Kaggle Machine Learning Project

Home Credit Default Risk Prediction

Academic Year: July 2020

Contents

1. Introduction 2
2. Dataset overview 3
3. Exploratory Data Analysis & Data Preprocessing 4
   3.1 Examination of the target variable 4
   3.2 Examination of the input variables 5
      3.2.1 Descriptive statistical summary 5
      3.2.2 Data quality check 8
4. Feature Engineering 9
   4.1 Feature creation 10
      4.1.1 Features created by domain knowledge 10
      4.1.2 Polynomial features 10
   4.2 Feature selection 10
      4.2.1 Feature importance 10
      4.2.2 Principle Component Analysis 12
      4.2.3 Correlation Analysis 13
5. Machine Learning Model Application 13
   5.1 Naïve Bayes 14
   5.2 Logistic Regression 17
   5.3 Random Forest 19
   5.4 LightGBM 21
   5.5 Voting and stacking Classifier 24
6. Conclusion 25
Appendix 26
References 32

1. Introduction

Traditionally, credit default risk is gauged using standard measurement tools like credit scorecards. With the rapid increases of data availability and computing power, machine learning now plays a vital role in both technology and finance. Credit default risk prediction has been one of the prevalent topics of machine learning application.

The credit default risk prediction is simply a binary classification problem. In this project, using the historical loan application data from Home Credit, a lending platform dedicated to providing loans to underserved borrowers, we implement several supervised machine learning algorithms to predict default risk of loan applicants. We also analyze and compare the model performance by ROC AUC and PR AUC. Since imbalanced dataset is a typical problem in credit default prediction and a severe skew in the class distribution would lead many machine learning algorithms astray, specifically resulting in ignorance of the minority class entirely, we use oversampling as an approach to randomly resample the training dataset before model application. For more advanced machine learning model like random forest and LightGBM, the hyperparameter class weight can be set appropriately to deal with imbalance. Most importantly, because our dataset is highly imbalanced with negative class dominates, ROC AUC can be misleadingly high with high recall but low precision. For this reason, we introduced PR AUC as another performance metric which is able to suggest the potential poor performance ROC AUC masks.

Besides model selection, data preprocessing and feature engineering are also key factors contributing to a reliable high-performance prediction model. To unlock the full potential of accessible information, we combine all relevant datasets from different sources, conduct a throughout examination of variables mainly focused on the detection of missing value and anomalies and generate smart feature creation and selection strategies.

The below flowchart demonstrates the whole process of our project. Additionally, with a structured model evolution process, single machine learning models are developed first and then ensemble methods, including bagging and Ad boost, and hyperparameter tuning by grid search and Bayesian optimization are applied to further empower models to be highly predictive of default risk. The classifier with ROC AUC of 0.77 and PR AUC of 0.25 is the best one we finally achieve. In the credit default classification problem, we are more tolerant of false positives than false negatives, so this result is acceptable.

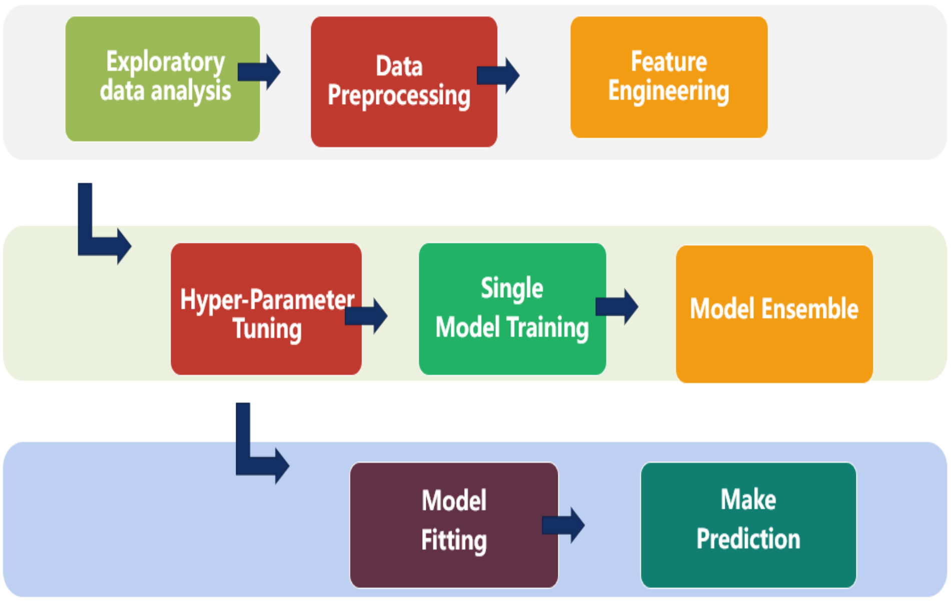
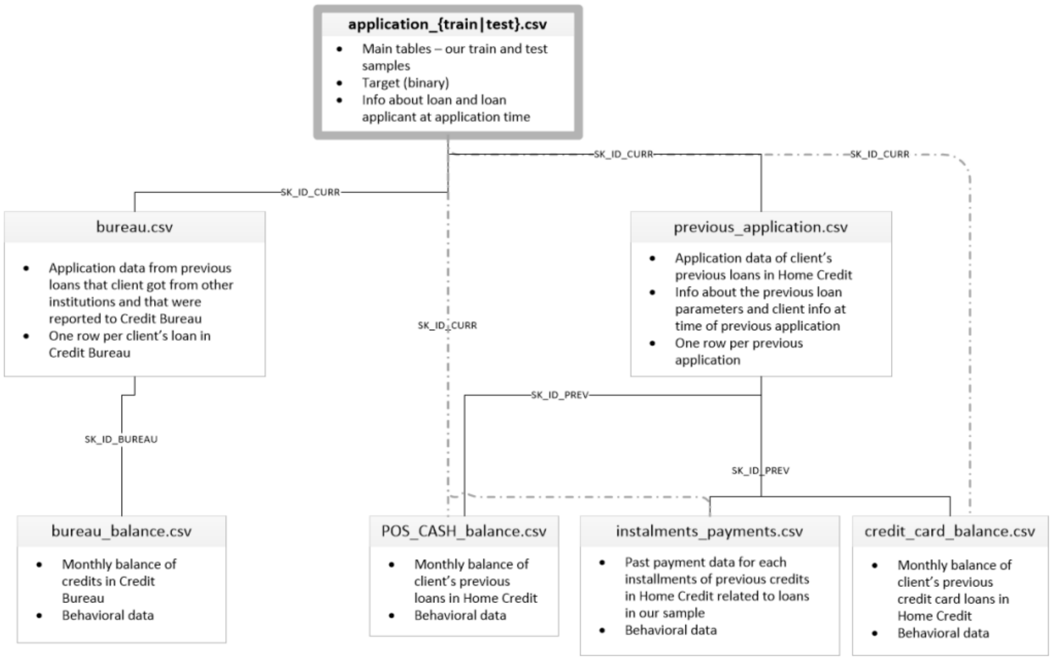


Figure 1: Workflow of credit default prediction

2. Dataset overview

The whole datasets contain a main training dataset with information about each loan applicants and 6 additional datasets regarding applicants' previous application records for loans in bureau and Home Credit. Different datasets are linked with ID of each applicants, so we merge all the dataset based on this unique ID. However, each secondary table has a many-to-one relationship with the main table. Therefore, we build our rules of data merging. After grouping all the records by ID, most frequent value is used for categorical variables, while the mean is used for numerical variables. The structure of the datasets is shown as below.



application_{train|test}.csv
Main tables – our train and test samples
Target (binary)
Info about loan and loan applicant at application time
SK_ID_CURR
SK_ID_CURR
SK_ID_CURR
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_CURR
previous_application.csv
Application data of client's previous loans in Home Credit
Info about the previous loan parameters and client info at time of previous application
One row per previous application
SK_ID_PREV
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 in Home Credit
Behavioral data
instalments_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 2 The structure of the whole dataset

The detailed descriptions of each dataset are listed as follows and the size of the dataset is presented in the bracket (rows × columns).

- application_train (307,512 × 122): the main training and testing data. Every loan has its own row and is identified by the feature SK_ID_CURR which is loan id.
- bureau (1,048,576 × 17): data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- bureau_balance (1,048,576 × 3): monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- previous_application (1,048,576 × 37): previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- POS_CASH_BALANCE (1,048,576 × 8): monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance (1,048,576 × 23): monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many ddrows.
- installments_payment (1,048,576 × 8): payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

3. Exploratory Data Analysis & Data Preprocessing

Apparently, exploratory data analysis allows us to better understand our datasets and give a preliminary discovery of the potential data problem we need to deal with later on. Feeding raw data to the model without preprocessing will produce unnecessary noise, making it hard to detect the patterns for reliable prediction. In this section, we statistically explore the data and the corresponding treatments of potential data problem are followed by the data description.

3.1 Examination of the target variable

The target variable is what we ask to predict. 0 stands for the loan is repaid on time, while 1 indicates that the applicant has difficulty in repaying the loan. As shown in the pie chart, 92% of the loans are repaid on time and only 8% are delinquent loans, indicating that loans that are repaid properly dominate the distribution. Obviously, the dataset has imbalanced problem, so model performance measured by accuracy is biased and resampling method should be applied as a treatment of this problem.

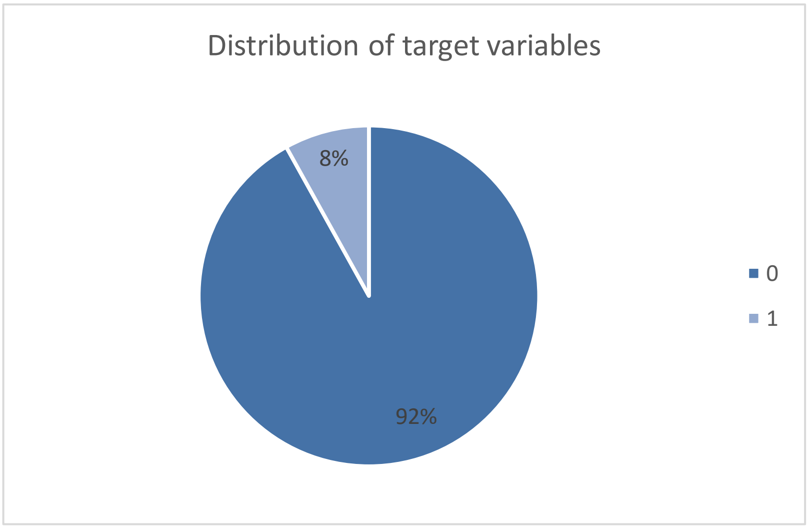


Figure 3: Distribution of target variables

3.2 Examination of the input variables

3.2.1 Descriptive statistical summary

(1) Type of the variables

Overall, 26% of the variables are categorical variables and the rest ones are all numerical variables. Among the categorical variables, the distribution of their unique values is skewed. Most of them have relatively small number of unique values, with 8 values on average while 57 values as maximum. The below table the statistical summary of the unique values of categorical variables.

Statistics	Value
Mean	8
min	2
25%	3
50%	5
75%	7
max	57

Table 1: Statistical summary of the unique values of categorical variables

(2) Distribution of the variables

Four categorical variables are selected as representative and the count of values in each category are plotted in histograms. From the following set of figures, it is evident that the counts are not evenly distributed. We remain this characteristic in our dataset and just use label encoding to convert all of them because some machine learning models cannot handle categorical variables directly.

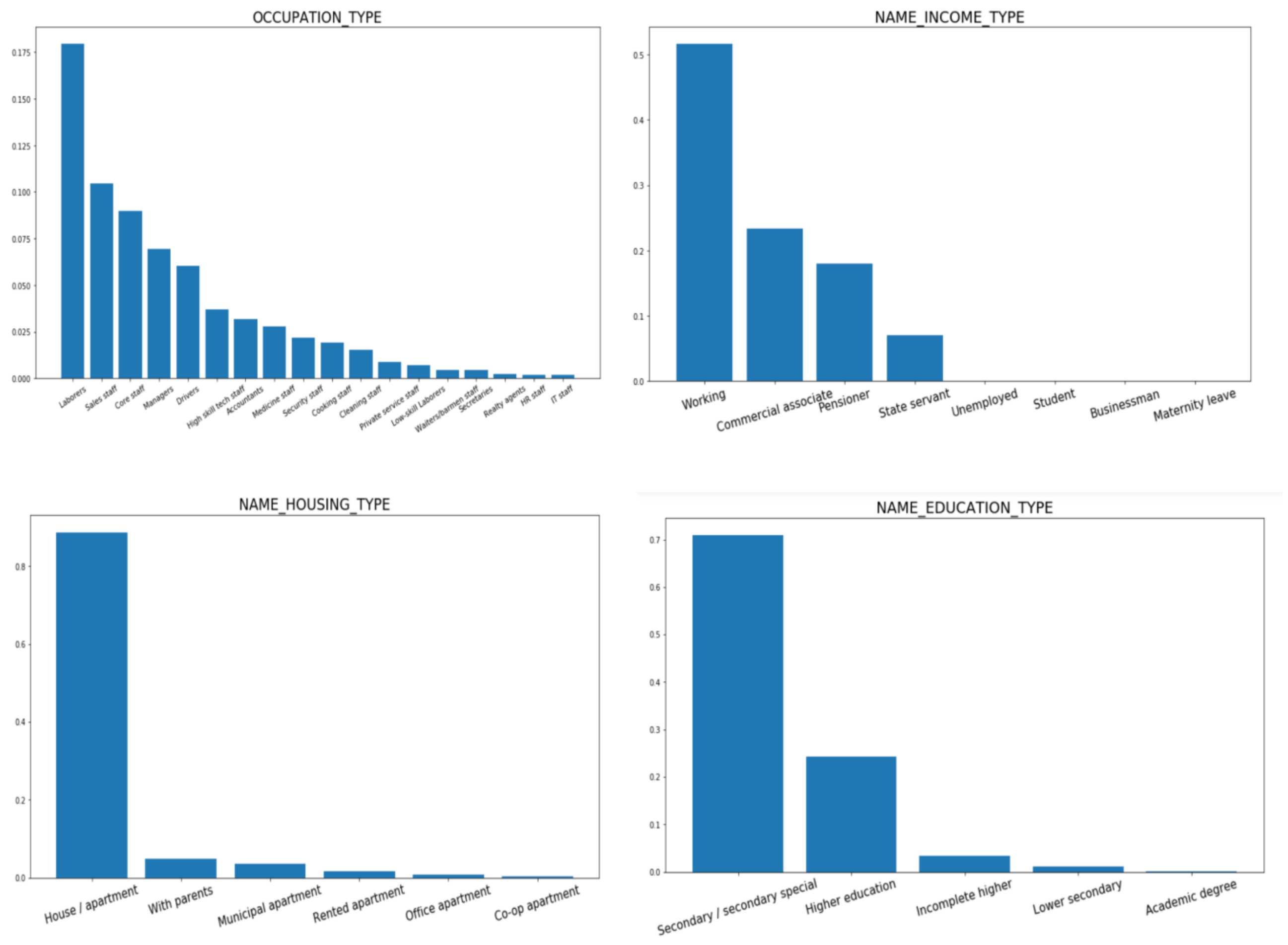
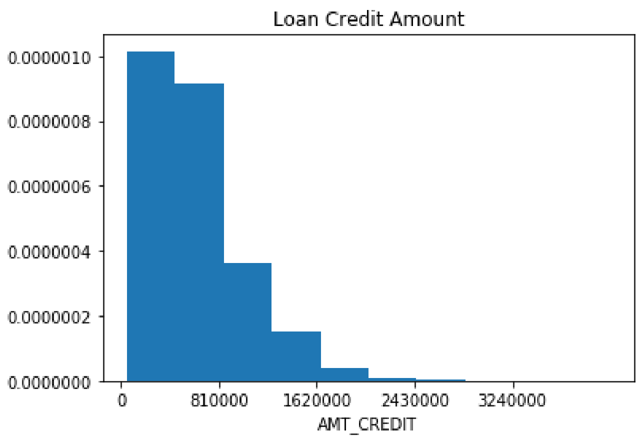
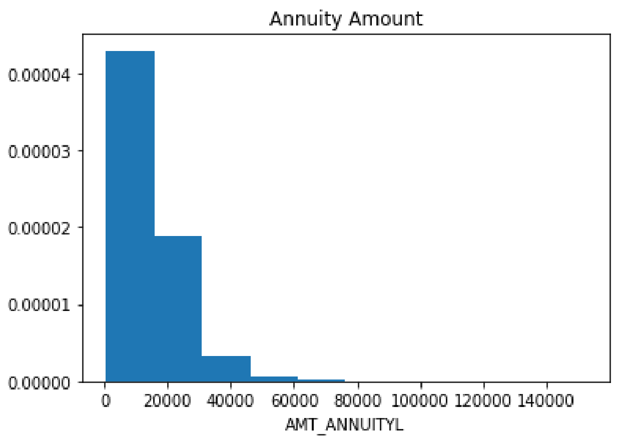


Figure 4: Distribution of categorical variables

The below plots are examples indicating that distribution of most of numerical variables in our dataset are positively skewed with data points are mostly concentrated on the left side. Certain machine learning models are quite sensitive to the skewness of the data, so Z-score is used for standardization. Narrowing down the range of inputs' values ensures the stability of the result.





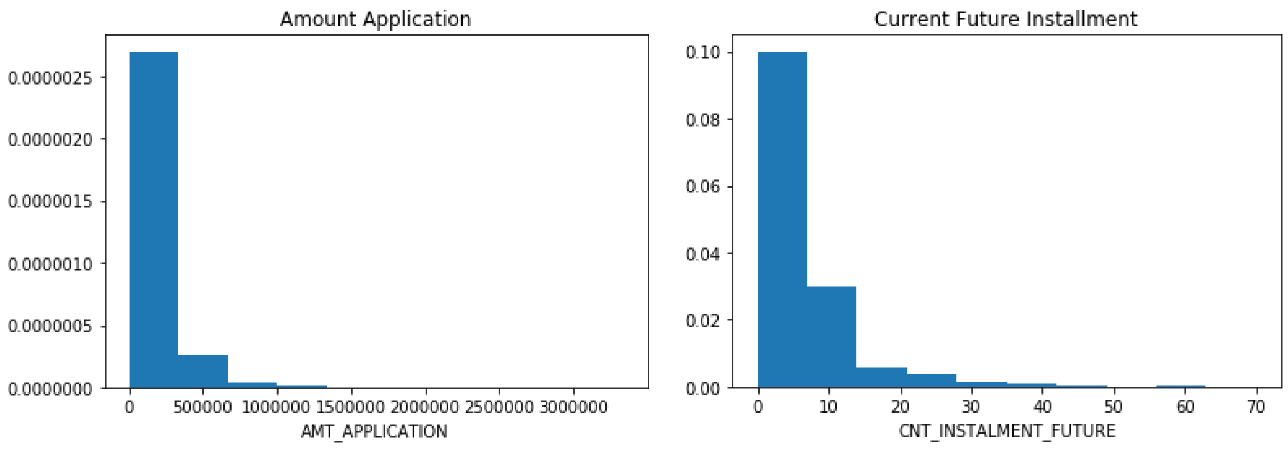


Figure 5: Distribution of numerical variables

(3) Correlation analysis

The correlation between different features and the target variable is visualized by the below heatmaps. Higher correlation with target means larger explanatory power, so we should keep these features. But if features are highly correlated with one another, this situation indicates that some of them are redundant inputs and should be dropped directly to improve the model performance. Based on our calculation, among the ten features only one pair of them is correlated over 0.7, reassuring us that high correlation problem is minor at this stage.

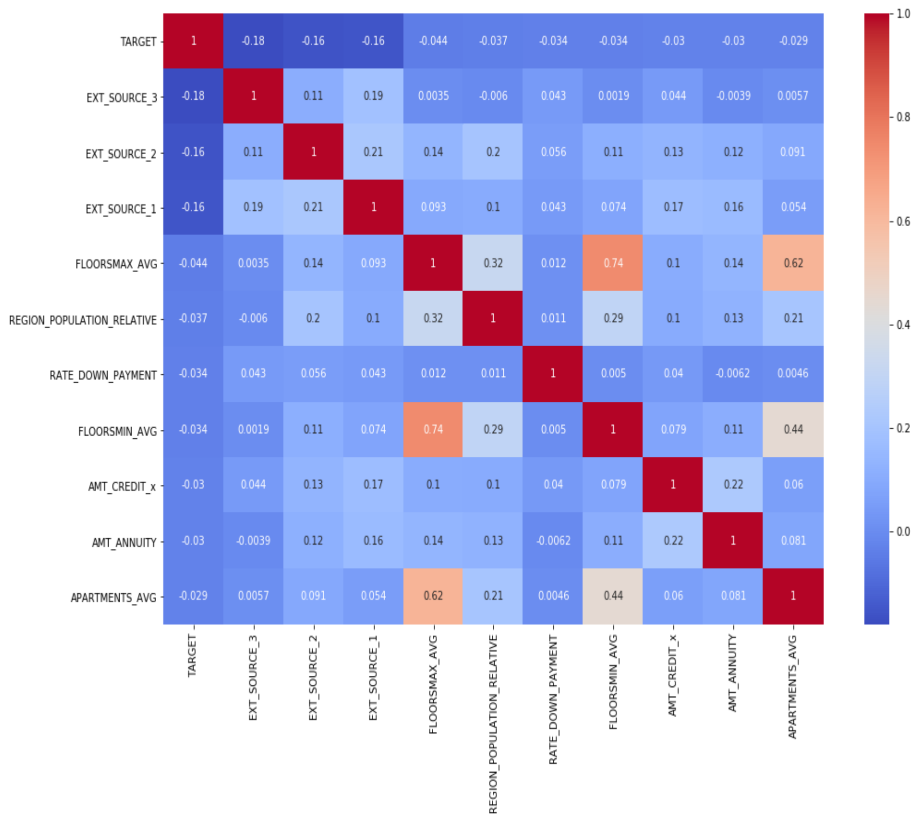


Figure 6: Heatmap of variables most negatively correlated with the target

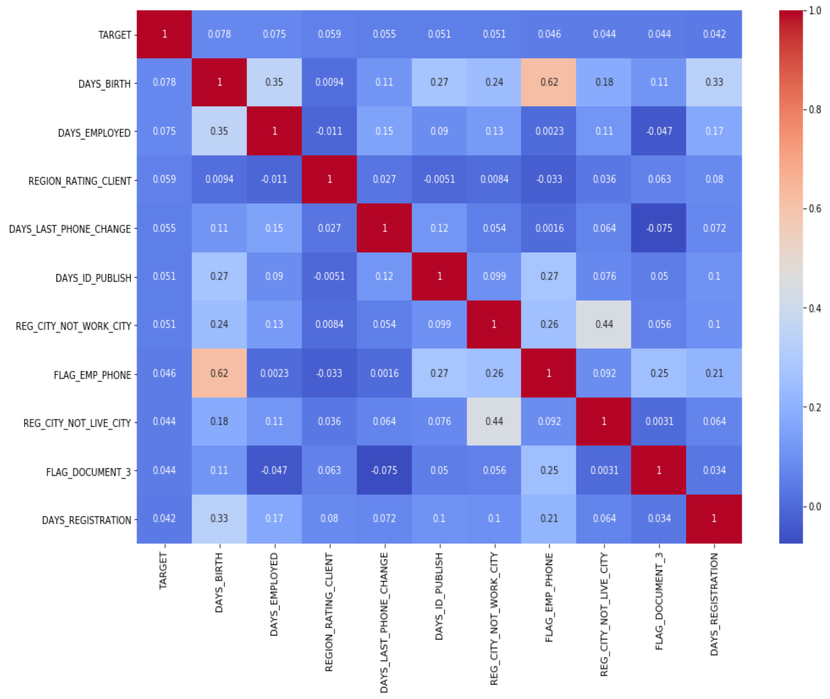


Figure 7: Heatmap of variables most positively correlated with the target

3.2.2 Data quality check

(1) Missing values

Checking missing value is a crucial step, because variables with high percentage of missing values contributes little to prediction and are likely to impede model performance. Most machine learning models cannot handle missing vales, so we are required to impute missing values before model application. Imputation fabricates additional information in the dataset and thus including features with too many missing values adversely affects our prediction.

As seen in the table, 50th quantile of missing values percentage is still zero. Thus, the missing value problem in the main dataset is not that severe, but we should still carefully screen the missing values in a descending order. In table 3, top 10 features are listed in terms of percentage of missing values. One simple way to deal with this issue is that we just discard the features with lots of missing values. However, if we strictly control missing values and drop too many features, we may lose some useful information which plays a role in prediction to some degree. Therefore, we set a minimum threshold for removing the features. For features whose percentage of missing value is greater than 70%, they will be dropped. After that, null values of remaining features will be filled with their mean value.

Statistics	Value
Mean	0.24
min	0
25%	0
50%	0
75%	0.5
max	0.6987

Table 2: Statistical summary of percentage of missing values in main dataset

Variables	Percentage of missing values
COMMONAREA_MEDI	0.6987
COMMONAREA_AVG	0.6987
COMMONAREA_MODE	0.6987
NONLIVINGAPARTMENTS_MODE	0.6943
NONLIVINGAPARTMENTS_MEDI	0.6943
NONLIVINGAPARTMENTS_AVG	0.6943
FONDKAPREMONT_MODE	0.6839
LIVINGAPARTMENTS_MEDI	0.6835
LIVINGAPARTMENTS_MODE	0.6835
LIVINGAPARTMENTS_AVG	0.6835

Table 3: List of variables with high percentage of missing values

(2) Anomalies
Anomalies can be easily detected by plotting the distribution. The below histograms illustrate that anomalies (highlighted in red) exist in the value of features. Intuitively, days greater than 30,000 and 350,000 are about 82 and 956 years respectively, which is not reasonable for credit end date and employed days. Leaving the anomalies in the dataset will bring misleading information and biased statistics and thus induce high possibility of wrong output given by the machine learning model. The safest approach we take is to firstly treat the anomalies the same as missing value. If the percentage of anomalies together with missing value exceed the threshold, namely 0.7, we will drop this feature, otherwise the anomalies will be filled with the average.

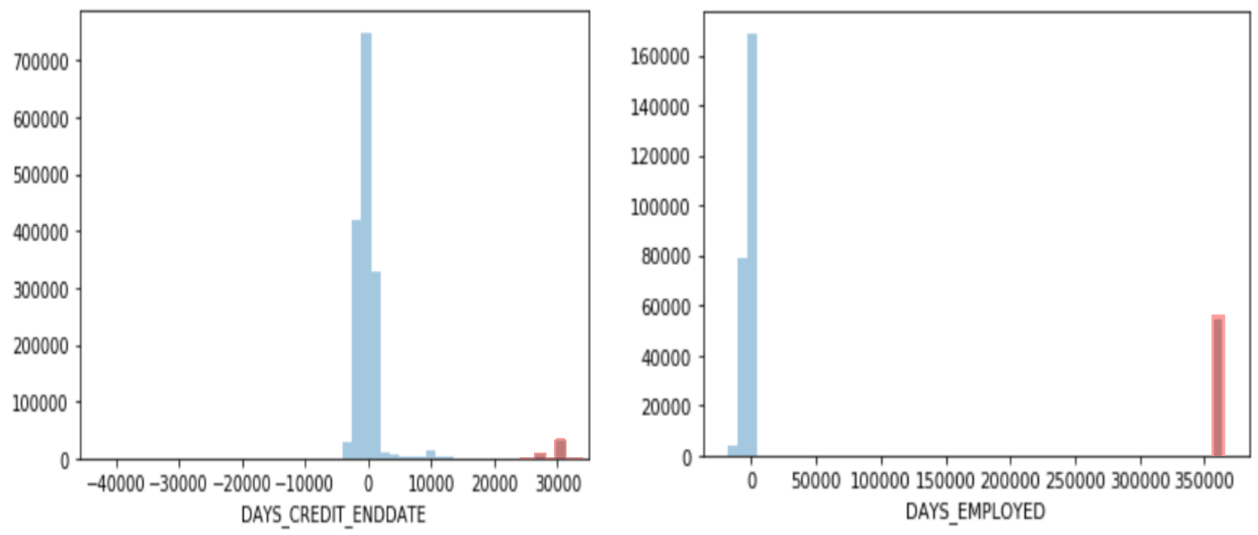


Figure 8: Anomalies in the features

4. Feature Engineering

As a rule of thumb, feature engineering can substantially boost machine learning model performance. Leveraging on domain knowledge and mathematic combination, more meaningful features can be extracted from raw data to provide efficient information as model inputs. In general, we create 8 new features with domain knowledge and 30 polynomial features. On the other hand, feature selection is also an indispensable process after feature engineering to reduce the dimension and eliminate irrelevant and redundant features. PCA has been tried as one of the feature selection strategy, but our later test shows that the models do not perform well after PCA. Therefore, just based on feature importance and correlation analysis, there are 255 features in total initially, whereas 124 features are included as inputs of our model training finally.

4.1 Feature creation

4.1.1 Features created by domain knowledge

Based on our domain expertise, we apply basic mathematic calculation to create features which further describe loan applicants in three aspects: current financial situation, credit profile and behavior profile.

(1) Current financial situation

The dependence on the loan shed light on applicants' current financial situation and can be quantitively evaluated by the debt credit ratio and credit application ratio defined as below. Intuitively, the higher the ratio is, the higher probability of default due to heavily reliance on loan.

- Debt credit ratio=Debt Amt/ Total Credit Amt
- Credit application ratio= Application Amt /Total Credit Amt

(2) Credit profile

The historical credit profile of loan applicant can be regarded as a supporting evidence for default prediction. Overdue debt ratio and days overdue are used to measure applicants' credit level. Higher overdue debt ratio and longer days overdue definitely indicates the high risk of delinquency.

- Overdue debt ratio=Overdue Amt/Total Credit Amt
- Days overdue=Days Enter Payment — Instalment Day

(3) Behavior profile

The probability of default can also be reasonably inferred from behavior patterns of loan applicants. From the pattern of loan repayment and usage perceived by the below indicators, applicants with higher default risk are strongly characterized.

- Installment ratio=current outstanding Installment /total Installment
- Payment installment ratio= Installment Amt /Payment Amt
- Draw limit ratio=Current Drawing Amt/ Credit Drawing limit
- Special expense= past average drawings — (current drawings+ other drawings)

4.1.2 Polynomial features

From the correlation calculation in exploratory data analysis, age and credit scores from 3 external institution, these 4 features are highly correlated with the target variable. It is interesting to see whether the interaction among them will enhance predictive power. As a result, polynomial features are created by these 4 features with degree of 3, meaning that they are nonlinearly combined by multiplication with the power no more than 3.

4.2 Feature selection

4.2.1 Feature importance

Tree-based machine learning models can determine the importance of the feature by how much they contribute to impurity reduction and the scores can be directly obtained by built-in function. To avoid overfitting problem, we fit the model 10th and use the average score of feature importance.

The below charts show the top 15 features we get from random forest and LightGBM classifier. The newly created features are listed, such as debt ratio, days overdue and interaction between two credit scores from external institution, demonstrating the success of our feature engineering.

The removal of zero importance features can expedite the training process and kind of highlight key information to help algorithms focus more on what's importance. Thus, this practice is implemented in our feature selection. The curve of cumulative importance presents that about 80 features account for nearly 90% of importance, so this result further secure that dropping features won't affect model performance.

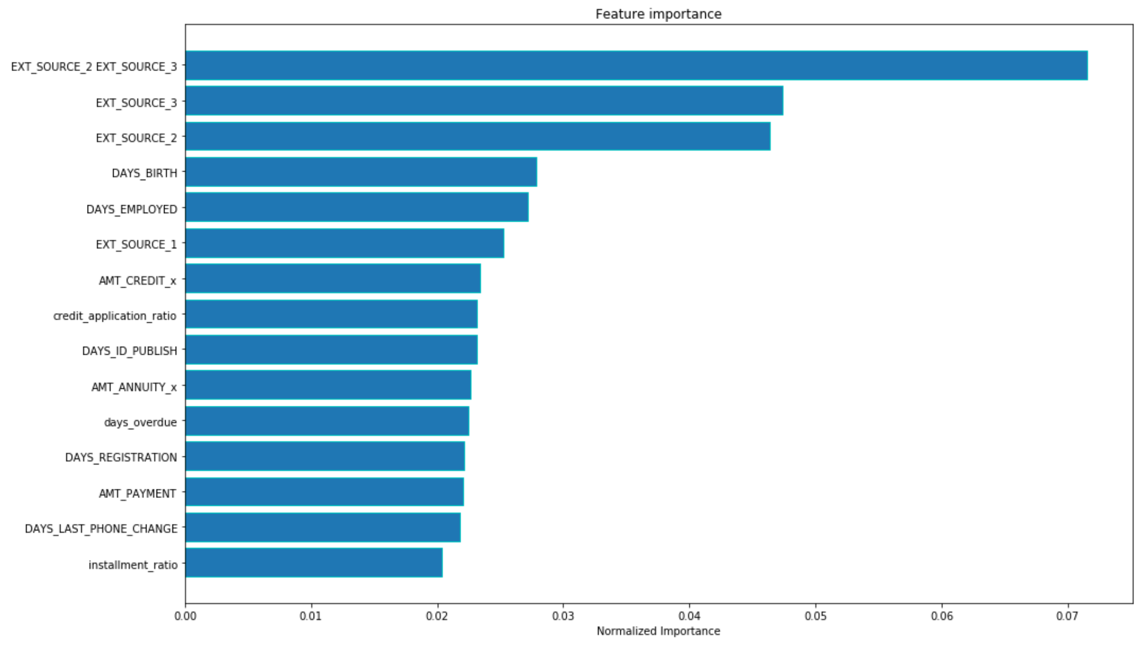


Figure 9: Top 15 important feature from Random forest

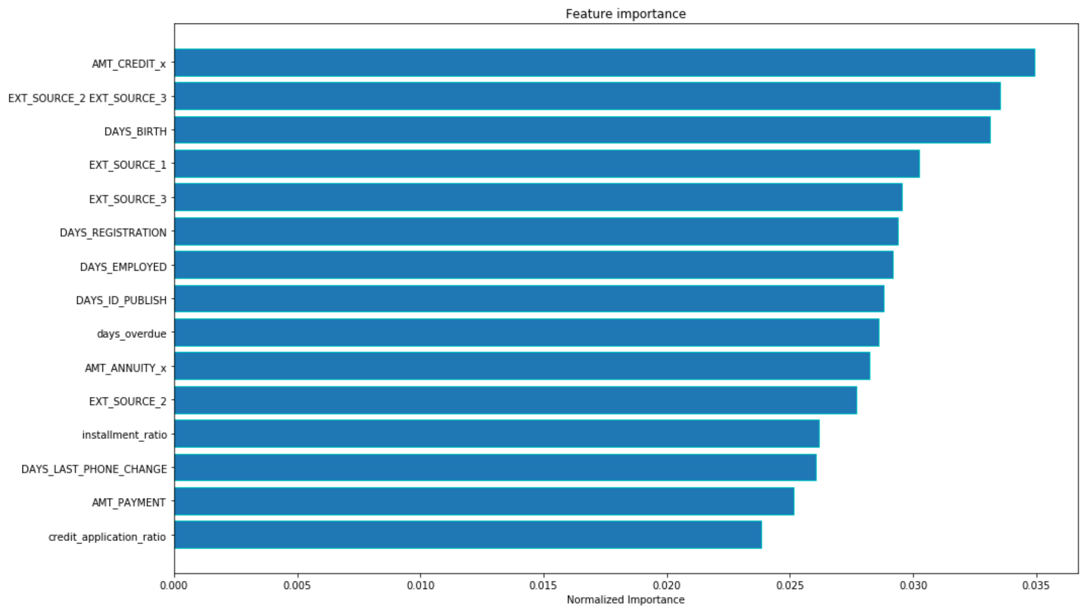


Figure 10: Top 15 important feature from LightGBM

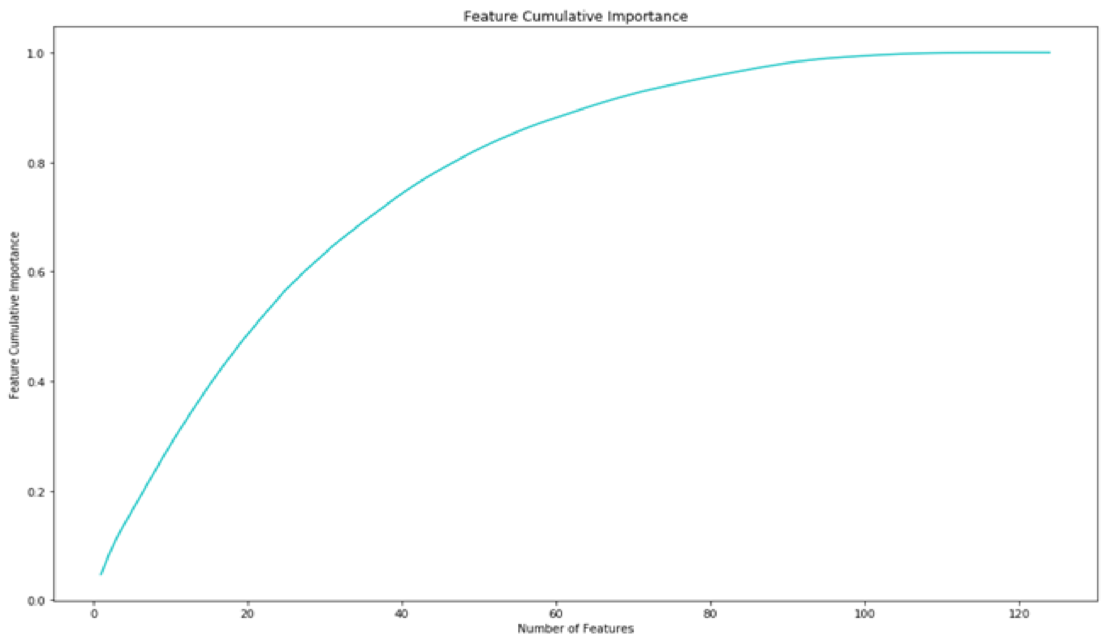


Figure 11: Cumulative importance of features

4.2.2 Principle Component Analysis

Principal component analysis can be a reasonable choice for dimension reduction. The features are projected into lower dimension and transformed into new components which can still attribute to explanation of the target. The explanatory ability of each principle components is measured by explained variance ratio. PCA is affected by scales, so we standardized the data before applying it. As shown in the below plot, approximately 100 PCs can explain 90% of total variance, so we keep all these PCs without seriously lowering predictive ability. But the transformation somewhat makes the features lose their original meaning and information, as proved by the model performance compared with not applying PCA, it is not used for feature selection eventually.

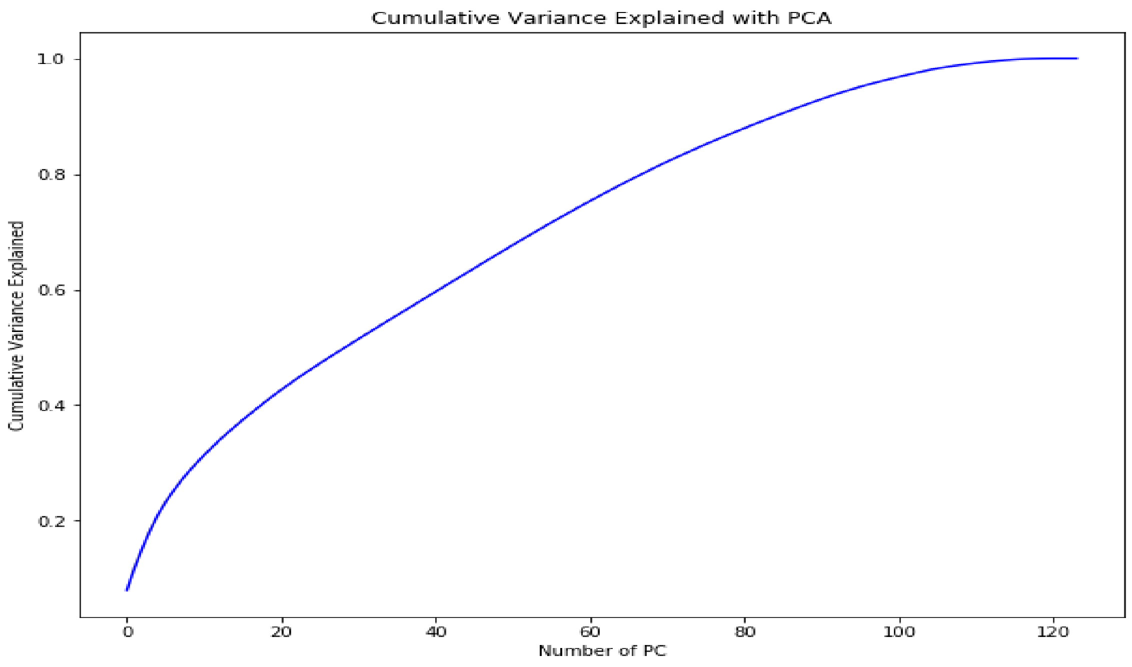


Figure 12: Cumulative variance explained with number of PCs

4.2.3 Correlation Analysis

Identifying highly correlated features is essential for eliminating redundant information. From the correlation heatmap, highly correlated features shown in warm colors lies in the bottom right corner conspicuously. Collinear variables are harmful for model interpretability, but blindly remove the majority may lead us to the other extreme. Consequently, we establish a threshold of 0.8. One out of any pair of the features will be removed if their correlation is higher than 0.8.

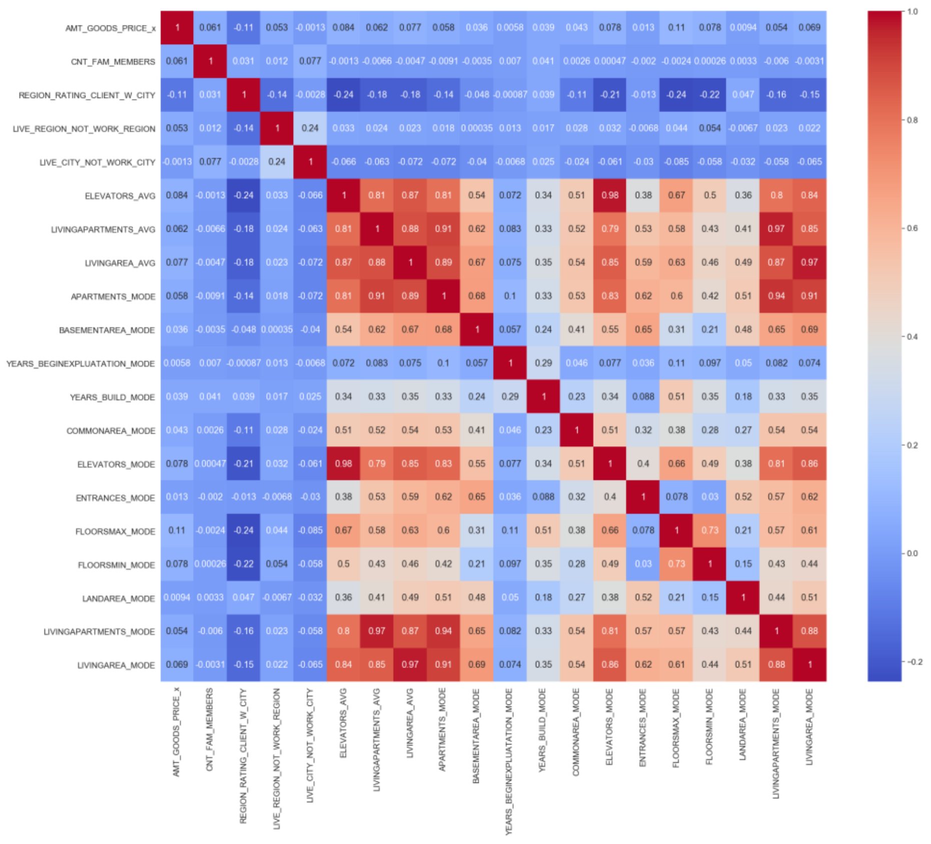


Figure 13: Correlation heatmap

5. Machine Learning Model Application

After previous steps, machine learning models can now be applied elegantly to build classifiers for prediction. Several models, including Naïve Bayes, logistic regression, random forest and LightGBM, are trained and improved in an organized way. Firstly, two basic models, Naïve Bayes and logistic regression are developed as our baseline model. Popular ensemble methods like bagging and Adaboost are then applied to each baseline model to look for any possible improvement. More sophisticated ensemble learning algorithm such as random forest and LightGBM are also tried to achieve better performance. Voting and stacking are finally used to integrate predictions from different classifiers. The following structure diagram summarize the model construction and evolution process in this project.

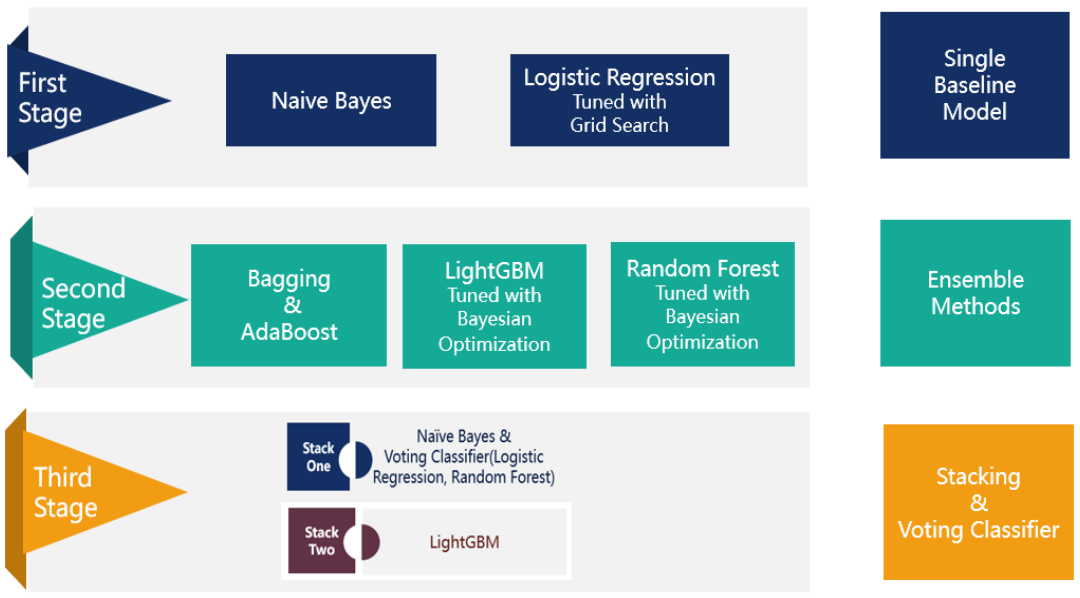

First Stage
Naive Bayes
Logistic Regression Tuned with Grid Search
Single Baseline Model
Second Stage
Bagging & AdaBoost
LightGBM Tuned with Bayesian Optimization
Random Forest Tuned with Bayesian Optimization
Ensemble Methods
Third Stage
Stack One
Naïve Bayes & Voting Classifier(Logistic Regression, Random Forest)
Stack Two
LightGBM
Stacking & Voting Classifier

Figure 14: Model Evolution Process

Hyperparameters are like the setting of an algorithm because they cannot be learned during model training. Fine tuning the hyperparameters can avoid underfitting or overfitting and thus optimize performance. Automated tuning methods like grid search and Bayesian optimization are implemented to find out the best configuration of our models.

The proper choice of model evaluation metrics is crucial for model comparison. Instead of solely relying on ROC AUC, more detailed performance analysis is conducted by looking into precision-recall curve. It is worth noting that blindly focusing on ROC AUC as the performance metric will bring on deception which conceals the truth of “garbage in, garbage out”, especially for imbalanced dataset. ROC AUC is unaffected by skew and very different precision-recall can be associated with same ROC AUC. That’s why we introduce PR AUC which can provide a more comprehensive view of our actual model performance.

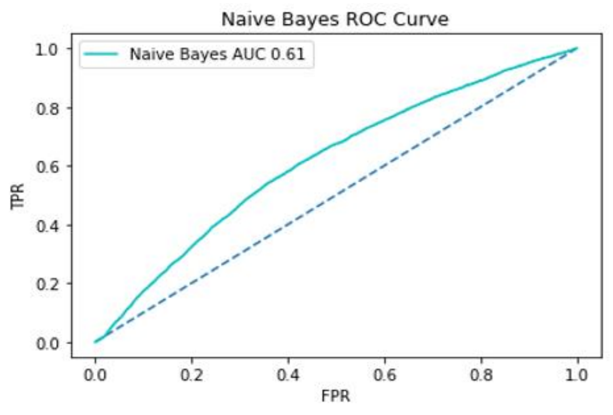
5.1 Naïve Bayes

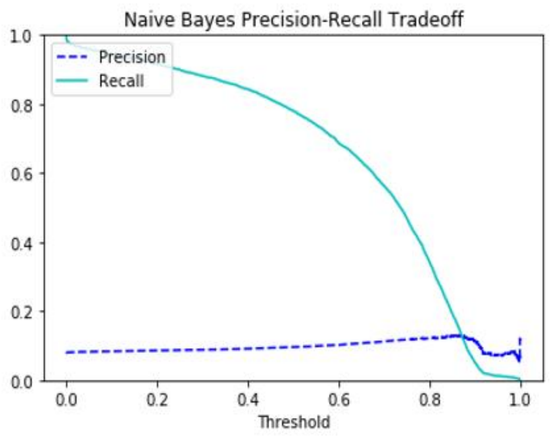
Naïve Bayes classifier calculate the posterior probability based on Bayes’ theorem with the independence assumptions between predictors. No complicated hyperparameter need to be tuned makes it simple and useful for large dataset.

The following figures show the results of model performance of Naïve Bayes. Without ensemble methods, Naïve Bayes model only achieve ROC AUC of 0.61. The recall reported in the default 0.5 threshold is 0.78, but the corresponding precision is only 0.1. After bagging and boosting, the precision-recall curve still lies in the bottom of the chart. PR AUC is only 0.11, so Naïve Bayes cannot be considered as a good classifier based on this metric.

Performance Metrics	
Accuracy	0.40
Precision	0.10
Recall	0.78
F1 Score	0.17
AUC	0.61

Confusion Matrix	
20746	35898
1066	3793





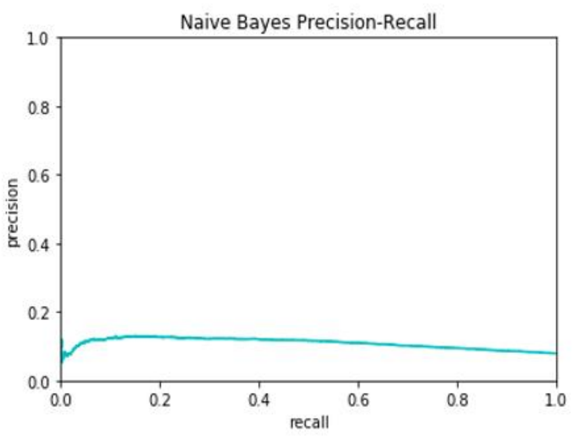
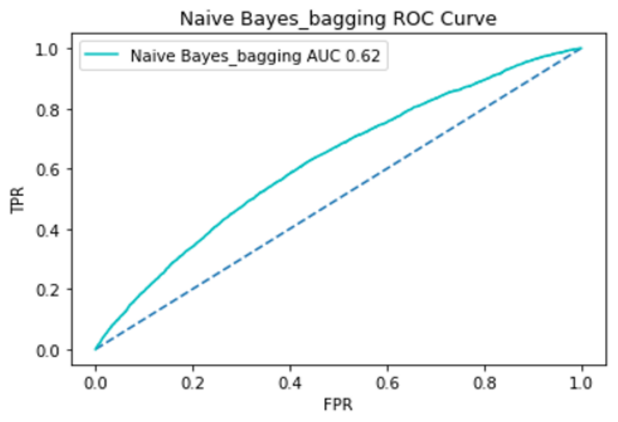
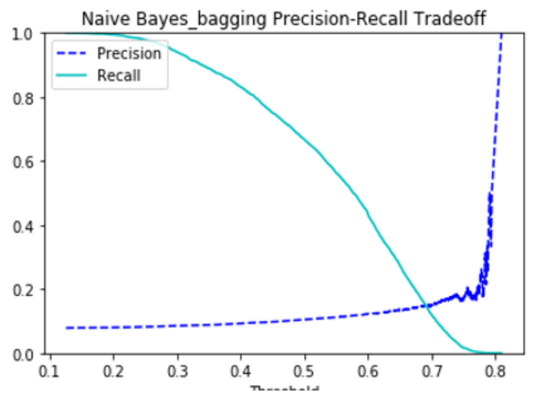


Figure 15: Model Performance of Naïve Bayes

Performance Metrics	
Accuracy	0.52
Precision	0.10
Recall	0.66
F1 Score	0.18
AUC	0.62

Confusion Matrix	
28936	27708
1611	3248





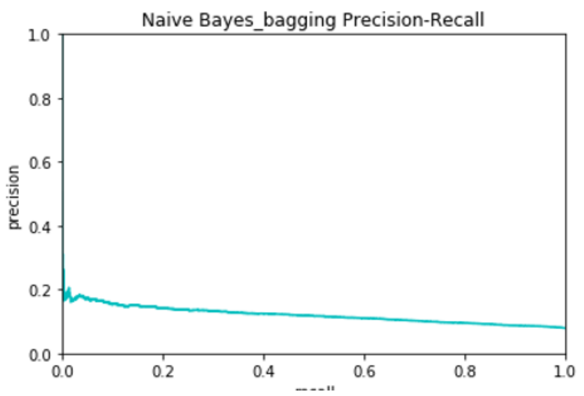
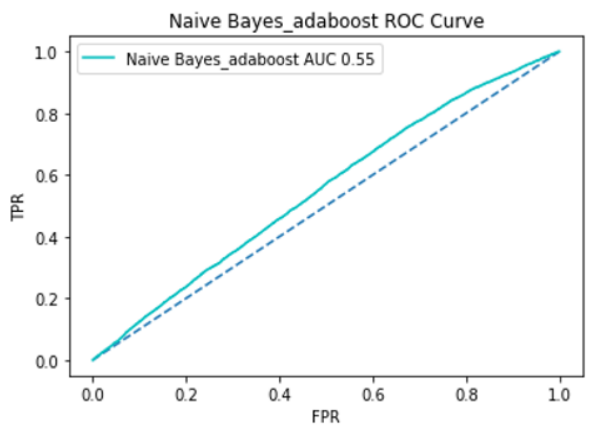


Figure 16: Model Performance of Naïve Bayes with baggi

Performance Metrics	
Accuracy	0.62
Precision	0.09
Recall	0.41
F1 Score	0.15
AUC	0.54

Confusion Matrix	
36199	20445
2848	2011



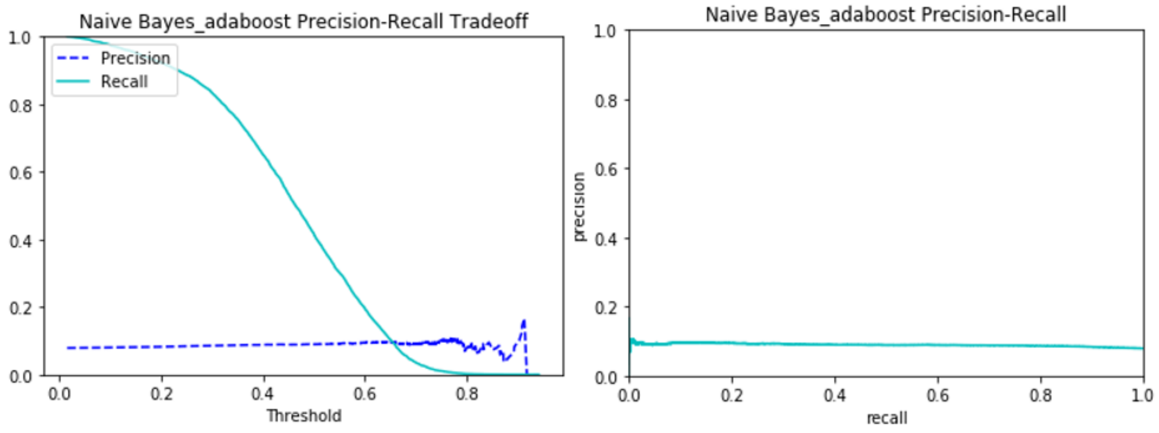


Figure 17: Model Performance of Naïve Bayes with Adaboost

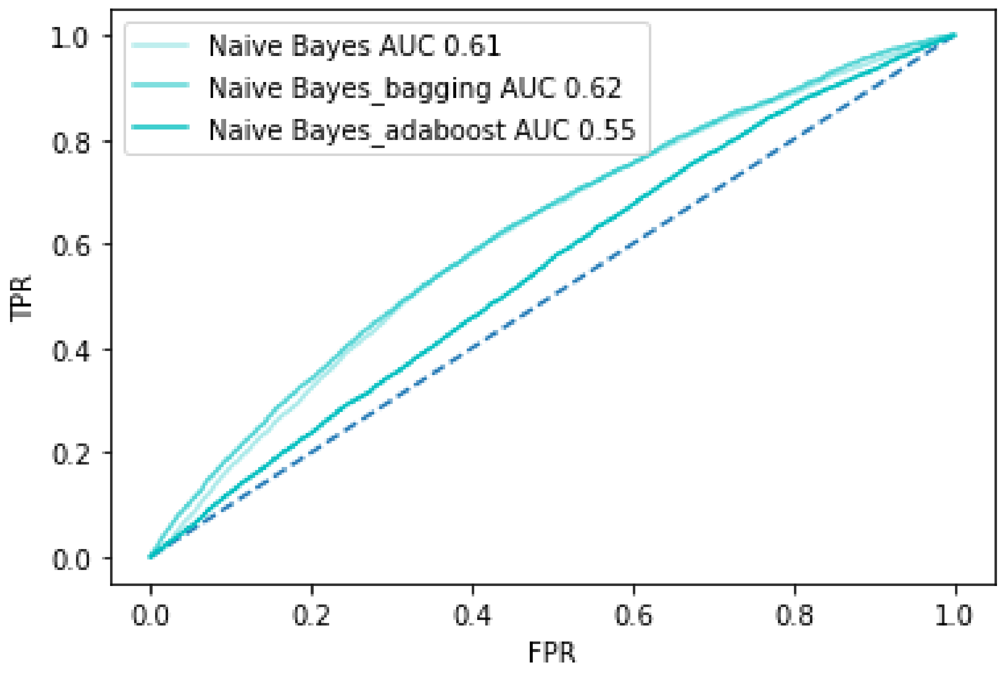


Figure 18: Model Performance Comparison of Naïve Bayes

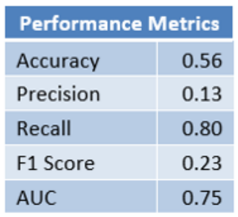
5.2 Logistic Regression

Logistic regression is also suitable for this binary classification problem. It is like a linear regression model but outputs the estimated probability formulated as below.

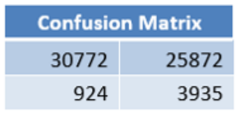
$$\hat{p} = h_\theta(x) = \sigma(\theta^T \cdot x) \text{ and}$$
$$\sigma(t) = \frac{1}{1 + \exp(-t)}$$

In the logistic regression algorithm, the hyperparameter "C" is the inverse of regularization strength. The smaller value specifies stronger regularization. We tune this hyperparameter by grid search and get best C equals to 0.1 for scoring ROC AUC. To further boost our model performance, bagging and Adaboost are applied sequentially. To avoid potential overfitting problem which may cause model performs worse on the test set, we carefully not choose aggressively large value for parameter "n_estimators" in both ensemble methods and "learning_rate" in Adaboost.

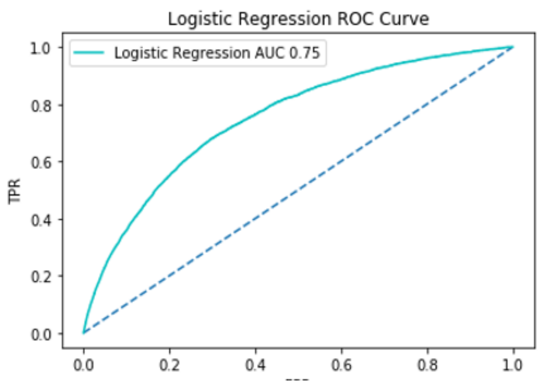
As indicated by the below figures, compared with Naive Bayes model, logistic regression performs better in terms of ROC AUC and PR AUC. ROC AUC improves from 0.61 to 0.75 and PR AUC increases to 0.21. The ROC AUC among original logistic regression model doesn't seem to be much difference. However, from the precision-recall curve, the recall from logistic regression with Adaboost drops sharply, while the one from logistic regression with bagging decline much gently. Therefore, logistic regression with bagging performs better because we are more likely to find threshold which give us a high recall with a little sacrifice for precision.

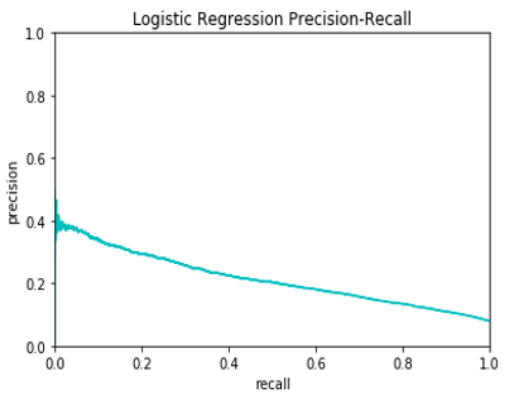


Performance Metrics	
Accuracy	0.56
Precision	0.13
Recall	0.80
F1 Score	0.23
AUC	0.75



Confusion Matrix	
30772	25872
924	3935





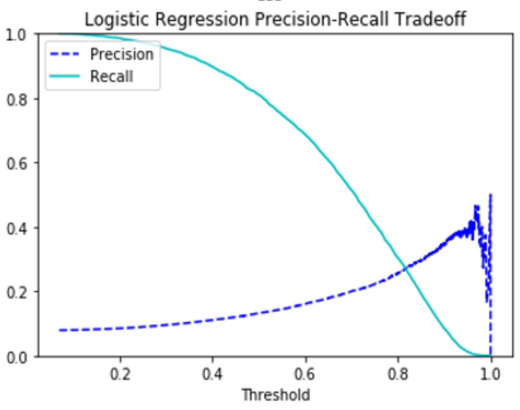
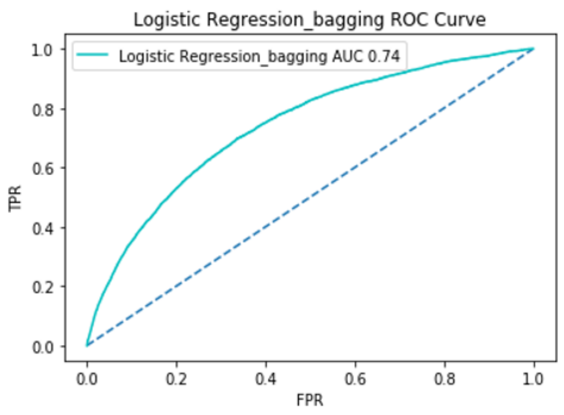
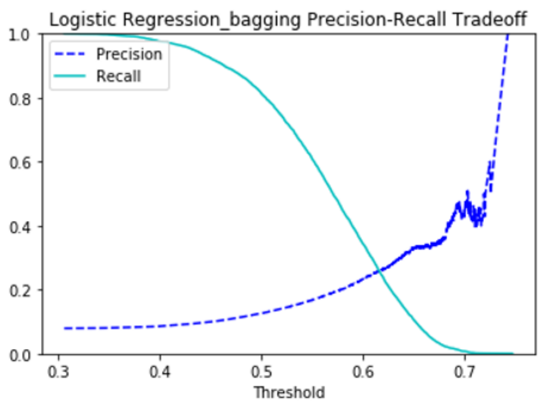


Figure 19: Model Performance of Logistic Regression

Performance Metrics	
Accuracy	0.54
Precision	0.13
Recall	0.81
F1 Score	0.21
AUC	0.74

Confusion Matrix	
29052	27592
897	3962





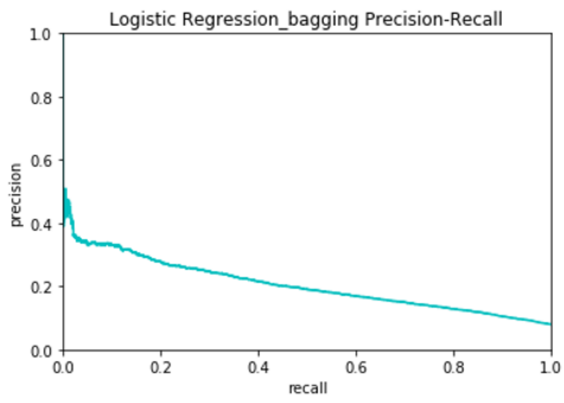
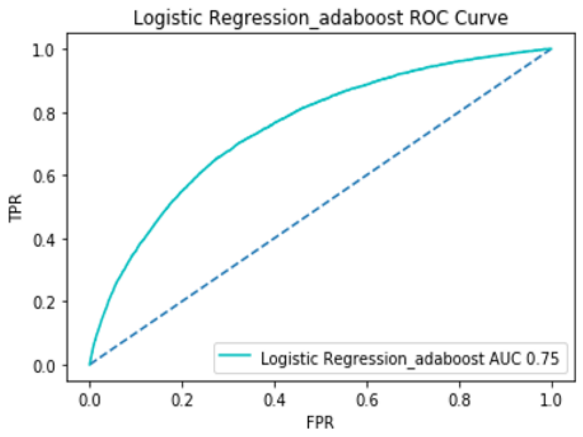
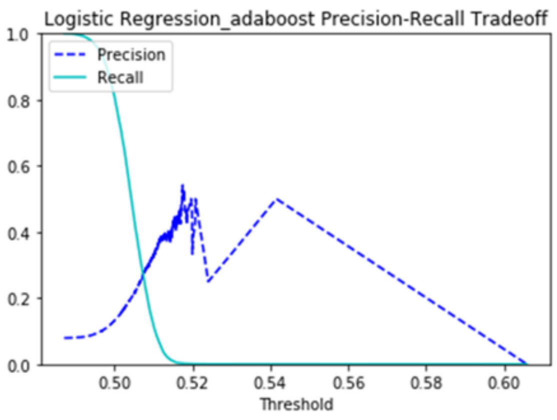


Figure 20: Model Performance of Logistic Regression with bagging

Performance Metrics	
Accuracy	0.56
Precision	0.13
Recall	0.80
F1 Score	0.22
AUC	0.75

Confusion Matrix	
30789	25855
927	3932





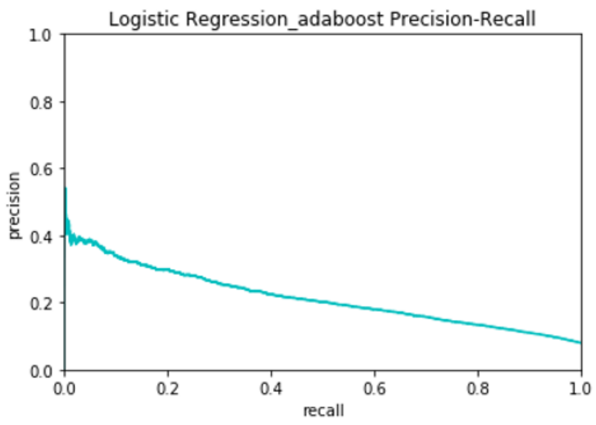


Figure 21: Model Performance of Logistic Regression with Adaboost

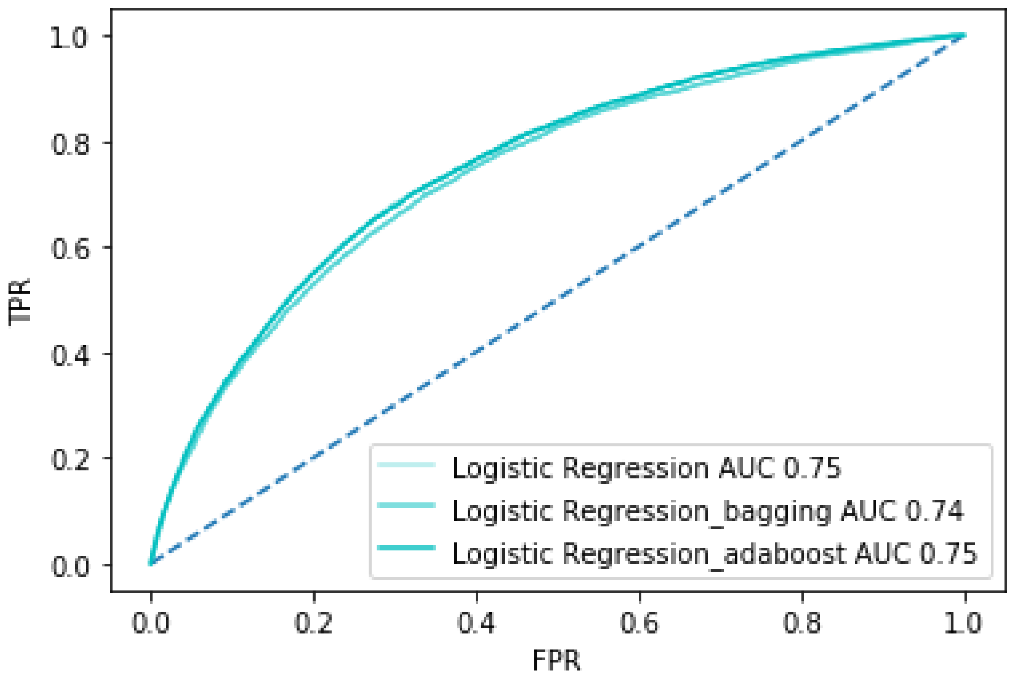


Figure 22: Model Performance Comparison of Logistic Regression

5.3 Random Forest

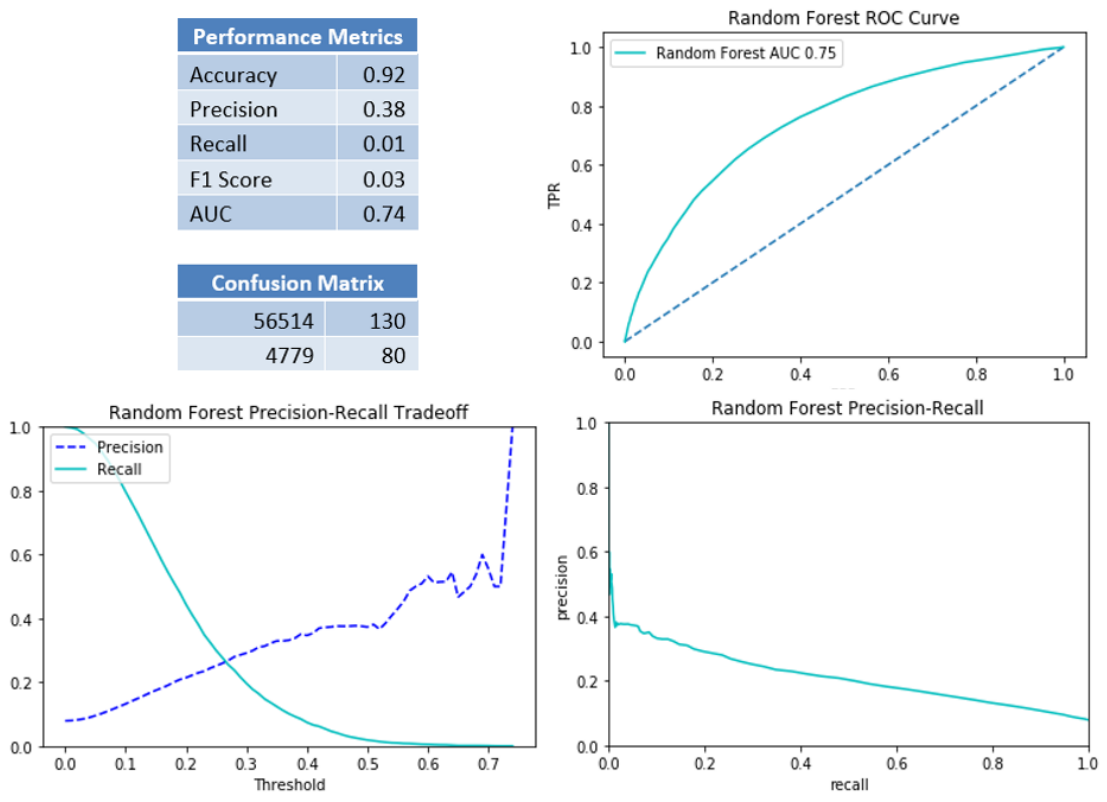
Random forest is just an ensemble of decision tree trained via bagging method. This algorithm introduces extra randomness when growing trees. It searches for the best features among a random subset of features, instead of searching for the very best feature when splitting a node.

Previously for logistic regression, grid search is used. But since grid search is a brutal way of finding the optimal parameters which trains and tests every possible combination, for this more sophisticated model, we use Bayesian optimization which learns from the past evaluation and takes less computation time. An objective scoring function and the range of the value set for hyperparameters are defined for Bayesian optimization to determine the best parameters within 5-fold cross validation. To better control how trees are grown and avoid overfitting, the below hyperparameters are tuned. After tuning, the best set of hyperparameters we get are reported in the table.

Hyperparameter	Range	Result
n_estimators	(0, 50)	41
max_depth	(1, 10)	9
min_samples_split	(2,10)	8
min_samples_leaf	(2,10)	6

Table 4: List of hyperparameters tuned in random forest

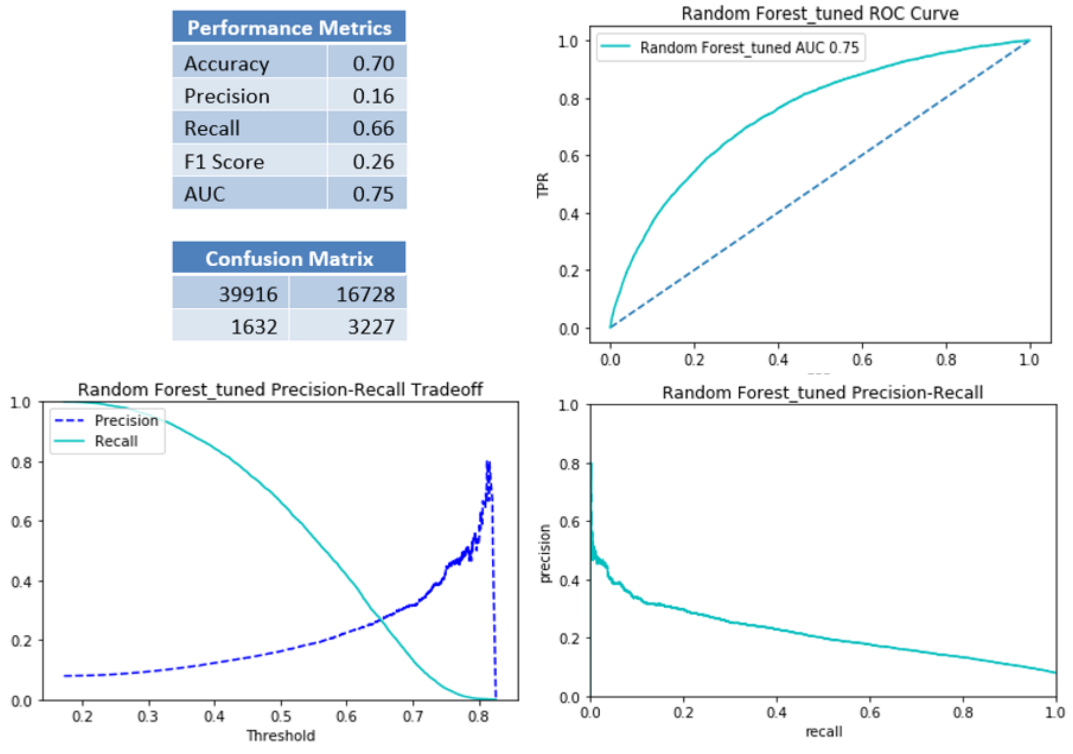
Based on the following model performance analysis, ROC AUC and PR AUC reaches to 0.75 and 0.22 respectively. Interestingly, after hyperparameter tuning, the recall improves dramatically from nearly 0 to 0.66 when the threshold is 0.5. The certain threshold that recall is lower than precision moves far towards larger value beyond 0.5, meaning that we can get a wider range of threshold which makes recall outperforms precision. A model with pretty low recall is useless, because we care more about recall than precision for default prediction. In this view, the tuned model can be regarded as a better classifier.



Performance Metrics	
Accuracy	0.92
Precision	0.38
Recall	0.01
F1 Score	0.03
AUC	0.74

Confusion Matrix	
56514	130
4779	80

Figure 23: Model Performance of Random Forest



Performance Metrics	
Accuracy	0.70
Precision	0.16
Recall	0.66
F1 Score	0.26
AUC	0.75

Confusion Matrix	
39916	16728
1632	3227

Figure 24: Model Performance of Random Forest After Hyperparameter Tuning

5.4 LightGBM

LightGBM is also a tree-based machine learning algorithm under a gradient boosting framework. Different from random forest, LightGBM grows tree vertically, meaning that LightGBM grows trees leaf-wise choosing the leaf with max delta loss to grow. Since LightGBM is efficient and takes lower memory to run, LightGBM is an ideal choice for large dataset with less concern about overfitting.

Bayesian optimization is also applied for hyperparameter tuning in LightGBM. The list of hyperparmeter we tuned are listed in the below table. 'feature_fraction', 'bagging_fraction', 'lambda_l1' and 'lambda_l2' are for learning control, while 'max_depth', 'min_split_gain', ' min_child_weight ' and ' num_leaves ' are for tree grown.

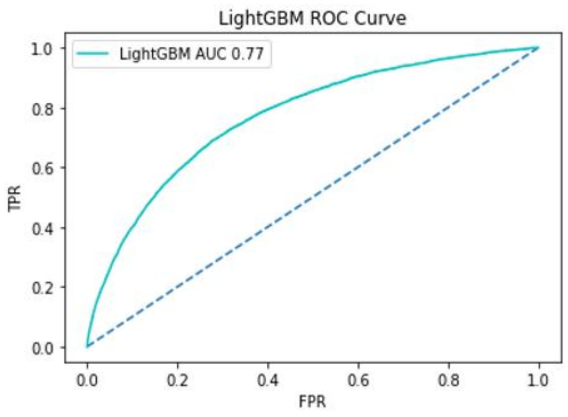
Hyperparameter	Range	Result
feature_fraction	(0.1, 0.9)	0.67
bagging_fraction	(0.8, 1)	0.9
lambda_l1	(0, 5)	3.01
lambda_l2	(0, 3)	1.63
max_depth	(5, 9)	7
min_split_gain	(0.001, 0.1)	0.044
min_child_weight	(5, 50)	34
num_leaves	(24, 45)	43

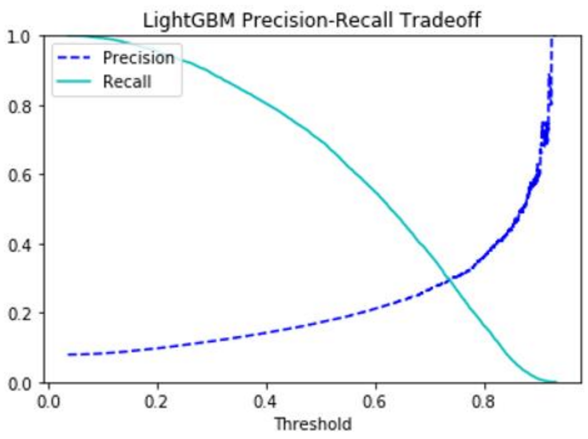
Table 5: List of hyperparameters tuned in LightGBM

LightGBM with bagging performs best with ROC AUC of 0.77 and PR AUC of 0.25. Unlike random forest, the recall and precision are stable before and after the model is tuned. At threshold of 0.5, the recall is around 0.7, which is fine for default prediction. Since we focus more on recall, LightGBM is a reasonable choice for default prediction. The detailed model performance results are shown as below.

Performance Metrics	
Accuracy	0.71
Precision	0.17
Recall	0.70
F1 Score	0.27
AUC	0.77

Confusion Matrix	
40330	16314
1468	3391





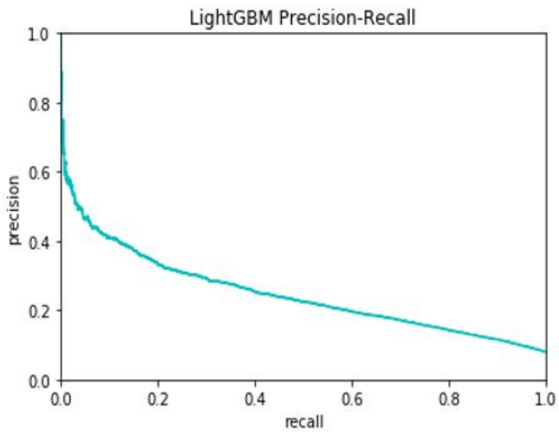
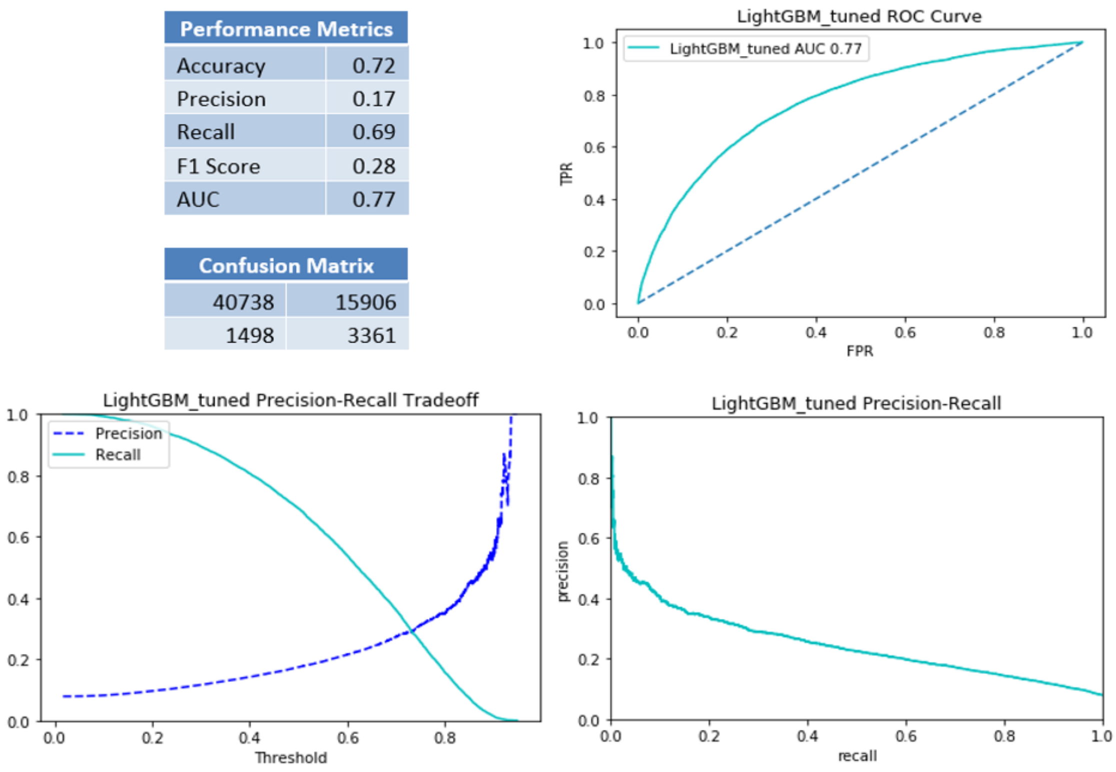


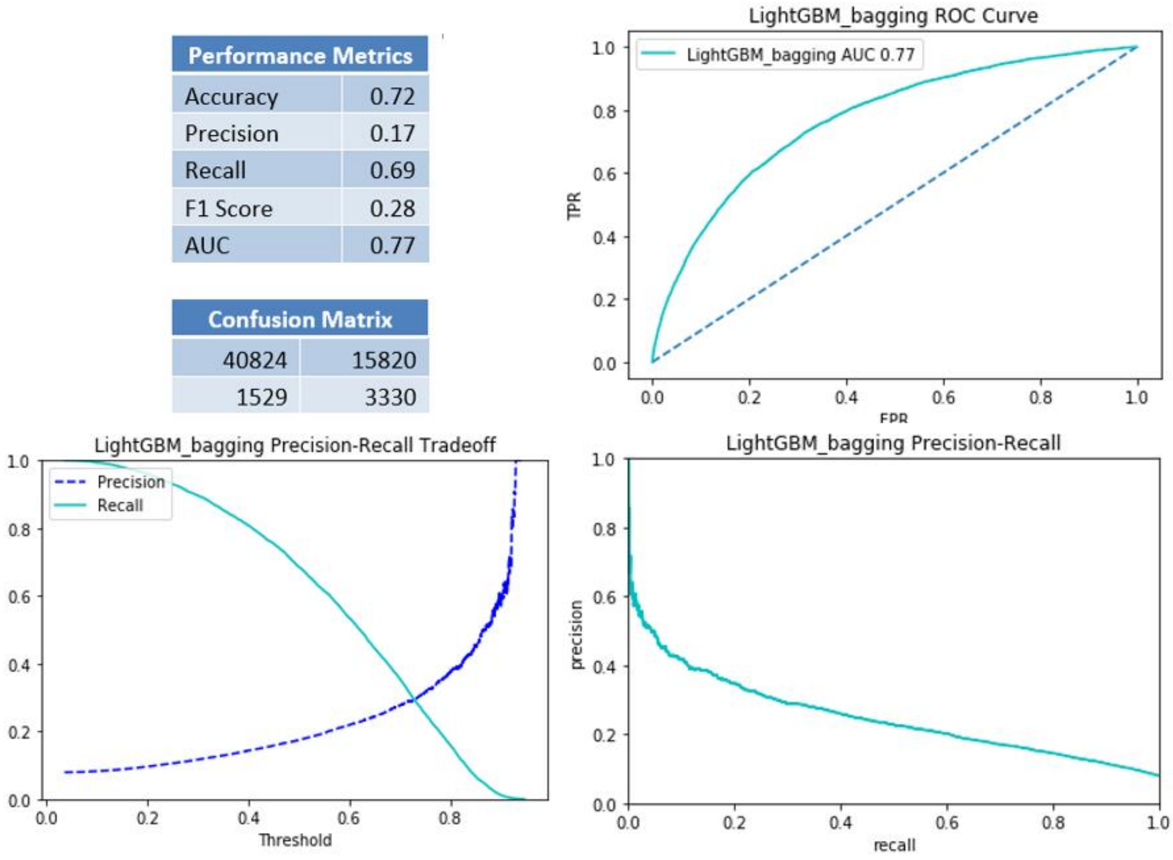
Figure 25: Model Performance of LightGBM



Performance Metrics	
Accuracy	0.72
Precision	0.17
Recall	0.69
F1 Score	0.28
AUC	0.77

Confusion Matrix	
40738	15906
1498	3361

Figure 26: Model Performance of LightGBM After Hyperparameter Tuning



Performance Metrics	
Accuracy	0.72
Precision	0.17
Recall	0.69
F1 Score	0.28
AUC	0.77

Confusion Matrix	
40824	15820
1529	3330

Figure 27: Model Performance of LightGBM with bagging

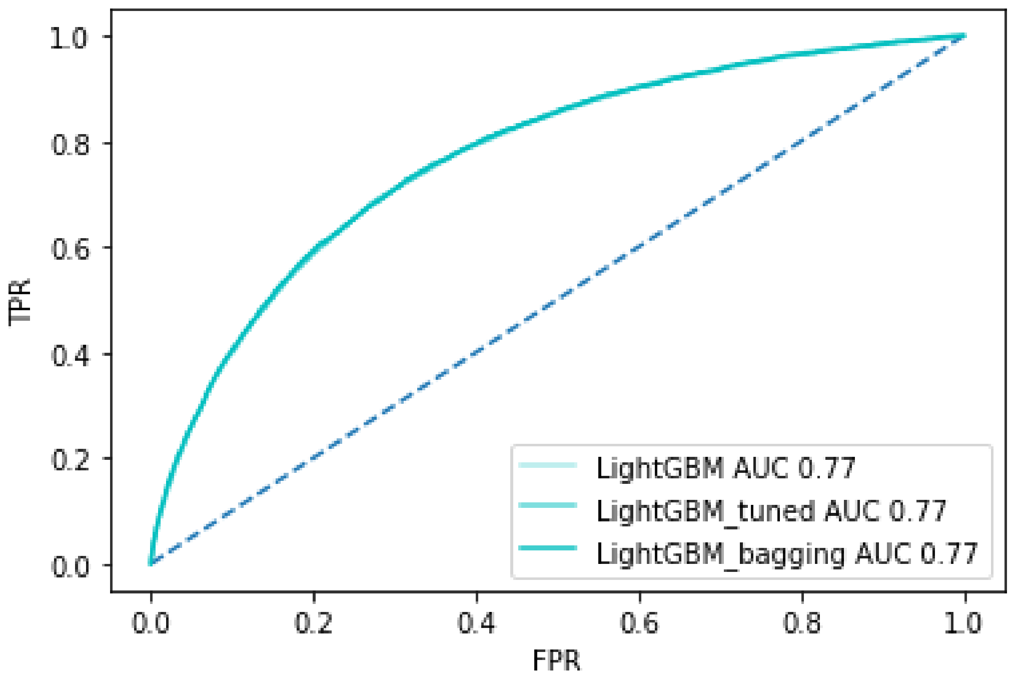


Figure 28: Model Comparison of LightGBM

5.5 Voting and stacking Classifier

Taking the advantage of aggregating the predictions of each classifier and predict the class that gets the most votes, voting classifier is strong learner. Instead of using trivial function like hard voting, stacking is an ensemble which attempts to enhance model performance by using out-of-fold predictions as new input features. In this final stage, we build a voting and stacking blended model. See the model structure as follows. The first stack contains Naïve Bayes and hard-voting classifier in which logistic regression and random forest are combined. LightGBM is used for the second stack. The hyperparameters set in this classifier are the preceding optimized ones.

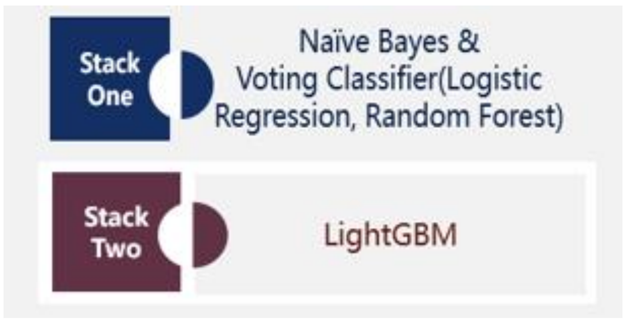
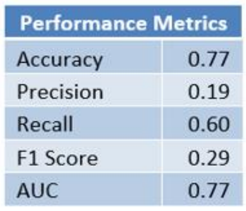
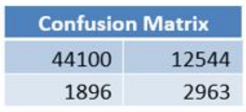

Stack One
Naïve Bayes & Voting Classifier(Logistic Regression, Random Forest)
Stack Two
LightGBM

Figure 29: The Structure of Voting and Stacking Blended Model

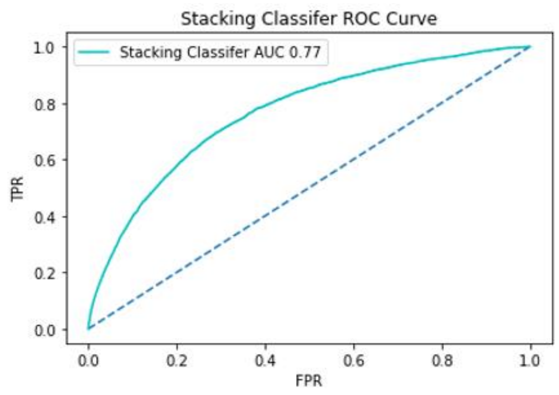
At the threshold of 0.5, we can get recall of 0.6 and precision of 0.19 from voting and stacking blended model. With ROC AUC of 0.77 and PR AUC of 0.24, this classifier can make a relatively good default detection. But compared with LightGBM bagging model, it performs slightly worse in terms of PR AUC score.

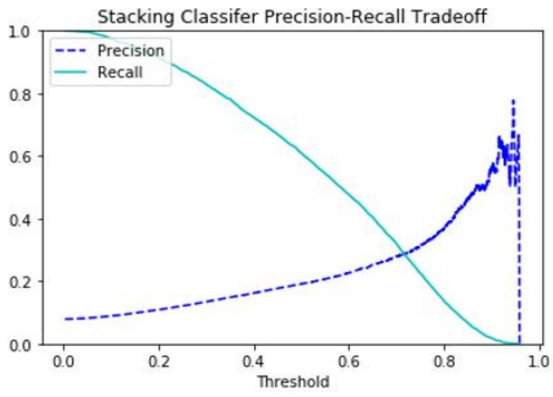


Performance Metrics	
Accuracy	0.77
Precision	0.19
Recall	0.60
F1 Score	0.29
AUC	0.77



Confusion Matrix	
44100	12544
1896	2963





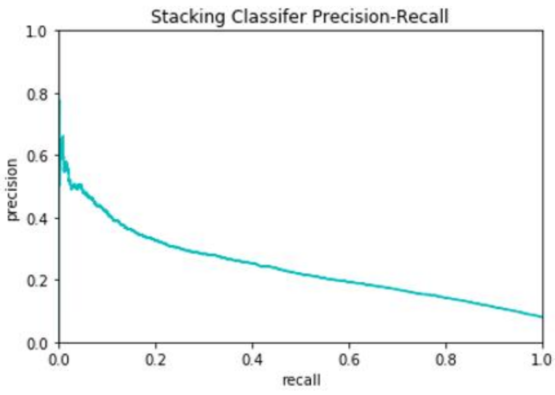


Figure 30: Model Performance of Voting and Stacking Classifier

6. Conclusion

In this project, we employ an extremely large dataset consisting of loan applicants' personal information and previous loan application records to build and test machine learning algorithms for credit card delinquency. After prudent data preprocessing and feature engineering, we find that except for Naïve Bayes, logistic regression, random forest, LightGBM and voting and stacking blended model perform at about the same level if measured by ROC AUC. But if we look further into precision-recall curve and PR AUC, LightGBM bagging outperforms all the others.

Although delinquency prediction is merely binary classification, as is usually the case, the dataset is imbalanced. Therefore, we are prone to build an actually worthless classifier with low ability of detecting the positive class or with precision and recall at opposite ends. In practice, reasonable treatment of imbalanced dataset and selection of performance metrics are two key factors of a good classifier. Firstly, resampling strategy or fine-tuning class weights in sophisticated models must be implemented. Secondly, we should bear in mind that despite a decent ROC AUC score, recall-precision curve may perform terribly. The plot of recall-precision curve helps us to decide a threshold for precision and recall trade-off. PR AUC is more sensitive of skewed class distribution, so it would be a better performance metric.

Among all the models we constructed, the precision curve for some of them is much bumpier than the others, suggesting that precision is volatile as threshold changes. When facing with this situation, it is somewhat more challenging for us to select a good precision-recall tradeoff. Therefore, we can further explore machine-learning models with other ensemble methods to achieve good a precision-recall profile and higher PR AUC. In-depth feature engineering and refined hyperparameter tuning can also be conducted for further improvement.

Appendix

Descriptions of variables used as model inputs

Number	Features	Description
1	NAME_CONTRACT_TYPE	Identification if loan is cash or revolving
2	CODE_GENDER	Gender of the client
3	FLAG_OWN_CAR	Flag if the client owns a car
4	FLAG_OWN_REALTY	Flag if client owns a house or flat
5	CNT_CHILDREN	Number of children the client has
6	AMT_INCOME_TOTAL	Income of the client
7	AMT_CREDIT	Credit amount of the loan
8	AMT_ANNUITY	Loan annuity
9	NAME_TYPE_SUITE	Who was accompanying client when he was applying for the loan
10	NAME_INCOME_TYPE	Clients income type (businessman, working, maternity leave,?)
11	NAME_EDUCATION_TYPE	Level of highest education the client achieved
12	NAME_FAMILY_STATUS	Family status of the client
13	NAME_HOUSING_TYPE	What is the housing situation of the client (renting, living with parents, ...)
14	REGION_POPULATION_RELATIVE	Normalized population of region where client lives (higher number means the client lives in more populated region)
15	DAYS_BIRTH	Client's age in days at the time of application
16	DAYS_EMPLOYED	How many days before the application the person started current employment
17	DAYS_REGISTRATION	How many days before the application did client change his registration
18	DAYS_ID_PUBLISH	How many days before the application did client change the identity document with which he applied for the loan
19	OWN_CAR_AGE	Age of client's car
20	FLAG_MOBIL	Did client provide mobile phone (1=YES, 0=NO)
21	FLAG_EMP_PHONE	Did client provide work phone (1=YES, 0=NO)
22	FLAG_WORK_PHONE	Did client provide home phone (1=YES, 0=NO)
23	FLAG_CONT_MOBILE	Was mobile phone reachable (1=YES, 0=NO)
24	FLAG_PHONE	Did client provide home phone (1=YES, 0=NO)
25	FLAG_EMAIL	Did client provide email (1=YES, 0=NO)

26	OCCUPATION_TYPE	What kind of occupation does the client have
27	REGION_RATING_CLIENT	Our rating of the region where client lives (1,2,3)
28	WEEKDAY_APPR_PROCESS_START	On which day of the week did the client apply for the loan
29	HOUR_APPR_PROCESS_START	Approximately at what hour did the client apply for the loan
30	REG_REGION_NOT_LIVE_REGION	Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)
31	REG_REGION_NOT_WORK_REGION	Flag if client's permanent address does not match work address (1=different, 0=same, at region level)
32	REG_CITY_NOT_LIVE_CITY	Flag if client's permanent address does not match contact address (1=different, 0=same, at city level)
33	REG_CITY_NOT_WORK_CITY	Flag if client's permanent address does not match work address (1=different, 0=same, at city level)
34	ORGANIZATION_TYPE	Type of organization where client works
35	EXT_SOURCE_1	Normalized score from external data source
36	EXT_SOURCE_2	Normalized score from external data source
37	EXT_SOURCE_3	Normalized score from external data source
38	APARTMENTS_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
39	BASEMENTAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
40	YEARS_BEGINEXPLUATATION_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
41	YEARS_BUILD_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor

42	COMMONAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
43	ENTRANCES_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
44	FLOORSMAX_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
45	FLOORSMIN_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
46	LANDAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
47	NONLIVINGAPARTMENTS_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
48	NONLIVINGAREA_AVG	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor

49	FONDKAPREMONT_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
50	HOUSETYPE_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
51	WALLSMATERIAL_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
52	EMERGENCYSTATE_MODE	Normalized information about building where the client lives, What is average (_AVG suffix), modus (_MODE suffix), median (_MEDI suffix) apartment size, common area, living area, age of building, number of elevators, number of entrances, state of the building, number of floor
53	OBS_30_CNT_SOCIAL_CIRCLE	How many observation of client's social surroundings with observable 30 DPD (days past due) default
54	DEF_30_CNT_SOCIAL_CIRCLE	How many observation of client's social surroundings defaulted on 30 DPD (days past due)
55	DAYS_LAST_PHONE_CHANGE	How many days before application did client change phone
56	FLAG_DOCUMENT_2	Did client provide document 2
57	FLAG_DOCUMENT_3	Did client provide document 3
58	FLAG_DOCUMENT_4	Did client provide document 4
59	FLAG_DOCUMENT_5	Did client provide document 5
60	FLAG_DOCUMENT_6	Did client provide document 6
61	FLAG_DOCUMENT_7	Did client provide document 7
62	FLAG_DOCUMENT_8	Did client provide document 8
63	FLAG_DOCUMENT_9	Did client provide document 9
64	FLAG_DOCUMENT_10	Did client provide document 10
65	FLAG_DOCUMENT_11	Did client provide document 11
66	FLAG_DOCUMENT_12	Did client provide document 12
67	FLAG_DOCUMENT_13	Did client provide document 13

68	FLAG_DOCUMENT_14	Did client provide document 14
69	FLAG_DOCUMENT_15	Did client provide document 15
70	FLAG_DOCUMENT_16	Did client provide document 16
71	AMT_REQ_CREDIT_BUREAU_HOUR	Number of enquiries to Credit Bureau about the client one hour before application
72	AMT_REQ_CREDIT_BUREAU_DAY	Number of enquiries to Credit Bureau about the client one day before application (excluding one hour before application)
73	AMT_REQ_CREDIT_BUREAU_WEEK	Number of enquiries to Credit Bureau about the client one week before application (excluding one day before application)
74	AMT_REQ_CREDIT_BUREAU_MON	Number of enquiries to Credit Bureau about the client one month before application (excluding one week before application)
75	AMT_REQ_CREDIT_BUREAU_QRT	Number of enquiries to Credit Bureau about the client 3 month before application (excluding one month before application)
76	AMT_REQ_CREDIT_BUREAU_YEAR	Number of enquiries to Credit Bureau about the client one day year (excluding last 3 months before application)
77	AMT_ANNUITY	Annuity of the Credit Bureau credit
78	CNT_INSTALMENT	Term of previous credit (can change over time)
79	CNT_INSTALMENT_FUTURE	Installments left to pay on the previous credit
80	NAME_CONTRACT_STATUS	Contract status during the month
81	SK_DPD	DPD (days past due) during the month of previous credit
82	SK_DPD_DEF	DPD during the month with tolerance (debts with low loan amounts are ignored) of the previous credit
83	NAME_CONTRACT_STATUS	Contract status (active signed,...) on the previous credit
84	SK_DPD	DPD (Days past due) during the month on the previous credit
85	SK_DPD_DEF	DPD (Days past due) during the month with tolerance (debts with low loan amounts are ignored) of the previous credit
86	NAME_CONTRACT_TYPE	Contract product type (Cash loan, consumer loan [POS] ,...) of the previous application
87	AMT_ANNUITY	Annuity of previous application
88	AMT_APPLICATION	For how much credit did client ask on the previous application

89	AMT_CREDIT	Final credit amount on the previous application. This differs from AMT_APPLICATION in a way that the AMT_APPLICATION is the amount for which the client initially applied for, but during our approval process he could have received different amount - AMT_CREDIT
90	AMT_DOWN_PAYMENT	Down payment on the previous application
91	WEEKDAY_APPR_PROCESS_START	On which day of the week did the client apply for previous application
92	HOUR_APPR_PROCESS_START	Approximately at what day hour did the client apply for the previous application
93	FLAG_LAST_APPL_PER_CONTRACT	Flag if it was last application for the previous contract. Sometimes by mistake of client or our clerk there could be more applications for one single contract
94	NFLAG_LAST_APPL_IN_DAY	Flag if the application was the last application per day of the client. Sometimes clients apply for more applications a day. Rarely it could also be error in our system that one application is in the database twice
95	RATE_DOWN_PAYMENT	Down payment rate normalized on previous credit
96	NAME_CASH_LOAN_PURPOSE	Purpose of the cash loan
97	NAME_CONTRACT_STATUS	Contract status (approved, cancelled, ...) of previous application
98	NAME_PAYMENT_TYPE	Payment method that client chose to pay for the previous application
99	CODE_REJECT_REASON	Why was the previous application rejected
100	NAME_TYPE_SUITE	Who accompanied client when applying for the previous application
101	NAME_CLIENT_TYPE	Was the client old or new client when applying for the previous application
102	NAME_GOODS_CATEGORY	What kind of goods did the client apply for in the previous application
103	NAME_PORTFOLIO	Was the previous application for CASH, POS, CAR, ?
104	NAME_PRODUCT_TYPE	Was the previous application x-sell o walk-in
105	CHANNEL_TYPE	Through which channel we acquired the client on the previous application
106	SELLERPLACE_AREA	Selling area of seller place of the previous application
107	NAME_SELLER_INDUSTRY	The industry of the seller
108	CNT_PAYMENT	Term of previous credit at application of the previous application
109	NAME_YIELD_GROUP	Grouped interest rate into small medium and high

		of the previous application
110	PRODUCT_COMBINATION	Detailed product combination of the previous application
111	DAYS_FIRST_DUE	Relative to application date of current application when was the first due supposed to be of the previous application
112	NFLAG_INSURED_ON_APPROVAL	Did the client requested insurance during the previous application
113	NUM_INSTALMENT_VERSION	Version of installment calendar (0 is for credit card) of previous credit. Change of installment version from month to month signifies that some parameter of payment calendar has changed
114	NUM_INSTALMENT_NUMBER	On which installment we observe payment
115	AMT_PAYMENT	What the client actually paid on previous credit on this installment
116	EXT_SOURCE_2 EXT_SOURCE_3'	EXT_SOURCE_2 EXT_SOURCE_3
117	Bureau_debt_credit_ratio	Debt Amt/ Total Credit Amt
118	Bureau_overdue_debt_ratio	Overdue Amt/Total Credit Amt
119	Credit_application_ratio	Application Amt /Total Credit Amt
120	Days_overdue	Days Enter Payment — Instalment Day
121	Draw_limit_ratio	Current Drawing Amt/ Credit Drawing limit
122	Installment_ratio	Installment Amt /Payment Amt
123	Payment_install_ratio	Installment Amt /Payment Amt
124	Special_expense	Past average drawings — (current drawings+ other drawings)

References

- Florentin Butaru, Qingqing Chen, Brian Clark, Sanmay Das, Andrew W. Lo, Akhtar Siddique, Risk and risk management in the credit card industry, Journal of Banking & Finance, Volume 72,2016, Pages 218-239, ISSN 0378-4266
- L. A. Jeni, J. F. Cohn and F. De La Torre, "Facing Imbalanced Data--Recommendations for the Use of Performance Metrics," 2013 Humaine Association Conference on Affective Computing and Intelligent Interaction, Geneva, 2013, pp. 245-251, doi: 10.1109/ACII.2013.47.
- Reilly Meinert, Optimizing Hyperparameters in Random Forest Classification, https://towardsdatascience.com/optimizing-hyperparameters-in-random-forest-classification-ec7741f9d3f6
- Lightgbm document, https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMModel.html
- Lightgbm.cv https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.cv.html
- Michael Galarnyk, PCA using Python (scikit-learn), https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60
- VotingClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.

VotingClassifier.html

- model_selection.KFold ,https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html
- RandomForestClassifier, https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- over_sampling, https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.RandomOverSampler.html
- Jason Brownlee, Random Oversampling and Undersampling for Imbalanced Classification, https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/
- Jason Brownlee, How to Implement Bayesian Optimization from Scratch in Python, https://machinelearningmastery.com/what-is-bayesian-optimization/
- Pushkar Mandot, What is LightGBM, How to implement it? How to fine tune the parameters? https://medium.com/@pushkarmandot/https-medium-com-pushkarmandot-what-is-lightgbm-how-to-implement-it-how-to-fine-tune-the-parameters-60347819b7fc
- SHAROON SAXENA, A Beginner’s Guide to Random Forest Hyperparameter Tuning, https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/
- Aurélien Géron, Hands-On Machine Learning with Scikit-Learn and TensorFlow, http://oreilly.com/catalog/errata.csp?isbn=9781491962299
- Andreas C. Mueller and Sarah Guido, Introduction to Machine Learning with Python, http://oreilly.com/catalog/errata.csp?isbn=9781491917213