previous_application.csv",na_values=['XNA','XAP'])
PREVIOUS_APPLICATION=PREVIOUS_APPLICATION[PREVIOUS_APPLICATION.SK_ID_CURR.isin(keys)]
col_count=len(PREVIOUS_APPLICATION.columns)

for col in ['DAYS_LAST_DUE','DAYS_TERMINATION','DAYS_FIRST_DRAWING','DAYS_FIRST_DUE','DAYS_LAST_DUE_1ST_VERSIO
N']:
    PREVIOUS_APPLICATION[col].replace(365243, np.nan, inplace=True)

PREVIOUS_APPLICATION=pd.concat([PREVIOUS_APPLICATION[PREVIOUS_APPLICATION.columns[:2]],
                                PREVIOUS_APPLICATION[PREVIOUS_APPLICATION[PREVIOUS_APPLICATION.columns[2:]].co
unt().sort_values(ascending=False).index]],axis=1)
```

In [10]:
```
PREVIOUS_APPLICATION=generate_interactions(PREVIOUS_APPLICATION,2)
PREVIOUS_APPLICATION.dropna(how='all',axis=1,inplace=True)
PREVIOUS_APPLICATION=pd.concat([PREVIOUS_APPLICATION[PREVIOUS_APPLICATION.columns[:col_count]],
                                PREVIOUS_APPLICATION[PREVIOUS_APPLICATION[PREVIOUS_APPLICATION.columns[col_cou
nt:]].count().sort_values(ascending=False).index]],axis=1)
```

In [11]:
```
PREVIOUS_APPLICATION.drop(correlated_columns(PREVIOUS_APPLICATION,.95),axis=1,inplace=True)
APPLICATION_TRAIN=aggregate_category_freq(PREVIOUS_APPLICATION,APPLICATION_TRAIN,'PREVIOUS_APPLICATION','SK_ID
_CURR')
PREVIOUS_APPLICATION.drop(PREVIOUS_APPLICATION.select_dtypes(include=[object]).columns,axis=1,inplace=True)
```

```
There are 709 total columns.
There are 524 columns to remove.
```

In [ ]:
```
## Add previous application dataset to entityset
```

In [12]:
```
es.entity_from_dataframe(
    entity_id = 'PREVIOUS_APPLICATION',
    dataframe = PREVIOUS_APPLICATION,
    index = 'SK_ID_PREV',
)

del PREVIOUS_APPLICATION; gc.collect()
```

Out[12]:
```
371
```

In [ ]:
```
## This process is repetitive, same process is done for all sub tables
```

In [13]:
```
POS_CASH_BALANCE = pd.read_csv("POS_CASH_balance.csv")
POS_CASH_BALANCE=POS_CASH_BALANCE[POS_CASH_BALANCE.SK_ID_CURR.isin(keys)]
col_count=len(POS_CASH_BALANCE.columns)
POS_CASH_BALANCE=pd.concat([POS_CASH_BALANCE[POS_CASH_BALANCE.columns[:2]],
                            POS_CASH_BALANCE[POS_CASH_BALANCE[POS_CASH_BALANCE.columns[2:]].count().sort_value
s(ascending=False).index]],axis=1)
```

In [14]:
```
POS_CASH_BALANCE=generate_interactions(POS_CASH_BALANCE,2)
POS_CASH_BALANCE.dropna(how='all',axis=1,inplace=True)
POS_CASH_BALANCE=pd.concat([POS_CASH_BALANCE[POS_CASH_BALANCE.columns[:col_count]],
                            POS_CASH_BALANCE[POS_CASH_BALANCE[POS_CASH_BALANCE.columns[col_count:]].count().so
rt_values(ascending=False).index]],axis=1)
```

In [15]:
```
POS_CASH_BALANCE.drop(correlated_columns(POS_CASH_BALANCE,.95),axis=1,inplace=True)
APPLICATION_TRAIN=aggregate_category_freq(POS_CASH_BALANCE,APPLICATION_TRAIN,'POS_CASH_BALANCE','SK_ID_CURR')
POS_CASH_BALANCE.drop(POS_CASH_BALANCE.select_dtypes(include=[object]).columns,axis=1,inplace=True)
```

```
There are 48 total columns.
There are 18 columns to remove.
```

In [16]:
```
es.entity_from_dataframe(
    entity_id = 'POS_CASH_BALANCE',
    dataframe = POS_CASH_BALANCE,
    make_index = True,
    index='index',
)

del POS_CASH_BALANCE; gc.collect()
```

Out[16]:
```
210
```

In [17]:
```
CREDIT_CARD_BALANCE = pd.read_csv("credit_card_balance.csv")
CREDIT_CARD_BALANCE=CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE.SK_ID_CURR.isin(keys)]
col_count=len(CREDIT_CARD_BALANCE.columns)
CREDIT_CARD_BALANCE=pd.concat([CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE.columns[:2]],
                               CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE.columns[2:]].count
().sort_values(ascending=False).index]],axis=1)
```

In [18]:
```
CREDIT_CARD_BALANCE=generate_interactions(CREDIT_CARD_BALANCE,2)
CREDIT_CARD_BALANCE.dropna(how='all',axis=1,inplace=True)
#CREDIT_CARD_BALANCE=pd.concat([CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE.columns[:col_count]],
#                               CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE[CREDIT_CARD_BALANCE.columns[col_coun
t:]].count().sort_values(ascending=False).index]],axis=1)
```

```
In [19]: CREDIT_CARD_BALANCE.drop(correlated_columns(CREDIT_CARD_BALANCE,.95),axis=1,inplace=True)
         APPLICATION_TRAIN=aggregate_category_freq(CREDIT_CARD_BALANCE,APPLICATION_TRAIN,'CREDIT_CARD_BALANCE','SK_ID_C
         URR')
         CREDIT_CARD_BALANCE.drop(CREDIT_CARD_BALANCE.select_dtypes(include=[object]).columns,axis=1,inplace=True)

         There are 783 total columns.
         There are 517 columns to remove.
```

```
In [20]: es.entity_from_dataframe(
             entity_id = 'CREDIT_CARD_BALANCE',
             dataframe = CREDIT_CARD_BALANCE,
             make_index = True,
             index='index',
         )

         del CREDIT_CARD_BALANCE; gc.collect()

Out[20]: 112
```

```
In [21]: INSTALLMENTS_PAYMENTS = pd.read_csv("installments_payments.csv")
         INSTALLMENTS_PAYMENTS=INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS.SK_ID_CURR.isin(keys)]
         col_count=len(INSTALLMENTS_PAYMENTS.columns)
         INSTALLMENTS_PAYMENTS=pd.concat([INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS.columns[:2]],
                                         INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS.columns[2:]].
         count().sort_values(ascending=False).index]],axis=1)
```

```
In [22]: INSTALLMENTS_PAYMENTS=generate_interactions(INSTALLMENTS_PAYMENTS,2)
         INSTALLMENTS_PAYMENTS.dropna(how='all',axis=1,inplace=True)
         INSTALLMENTS_PAYMENTS=pd.concat([INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS.columns[:col_count]],
                                           INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS[INSTALLMENTS_PAYMENTS.columns[col
         _count:]].count().sort_values(ascending=False).index]],axis=1)
```

```
In [23]: INSTALLMENTS_PAYMENTS.drop(correlated_columns(INSTALLMENTS_PAYMENTS,.95),axis=1,inplace=True)
         APPLICATION_TRAIN=aggregate_category_freq(INSTALLMENTS_PAYMENTS,APPLICATION_TRAIN,'INSTALLMENTS_PAYMENTS','SK_
         ID_CURR')
         INSTALLMENTS_PAYMENTS.drop(INSTALLMENTS_PAYMENTS.select_dtypes(include=[object]).columns,axis=1,inplace=True)

         There are 68 total columns.
         There are 39 columns to remove.
```

```
In [25]: es.entity_from_dataframe(
             entity_id = 'INSTALLMENTS_PAYMENTS',
             dataframe = INSTALLMENTS_PAYMENTS,
             make_index = True,
             index='index',
         )

         del INSTALLMENTS_PAYMENTS; gc.collect()

Out[25]: 49
```

```
In [26]: BUREAU=pd.read_csv("bureau.csv",na_values=['XNA','XAP'])
         BUREAU=BUREAU[BUREAU.SK_ID_CURR.isin(keys)]
         bureaukeys=BUREAU.SK_ID_BUREAU
         col_count=len(BUREAU.columns)
         BUREAU=pd.concat([BUREAU[BUREAU.columns[:2]],
                           BUREAU[BUREAU[BUREAU.columns[2:]].count().sort_values(ascending=False).index]],axis=1)
```

```
In [27]: BUREAU=generate_interactions(BUREAU,2)
         BUREAU.dropna(how='all',axis=1,inplace=True)
         BUREAUS=pd.concat([BUREAU[BUREAU.columns[:col_count]],
                            BUREAU[BUREAU[BUREAU.columns[col_count:]].count().sort_values(ascending=False).index]],axis
         =1)
         BUREAU=BUREAU.set_index('SK_ID_BUREAU')
```

```
In [29]: BUREAU.drop(correlated_columns(BUREAU,.95),axis=1,inplace=True)
         APPLICATION_TRAIN=aggregate_category_freq(BUREAU,APPLICATION_TRAIN,'BUREAU','SK_ID_CURR')
         BUREAU.drop(BUREAU.select_dtypes(include=[object]).columns,axis=1,inplace=True)

         There are 280 total columns.
         There are 153 columns to remove.
```

In [30]:
```
BUREAU_BALANCE = pd.read_csv("bureau_balance.csv",na_values=['X'])
BUREAU_BALANCE=BUREAU_BALANCE[BUREAU_BALANCE.SK_ID_BUREAU.isin(bureaukeys)]
```

In [31]:
```
BUREAU=aggregate_category_freq(BUREAU_BALANCE,BUREAU,'BUREAU_BALANCE','SK_ID_BUREAU')
BUREAU_BALANCE.drop(BUREAU_BALANCE.select_dtypes(include=[object]).columns,axis=1,inplace=True)
```

In [ ]:
```
## BUREAU_BALANCE table contains the status for each month in DPD.

## Many of the statuses were blank (Status of "X"), so flash fill method was used to fill in the previous mont
h's status of the of loan for each NA.  The model performed better leaving these as NA, so we abandoned this s
trategy.

# The status column is categorical, we tried converting this to numeric by replacing "C" (completed) status by
 0, but leaving as categorical was best for our model.
```

In [32]:
```
BUREAU.reset_index(level=0,inplace=True)
es.entity_from_dataframe(
    entity_id = 'BUREAU',
    dataframe = BUREAU,
    index = 'SK_ID_BUREAU',
)


del BUREAU; gc.collect()


es.entity_from_dataframe(
    entity_id = 'BUREAU_BALANCE',
    dataframe = BUREAU_BALANCE,
    make_index=True,
    index='index',
)


del BUREAU_BALANCE; gc.collect()
```

Out[32]:
```
28
```

In [33]:
```
APPLICATION_TRAIN.reset_index(level=0,inplace=True)
es.entity_from_dataframe(
    entity_id='APPLICATION_TRAIN',
    dataframe=APPLICATION_TRAIN.drop('TARGET', axis=1),
    index='SK_ID_CURR',
    variable_types={
        f: ft.variable_types.Categorical
        for f in APPLICATION_TRAIN.columns if f.startswith('FLAG_')
    }
)

del APPLICATION_TRAIN; gc.collect()


relationship1 = ft.Relationship(
    es['APPLICATION_TRAIN']['SK_ID_CURR'],
    es['PREVIOUS_APPLICATION']['SK_ID_CURR']
)

relationship2 = ft.Relationship(
    es['APPLICATION_TRAIN']['SK_ID_CURR'],
    es['BUREAU']['SK_ID_CURR']
)

relationship3 = ft.Relationship(
    es['APPLICATION_TRAIN']['SK_ID_CURR'],
    es['POS_CASH_BALANCE']['SK_ID_CURR']
)

relationship4 = ft.Relationship(
    es['APPLICATION_TRAIN']['SK_ID_CURR'],
    es['CREDIT_CARD_BALANCE']['SK_ID_CURR']
)


relationship5 = ft.Relationship(
    es['APPLICATION_TRAIN']['SK_ID_CURR'],
    es['INSTALLMENTS_PAYMENTS']['SK_ID_CURR']
)

relationship6 = ft.Relationship(
    es['BUREAU']['SK_ID_BUREAU'],
    es['BUREAU_BALANCE']['SK_ID_BUREAU']
)

es.add_relationship(relationship1)
es.add_relationship(relationship2)
es.add_relationship(relationship3)
es.add_relationship(relationship4)
es.add_relationship(relationship5)
es.add_relationship(relationship6)
```

Out[33]:
```
Entityset: loan_data
  Entities:
    PREVIOUS_APPLICATION [Rows: 234942, Columns: 169]
    POS_CASH_BALANCE [Rows: 1321544, Columns: 30]
    CREDIT_CARD_BALANCE [Rows: 488676, Columns: 266]
    INSTALLMENTS_PAYMENTS [Rows: 1813984, Columns: 30]
    BUREAU [Rows: 232790, Columns: 132]
    BUREAU_BALANCE [Rows: 2202742, Columns: 3]
    APPLICATION_TRAIN [Rows: 49650, Columns: 286]
  Relationships:
    PREVIOUS_APPLICATION.SK_ID_CURR -> APPLICATION_TRAIN.SK_ID_CURR
    BUREAU.SK_ID_CURR -> APPLICATION_TRAIN.SK_ID_CURR
    POS_CASH_BALANCE.SK_ID_CURR -> APPLICATION_TRAIN.SK_ID_CURR
    CREDIT_CARD_BALANCE.SK_ID_CURR -> APPLICATION_TRAIN.SK_ID_CURR
    INSTALLMENTS_PAYMENTS.SK_ID_CURR -> APPLICATION_TRAIN.SK_ID_CURR
    BUREAU_BALANCE.SK_ID_BUREAU -> BUREAU.SK_ID_BUREAU
```

In [ ]:
```
## Below we are creating the feature matrix (fm) that adds in aggregate features defined below for all sub tab
les.

## For all numeric features, we'll add the instance count, mean, median, skew, sum, standard deviation, max, a
nd min.

## categorical features were converted to numeric proportions (explained eariler).
```

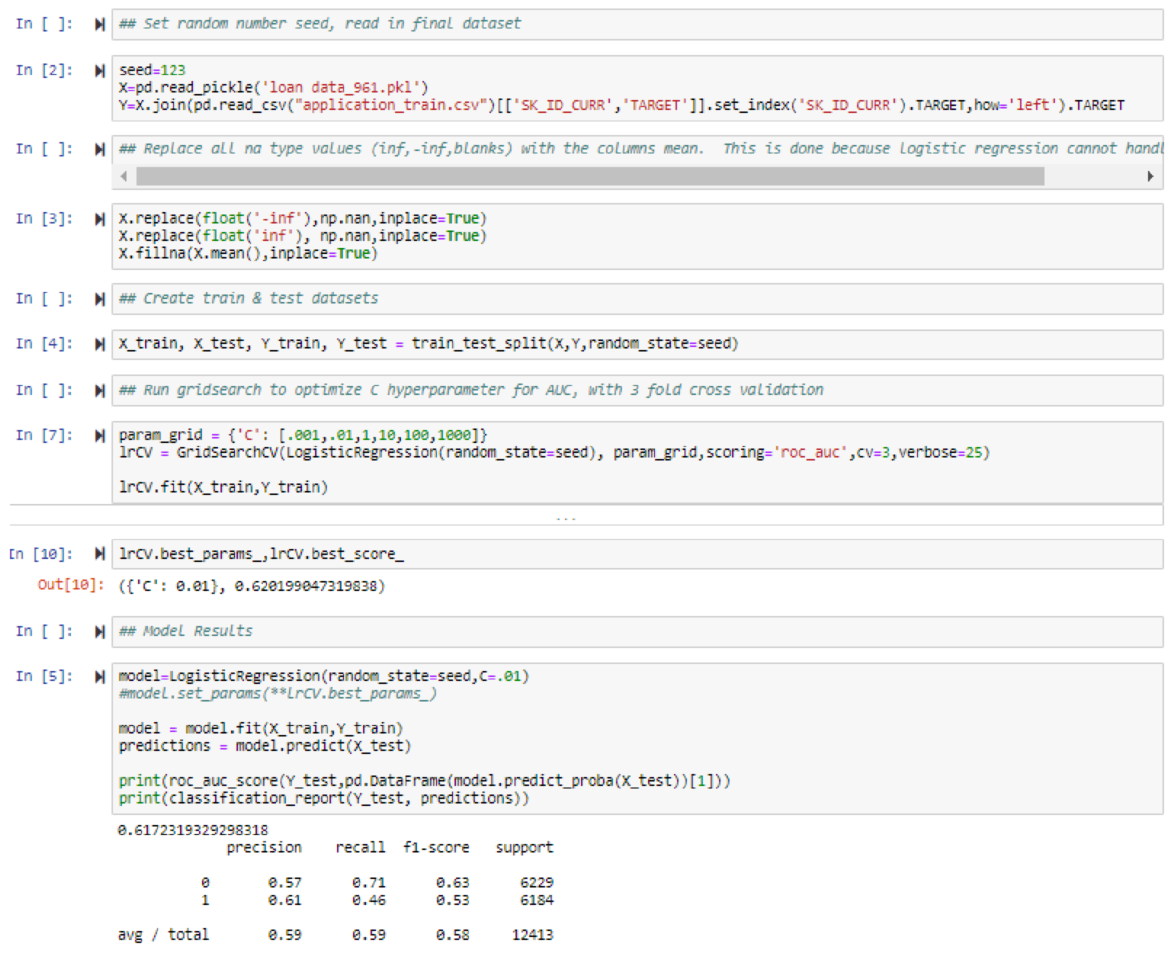
In [34]:
```
fm, feature_defs = ft.dfs(
    entityset=es,
    target_entity="APPLICATION_TRAIN",
    agg_primitives=[
        #"avg_time_between",
        #"time_since_last",
        "mean",
        "median",
        #"num_unique",
        "count",
        "skew",
        "sum",
        "std",
        #"mode",
        "max",
        "min"
    ],
    trans_primitives=[
        #"time_since_previous",
        #"cum_mean",
        #"cum_max",
        #"cum_min"
        #"percentile",
        #"add"
    ],
    max_depth=2,
    #cutoff_time=cutoff_times,
    training_window=ft.Timedelta(60, "d"),
    max_features=5000,
    chunk_size=4000,
    verbose=True,
)
```

```
Built 4667 features
Elapsed: 2:06:48 | Remaining: 00:00 | Progress: 100%|██████████| Calculated: 13/13 chunks
```

In [ ]:
```
## As this is a long process, will save to a .pkl file for use in the modeling notebook
```

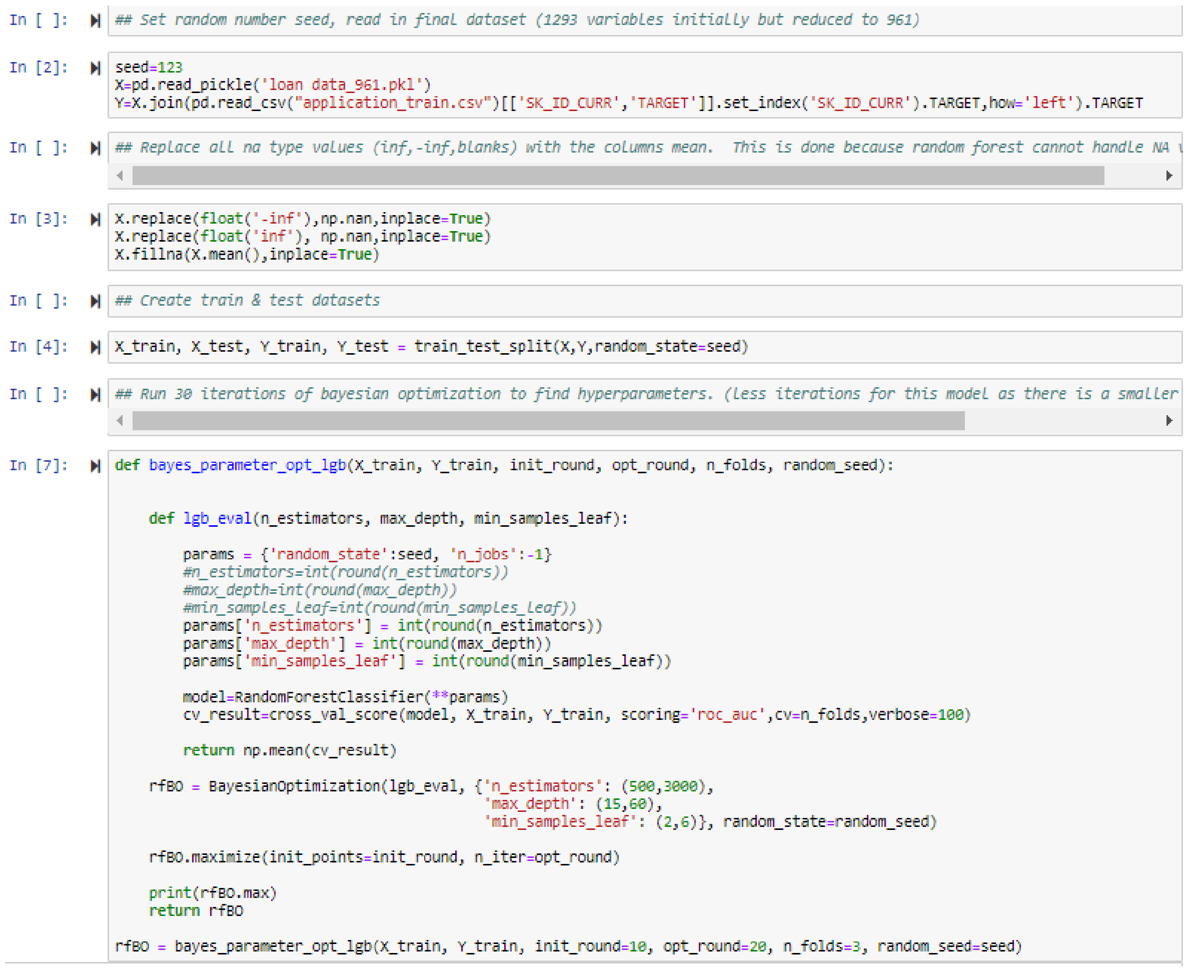
In [35]:
```
fm.to_pickle('loan data.pkl')
```

Appendix C. Python code for Logistic Regression



```
In [ ]:  ## Set random number seed, read in final dataset

In [2]:  seed=123
         X=pd.read_pickle('loan data_961.pkl')
         Y=X.join(pd.read_csv("application_train.csv")[['SK_ID_CURR','TARGET']].set_index('SK_ID_CURR').TARGET,how='left').TARGET

In [ ]:  ## Replace all na type values (inf,-inf,blanks) with the columns mean.  This is done because logistic regression cannot handl

In [3]:  X.replace(float('-inf'),np.nan,inplace=True)
         X.replace(float('inf'), np.nan,inplace=True)
         X.fillna(X.mean(),inplace=True)

In [ ]:  ## Create train & test datasets

In [4]:  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,random_state=seed)

In [ ]:  ## Run gridsearch to optimize C hyperparameter for AUC, with 3 fold cross validation

In [7]:  param_grid = {'C': [.001,.01,1,10,100,1000]}
         lrCV = GridSearchCV(LogisticRegression(random_state=seed), param_grid,scoring='roc_auc',cv=3,verbose=25)

         lrCV.fit(X_train,Y_train)

                                          ...

In [10]: lrCV.best_params_,lrCV.best_score_

Out[10]: ({'C': 0.01}, 0.620199047319838)

In [ ]:  ## Model Results

In [5]:  model=LogisticRegression(random_state=seed,C=.01)
         #model.set_params(**lrCV.best_params_)

         model = model.fit(X_train,Y_train)
         predictions = model.predict(X_test)

         print(roc_auc_score(Y_test,pd.DataFrame(model.predict_proba(X_test))[1]))
         print(classification_report(Y_test, predictions))

         0.6172319329298318
                      precision    recall  f1-score   support

                   0       0.57      0.71      0.63      6229
                   1       0.61      0.46      0.53      6184

         avg / total       0.59      0.59      0.58     12413
```

Appendix D. Python code for Random Forest



```
In [ ]:  ## Set random number seed, read in final dataset (1293 variables initially but reduced to 961)

In [2]:  seed=123
         X=pd.read_pickle('loan data_961.pkl')
         Y=X.join(pd.read_csv("application_train.csv")[['SK_ID_CURR','TARGET']].set_index('SK_ID_CURR').TARGET,how='left').TARGET

In [ ]:  ## Replace all na type values (inf,-inf,blanks) with the columns mean.  This is done because random forest cannot handle NA

In [3]:  X.replace(float('-inf'),np.nan,inplace=True)
         X.replace(float('inf'), np.nan,inplace=True)
         X.fillna(X.mean(),inplace=True)

In [ ]:  ## Create train & test datasets

In [4]:  X_train, X_test, Y_train, Y_test = train_test_split(X,Y,random_state=seed)

In [ ]:  ## Run 30 iterations of bayesian optimization to find hyperparameters. (Less iterations for this model as there is a smaller

In [7]:  def bayes_parameter_opt_lgb(X_train, Y_train, init_round, opt_round, n_folds, random_seed):


             def lgb_eval(n_estimators, max_depth, min_samples_leaf):

                 params = {'random_state':seed, 'n_jobs':-1}
                 #n_estimators=int(round(n_estimators))
                 #max_depth=int(round(max_depth))
                 #min_samples_leaf=int(round(min_samples_leaf))
                 params['n_estimators'] = int(round(n_estimators))
                 params['max_depth'] = int(round(max_depth))
                 params['min_samples_leaf'] = int(round(min_samples_leaf))

                 model=RandomForestClassifier(**params)
                 cv_result=cross_val_score(model, X_train, Y_train, scoring='roc_auc',cv=n_folds,verbose=100)

                 return np.mean(cv_result)

             rfBO = BayesianOptimization(lgb_eval, {'n_estimators': (500,3000),
                                                     'max_depth': (15,60),
                                                     'min_samples_leaf': (2,6)}, random_state=random_seed)

             rfBO.maximize(init_points=init_round, n_iter=opt_round)

             print(rfBO.max)
             return rfBO

         rfBO = bayes_parameter_opt_lgb(X_train, Y_train, init_round=10, opt_round=20, n_folds=3, random_seed=seed)
```

In [ ]:
```
## Convert necessary paramaters to integer format
```

In [44]:
```
rf_params=rfBO.max
rf_params['params']['n_estimators']=int(rf_params['params']['n_estimators'])
rf_params['params']['max_depth']=int(rf_params['params']['max_depth'])
rf_params['params']['min_samples_leaf']=int(rf_params['params']['min_samples_leaf'])
```

In [ ]:
```
## Model performance for test data
```

In [5]:
```
model=RandomForestClassifier(random_state=seed,n_jobs=-1,n_estimators=2000,max_depth=15,min_samples_leaf=5)
#model.set_params(**rf_params['params'])

model = model.fit(X_train,Y_train)
predictions = model.predict(X_test)

print(roc_auc_score(Y_test,pd.DataFrame(model.predict_proba(X_test))[1]))
print(classification_report(Y_test, predictions))
```

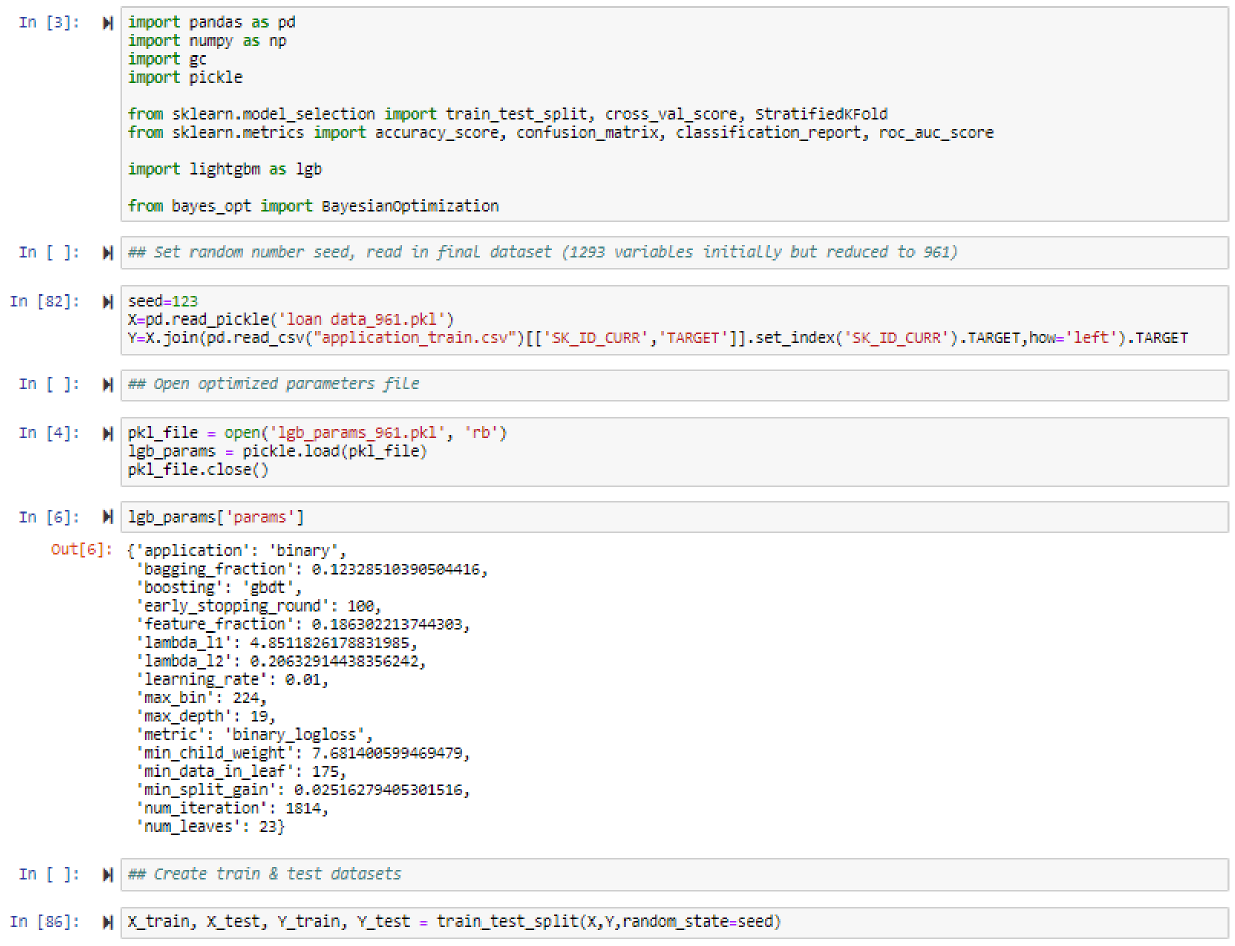
```
0.7619858091882127
             precision    recall  f1-score   support

          0       0.69      0.69      0.69      6229
          1       0.69      0.69      0.69      6184

avg / total       0.69      0.69      0.69     12413
```

In [ ]:
```
## Model performance for train data
```

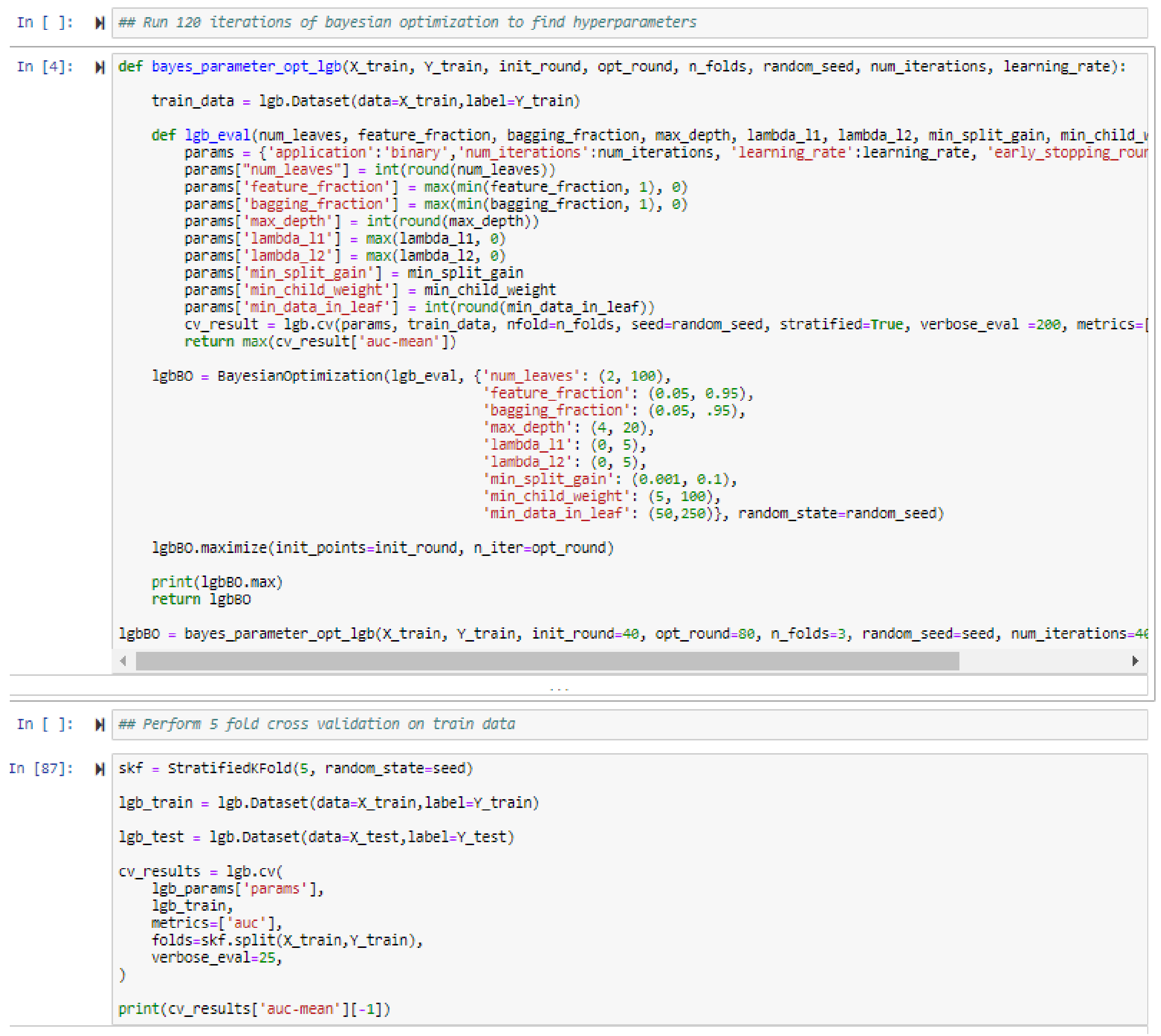
In [7]:
```
predictions = model.predict(X_train)
print(roc_auc_score(Y_train,pd.DataFrame(model.predict_proba(X_train))[1]))
print(classification_report(Y_train, predictions))
```

```
0.9880436391683465
             precision    recall  f1-score   support

          0       0.95      0.94      0.94     18596
          1       0.94      0.95      0.94     18641

avg / total       0.94      0.94      0.94     37237
```

Appendix E. Python code for Light GBM



```
In [3]:  import pandas as pd
         import numpy as np
         import gc
         import pickle

         from sklearn.model_selection import train_test_split, cross_val_score, StratifiedKFold
         from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

         import lightgbm as lgb

         from bayes_opt import BayesianOptimization

In [ ]:  ## Set random number seed, read in final dataset (1293 variables initially but reduced to 961)

In [82]: seed=123
         X=pd.read_pickle('loan data_961.pkl')
         Y=X.join(pd.read_csv("application_train.csv")[['SK_ID_CURR','TARGET']].set_index('SK_ID_CURR').TARGET,how='left').TARGET

In [ ]:  ## Open optimized parameters file

In [4]:  pkl_file = open('lgb_params_961.pkl', 'rb')
         lgb_params = pickle.load(pkl_file)
         pkl_file.close()

In [6]:  lgb_params['params']

Out[6]: {'application': 'binary',
         'bagging_fraction': 0.12328510390504416,
         'boosting': 'gbdt',
         'early_stopping_round': 100,
         'feature_fraction': 0.186302213744303,
         'lambda_l1': 4.8511826178831985,
         'lambda_l2': 0.20632914438356242,
         'learning_rate': 0.01,
         'max_bin': 224,
         'max_depth': 19,
         'metric': 'binary_logloss',
         'min_child_weight': 7.681400599469479,
         'min_data_in_leaf': 175,
         'min_split_gain': 0.02516279405301516,
         'num_iteration': 1814,
         'num_leaves': 23}

In [ ]:  ## Create train & test datasets

In [86]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y,random_state=seed)
```



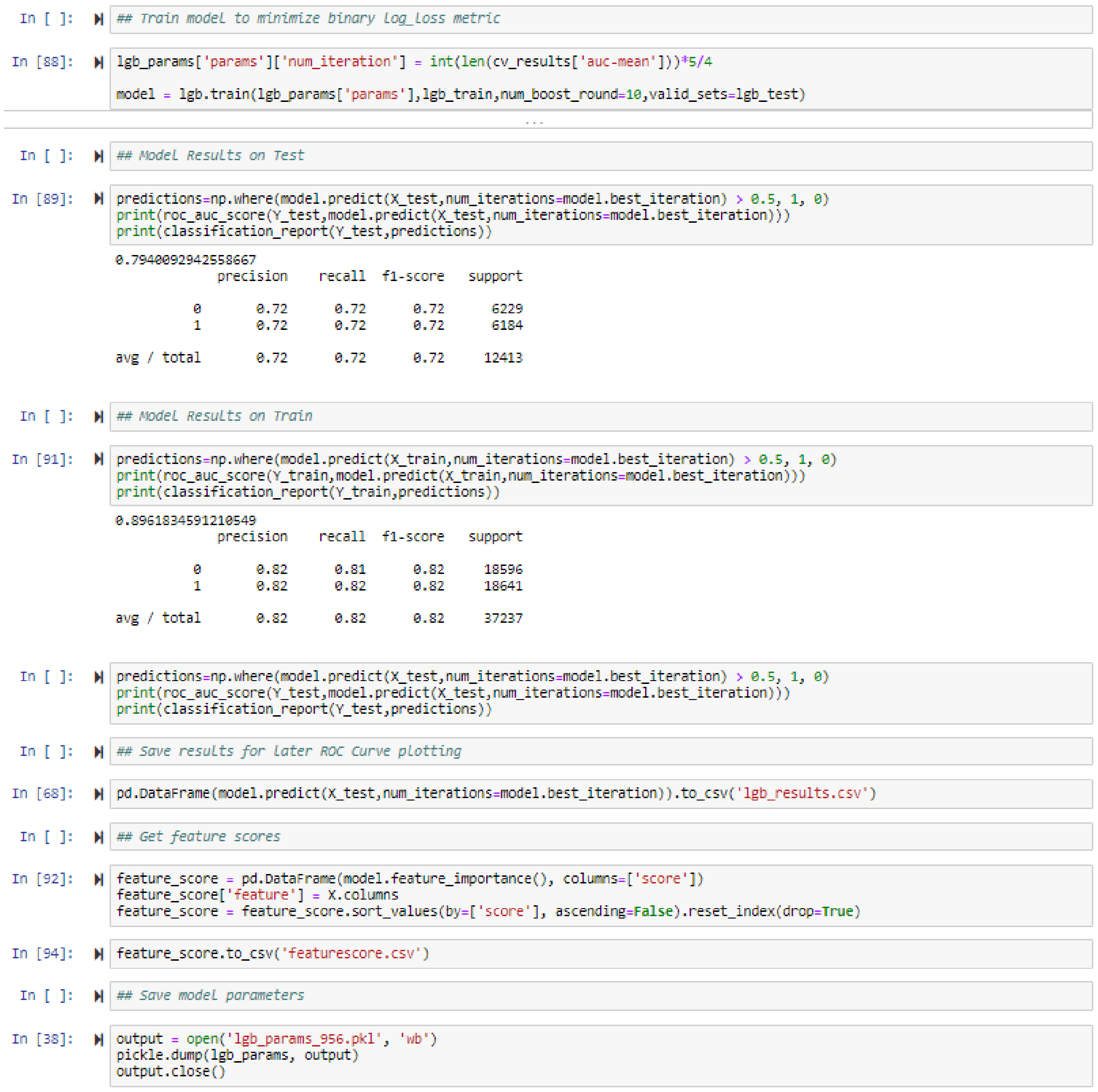
```
In [ ]:  ## Run 120 iterations of bayesian optimization to find hyperparameters
```

```
In [4]:  def bayes_parameter_opt_lgb(X_train, Y_train, init_round, opt_round, n_folds, random_seed, num_iterations, learning_rate):

             train_data = lgb.Dataset(data=X_train,label=Y_train)

             def lgb_eval(num_leaves, feature_fraction, bagging_fraction, max_depth, lambda_l1, lambda_l2, min_split_gain, min_child_w
                 params = {'application':'binary','num_iterations':num_iterations, 'learning_rate':learning_rate, 'early_stopping_rour
                 params["num_leaves"] = int(round(num_leaves))
                 params['feature_fraction'] = max(min(feature_fraction, 1), 0)
                 params['bagging_fraction'] = max(min(bagging_fraction, 1), 0)
                 params['max_depth'] = int(round(max_depth))
                 params['lambda_l1'] = max(lambda_l1, 0)
                 params['lambda_l2'] = max(lambda_l2, 0)
                 params['min_split_gain'] = min_split_gain
                 params['min_child_weight'] = min_child_weight
                 params['min_data_in_leaf'] = int(round(min_data_in_leaf))
                 cv_result = lgb.cv(params, train_data, nfold=n_folds, seed=random_seed, stratified=True, verbose_eval =200, metrics=[
                 return max(cv_result['auc-mean'])

             lgbBO = BayesianOptimization(lgb_eval, {'num_leaves': (2, 100),
                                                     'feature_fraction': (0.05, 0.95),
                                                     'bagging_fraction': (0.05, .95),
                                                     'max_depth': (4, 20),
                                                     'lambda_l1': (0, 5),
                                                     'lambda_l2': (0, 5),
                                                     'min_split_gain': (0.001, 0.1),
                                                     'min_child_weight': (5, 100),
                                                     'min_data_in_leaf': (50,250)}, random_state=random_seed)

             lgbBO.maximize(init_points=init_round, n_iter=opt_round)

             print(lgbBO.max)
             return lgbBO

         lgbBO = bayes_parameter_opt_lgb(X_train, Y_train, init_round=40, opt_round=80, n_folds=3, random_seed=seed, num_iterations=4(
```

```
...
```

```
In [ ]:  ## Perform 5 fold cross validation on train data
```

```
In [87]: skf = StratifiedKFold(5, random_state=seed)

         lgb_train = lgb.Dataset(data=X_train,label=Y_train)

         lgb_test = lgb.Dataset(data=X_test,label=Y_test)

         cv_results = lgb.cv(
             lgb_params['params'],
             lgb_train,
             metrics=['auc'],
             folds=skf.split(X_train,Y_train),
             verbose_eval=25,
         )

         print(cv_results['auc-mean'][-1])
```



```
In [ ]:  ## Train model to minimize binary log_loss metric

In [88]: lgb_params['params']['num_iteration'] = int(len(cv_results['auc-mean']))*5/4

         model = lgb.train(lgb_params['params'],lgb_train,num_boost_round=10,valid_sets=lgb_test)
         ...

In [ ]:  ## Model Results on Test

In [89]: predictions=np.where(model.predict(X_test,num_iterations=model.best_iteration) > 0.5, 1, 0)
         print(roc_auc_score(Y_test,model.predict(X_test,num_iterations=model.best_iteration)))
         print(classification_report(Y_test,predictions))

         0.7940092942558667
                      precision    recall  f1-score   support

                   0       0.72      0.72      0.72      6229
                   1       0.72      0.72      0.72      6184

         avg / total       0.72      0.72      0.72     12413

In [ ]:  ## Model Results on Train

In [91]: predictions=np.where(model.predict(X_train,num_iterations=model.best_iteration) > 0.5, 1, 0)
         print(roc_auc_score(Y_train,model.predict(X_train,num_iterations=model.best_iteration)))
         print(classification_report(Y_train,predictions))

         0.8961834591210549
                      precision    recall  f1-score   support

                   0       0.82      0.81      0.82     18596
                   1       0.82      0.82      0.82     18641

         avg / total       0.82      0.82      0.82     37237

In [ ]:  predictions=np.where(model.predict(X_test,num_iterations=model.best_iteration) > 0.5, 1, 0)
         print(roc_auc_score(Y_test,model.predict(X_test,num_iterations=model.best_iteration)))
         print(classification_report(Y_test,predictions))

In [ ]:  ## Save results for later ROC Curve plotting

In [68]: pd.DataFrame(model.predict(X_test,num_iterations=model.best_iteration)).to_csv('lgb_results.csv')

In [ ]:  ## Get feature scores

In [92]: feature_score = pd.DataFrame(model.feature_importance(), columns=['score'])
         feature_score['feature'] = X.columns
         feature_score = feature_score.sort_values(by=['score'], ascending=False).reset_index(drop=True)

In [94]: feature_score.to_csv('featurescore.csv')

In [ ]:  ## Save model parameters

In [38]: output = open('lgb_params_956.pkl', 'wb')
         pickle.dump(lgb_params, output)
         output.close()
```