Credit Default Prediction Project Report

1. Background Introduction

The ubiquity of the use of credit cards correlates with an increased chance of payment default for the credit issuers; as such, being able to accurately monitor an ever increasing customer base is crucial. As a first step, we aim at creating a machine learning algorithm to predict the likelihood of customers defaulting based on the metrics of the user profile and banking history over a given month.

2. Dataset Description

We have tabular data about customer profiles at American Express, having 5531451 rows and 191 columns. Rows consist of 458913 unique customers with multiple records for every customer. The columns can be broken down into two ways: if classified by their functions, the features consist of date, delinquency variables(D_<N>), spend variable (S_<N>), payment variables(P_<N>), balance variables(B_<N>), and risk variables(R_<N>); if classified by the data type, the 191 features divide into 12 categorical data, 177 numerical data, 1 date data, and 1 target variable in this dataset. The actual default rate was too low; as such, they were upsampled so that the ratio of default and non-default labels is roughly 1:3 now.

3. Exploratory Data Analysis

In order to gain a clearer view of the data, we constructed violin plots for each variable. From Figure 1, negative(blue) indicates the customer does not pay the due amount in 120 days, positive otherwise. We can see that for most correlated features P_2 and D_48, P_2 negative customers tend to have a higher median. Its distribution is right-skewed while positive customers have roughly normal distribution. For D_48, negative customers tend to have lower median and it is right-skewed. For S_3, there is no certain tendency on what types of customers choose positive or negative. For B_2, it seems positive customers and negative customers have very different values. Most negative customers have higher B_2 while others have lower. Besides, Positive customers tend to have a normal distribution.

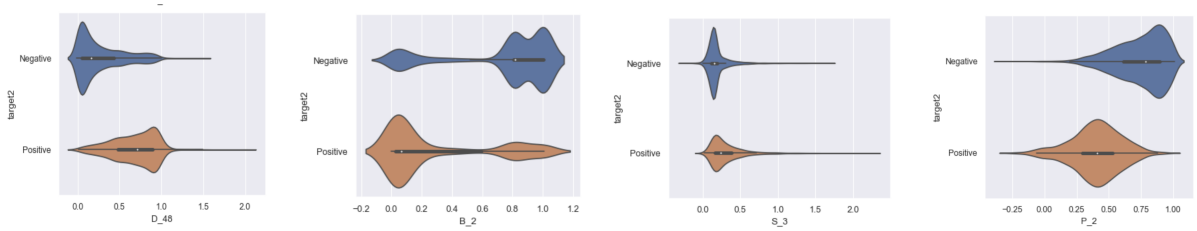

Figure 1 Violin Plots for Representative Variables

4. Machine Learning Techniques

a) Data Preprocessing

i). Data Compression & Feature Engineering

We compressed most “float64” data types into “float16” equivalents to bring down the size of the training set to 2.1 GB from 7.9 GB. We then began feature engineering by first generating abundant column-wise aggregation-based features. For each of the 177 numerical features of every customer record, we computed its 4 common statistics’-‘min’, ‘max’, ‘mean’, ‘std’, and last’, and the ‘difference’ between the last-seen and second-last-seen values of every feature; for each of the 13 categorical features, we computed its ‘count’, ‘last’, and ’nunique’. The step amounted to 458913 rows, each representing a customer, and 1101 features for each row. (1101 = 177*6 + 13*4).

ii). Data Cleaning & Transformation

We removed columns with 50% or more missing values to result in 945 features. For sporadic missing values we imputed it with the ‘mean’ of that column. We then performed scaling with StandardScaler() and stratified splitting with 8:2 ratio on the training set to break it into development and test sets.

iii). Feature Selection

We then aimed to select a subset of 300 ones from the 945 features. Our first approach was to use SelectKBest() that accounts for feature correlations through ‘fclass_if’ values; the second approach was to use logistic regression with L1 norm, which implicitly assumes linear relationships. We used forward and backward

search to evaluate the features. For each approach, we ran the 5-fold cross validation for each feature set and prioritized for higher precision.

b) Model Training

In this part, we built three models to predict the default rate of each customer_ID. Then we compared the evaluation metrics of each model and calibrated the best model out of the three.

Model 1: XGBoost

XGBoost, a.k.a. Extreme Gradient Boosting, is a parallelized and optimized version of the gradient boosting algorithm. It is featured with parallelization, non-linearity, and scalability, which applies to our large dataset encompassing millions of rows and massive features.

We applied the Bayesian Optimization method, a sequential design strategy for global optimization, using the Optuna library to fine-tune the hyperparameters for the XGBoost model. Due to the vast size of our dataset, we used a three-way holdout approach instead of cross-validation to save computation resources and time. We split the development dataset into training and validation datasets by 3:1 and defined “AUC” as our evaluation metric. We finally found the best hyperparameters as: eta = 0.056, gamma = 7.97, max_depth = 149, min_child_weight=0.397, and alpha=1.77.

Model 2: Random Forest

Differing from XGBoost, random forest is a bagging model. It handles the overfitting issue in decision trees and trains large datasets efficiently. Trees in the random forest are built with bootstrap samples. Variation between trees is achieved by random selection of observations.

Considering the large volume of the dataset, we use the random search method with 3-folds cross-validation to find the best hyperparameters. We choose two hyperparameters (n_estimators and max_depth) to tune the model. With large n_estimators, we can reduce error and get more accurate predictions; with max_depth, we early stop the tree to avoid reaching pure leaf and overfitting. The best hyper-parameters found are n_estimators = 300 and max_depth = 30.

Model 3: LightGBM

LightGBM is a gradient boosting framework that uses tree based learning algorithms. It has faster training speed, higher efficiency, lower memory usage and it supports parallel learning, so it’s capable of handling large-scale data.

We used the 5-folds cross-validation and Bayesian Optimization approach via Optuna to fine-tune the hyper-parameters of the Lightgbm model. Bayesian Optimization provides a principled technique based on Bayes Theorem to direct a search of a global optimization problem that is efficient and effective.

The hyper-parameters we fine-tuned includes *n_estimators, learning_rate, num_leaves, max_depth, min_data_in_leaf, lambda_l1, lambda_l2, min_gain_to_split, bagging_fraction, bagging_freq, feature_fraction.*

We used log-loss as the evaluation metric to find the best set.

c) Model Evaluation

i). Comparison of Accuracy, Precision, Recall, F1-score, and AP score

Model	Dataset	Accuray	Precision	Recall	F1-score	AP Score
XGBoost	Development	0.9729	0.9536	0.9410	0.9472	0.9884
	Test	0.8960	0.8040	0.7948	0.7994	0.8868
Random Forest	Development	0.9974	1.0000	0.9900	0.9950	0.9926
	Test	0.8927	0.7957	0.7916	0.7936	0.6842
LightGBM	Development	0.8960	0.8652	0.8625	0.8638	0.8840

	Test	0.8987	0.8693	0.8669	0.8681	0.8899

Table 1. Model Evaluation Results

This table summarizes the accuracy, precision, recall, F1-score, and average precision score on the development and test datasets of XGBoost, random forest, and LightGBM models. Among these models, LightGBM outperforms the other two models on the test dataset, especially in terms of precision, recall, and F1-score, and effectively reduces Type I and II errors. Both random forest and XGboost exhibit a slight overfitting problem. Random forest, as a representative of a bagging model, does not outperform boosting models like LightGBM and XGBoost.

ii). Comparison of ROC Curve and AUC

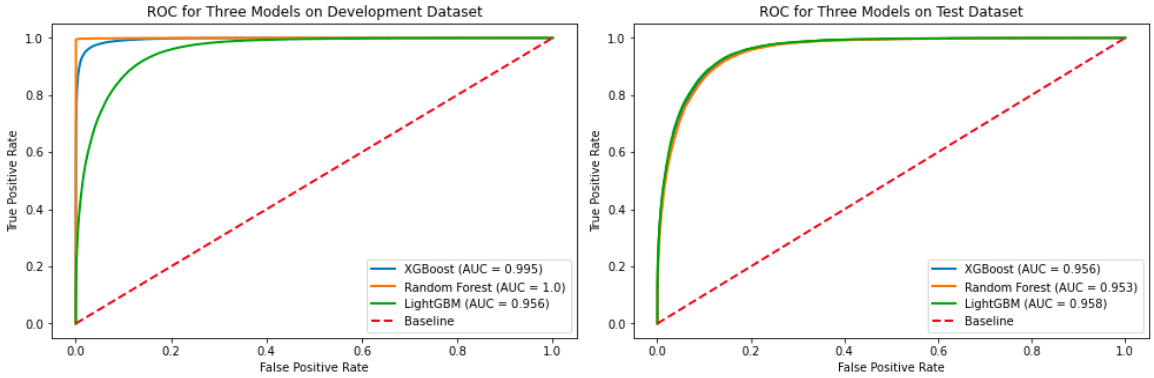

Figure 2. ROC Curve and AUC Among Three Models

These two graphs show the ROC curve and AUC score of three models on the development and test datasets. For the development dataset, XGBoost and random forest have higher AUC under the ROC curve than LightGBM. For the test dataset, these three models exhibit very similar curves and AUC scores, which suggests that they have equally excellent abilities to distinguish two classes (default and not default).

d) Model Calibration

In order to interpret the output in terms of a probability, we need to calibrate the model. We calibrated the trained LightGBM model which has better performance in the previous steps. The Brier score of the original LightGBM model on the test set is 0.072, then we used the **Isotonic** method to calibrate the model. The new Brier score was reduced to 0.0711.

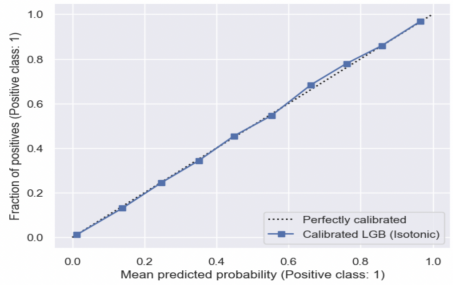

Figure 3. Calibration Plots Using Isotonic Method

5. Limitation & Conclusion:

i). False Prediction

A series of metrics from a single customer tended to be closely related, any one of which could show ominous signs when its potential default came close even though the response variable Y did not reflect that momentum. The metric for the defaulting month and that for a month prior to the default were very similar, making it harder to determine likelihood of default for an exact month.

ii). Default Rate

For a dataset that has a very low default rate, such as ours, gathering default data was inherently difficult because a customer defaults only once before the system denies his/her further credibility. And the fact that we took multiple data points from non-defaulting dates but ended up with just one default month made it hard to gather a comparable number of dataset for each customer.

iii). Future Direction

Our project focused on monthly prediction; in the future, this project could be furthered to have the period of prediction extended from a month ahead to a year, which will provide more insights for preventive management.