UCI Credit Default Analysis

Qinghao Lin
Columbia University
ql2461@columbia.edu

Letian Qi
Columbia University
lq2217@columbia.edu

Bowen Zhang
Columbia University
bz2461@columbia.edu

Zhonglin Liu
Columbia University
zl3217@columbia.edu

Daisy Liu
Columbia University
xl3236@columbia.edu

Abstract

As loan default continues to become a major problem for loan lenders, we are curious to explore how we could leverage the user demographic and historical credit data to predict their probability of default. In this project, we built multiple visualizations to explore the relationships among different variables and constructed several machine learning models such as Random Forest Classifier, XGBoost, LightGBM, and CatBoost to perform the classification task. We evaluated the model performance using metrics such as AUC, f1, and accuracy, and the results demonstrated that our models could significantly improve the prediction from the baseline model.

1. Background

In recent years, the credit card issuers in Taiwan faced the cash and credit card debt crisis. A Taiwan-based credit card issuer wants to better predict the likelihood of default for its customers. The results would inform the issuer's decisions on who to give a credit card to and what credit limit to provide. The results would help the issuer have a better understanding of their current and potential customers, which would inform their future strategy, including their planning of offering targeted credit products to their customers and reducing the damage and uncertainty in terms of risk control(Evgeniou, Zoumpoulis, n.d.).

2. Data Profiling

The loan default dataset contains 25 columns, with demographic features including ID, SEX (1=male, 2=female), MARRIAGE (1=married, 2=single, 3=others), AGE, behavioural features including payment status (PAY_0 - PAY_6), bill statement (BILL_AMT1 - BILL_AMT6), and payment history (PAY_AMT1 - PAY_AMT6), and amount of credit given (LIMIT_BAL). The target class to predict is default.payment.next.month (1=yes, 0=no), which indicates whether a client defaults.

3. Exploratory Data Analysis

We first check the distribution of the target class, as visualized in the below chart. In the dataset, we have 22.1% of the clients defaulting, versus 77.9% of those who do not default, indicating a class imbalance.

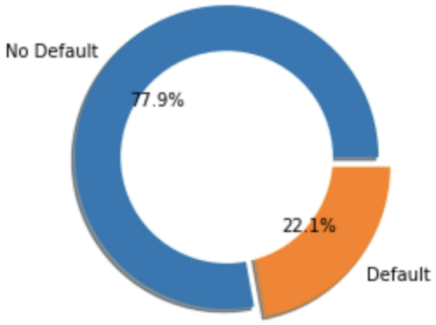

Figure 1. Class Distribution

Then, we check the correlation between the features, excluding the ID column. From the correlation heatmap below (Figure 2), we notice that the payment status features (PAY_0 - PAY_6) and the payment history features (PAY_AMT1 - PAY_AMT6) are highly correlated, indicating that clients generally tend to follow their payment habits over time.

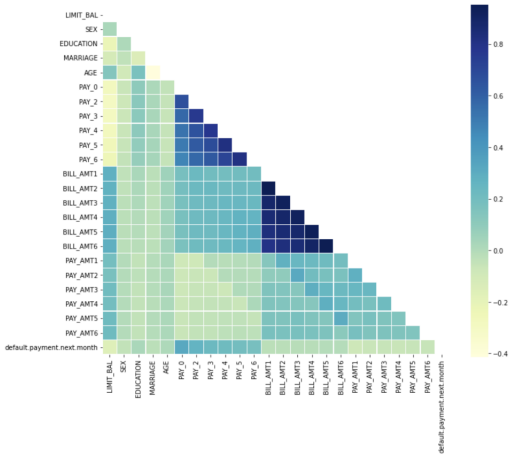

Figure 2. Correlation Heatmap

For each of the three categorical features, SEX, MARRIAGE, and AGE, we visualize the distribution of each category, among clients who default and clients who do not default. Since the dataset is imbalanced, we choose to visualize using percentages instead of frequencies. Clearly, from the charts, the distributions of gender and marital status are relatively balanced in the two target classes.

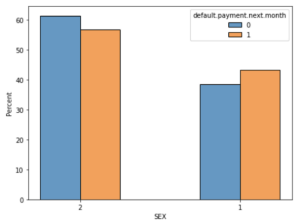

Figure 3. Sex

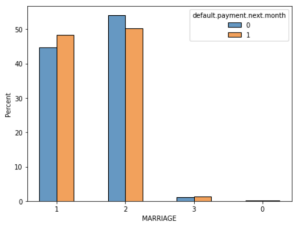

Figure 4. Marriage

On the other hand, the plot for the EDUCATION column indicates that clients who received undergraduate or graduate education are in general less likely to default.

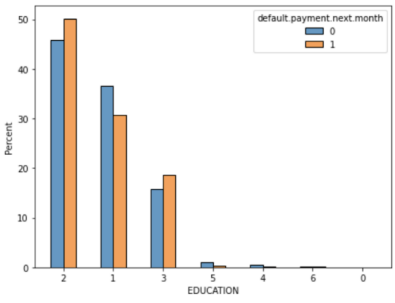

Figure 5. Education

To explore the numerical features, similarly, we normalize and visualize the percentages using histograms, across the two target classes. The limit balance plot shows that clients who are more likely to default payments are within the 0-0.15 credit range, and within the 0.15-0.6 range, fewer clients default.

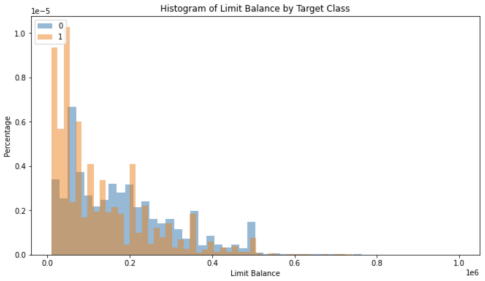

Figure 6. Limit Balance

The age plot indicates that within the 25 - 35 age range, clients are less likely to default, otherwise, the distributions are similar in general.

Figure 7. Age

4. Preprocessing

Data preprocessing refers to the manipulation or dropping of data before it is used in order to ensure or enhance performance, and is an important step in our fraud detection project and general machine learning processes. The quality of data is first and foremost before running any analysis. If there is much irrelevant and redundant information present or noisy and unreliable data, then knowledge discovery during the training phase is more difficult. Besides the improvement of data quality, essential transformations of data structure and organization will simplify the analysis in the next step. To better prepare our data for training and testing, multiple preprocessing methods are applied before model building, including data cleaning, label drop and column transformation.

4.1. Data Cleaning

Rule-based data cleaning strategies are introduced to ensure the quality of training and testing datasets. Bad records

with negative payment amounts, out-of-range categorical indicators and null values, which make no sense for further analysis and will generate noise when included, are removed.

4.2. Label Drop

Given the property of training and test data, different proportions of the original datasets are kept in the two sets by dropping responded labels. In addition, the label of 'ID' is dropped for training. subsectionColumn Transformation Labels of numerical, categorical and binary variables follow different storage logics. To handle the difference between original data, a preprocessor leveraging make_column_transformer and OneHotEncoder from scikit-learn package is built to convert columns in different settings to one-hot numeric arrays. Such transformation is necessary for following models as scikit-learn models require formatted input of numerical values.

5. Model Building

In this project, we tested out a couple of different machine learning models: Logistic Regression, Decision Tree, Random Forest Classifier, XGBoost, LightGBM, and CatBoost. Since the dataset is heavily imbalanced, we applied several measures in each model to balance the dataset. For example, we calculated the ratio of positive and negative samples and passed it as a hyperparameter into XGBoost, LGBM, and CatBoost Models. For random forest and decision tree models, we simply set the class_weight hyperparameter to "balanced", which gives higher weights to the minority class.

5.1. Logistic Regression

Logistic Regression fits a regression analysis on all variables and applies a sigmoid function to map the result into a binary number.

5.2. Decision Tree

Decision Tree Classifier calculates the Gini index at each split and determines the best feature to make the split. However, it could easily overfit without tuning hyperparameters.

5.3. Random Forest

Random Forest is an ensemble method that takes a majority vote of multiple decision trees, which usually yields better results than a single decision tree classifier.

5.4. XGBoost

XGBoost is one type of gradient boosting technique with 2 key improvements: the algorithm applies openMP while building a decision tree to enable its branches to grow independently, improving the performance speed. At the same time, it implements regularization to avoid overfitting.

5.5. LightGBM

LGBM divides continuous feature values into discrete bins to fasten the training procedure. At the same time, it takes a heuristic approach by splitting the nodes leaf-wise instead of level-wise, which might cause overfitting.

5.6. CatBoost

CatBoost grows each leaf node based on the same condition instead of using different features and generally has a much slower performance than other models.

6. Feature Importance

To improve the model interpretability, we find the most important features that contribute to loan default. As shown in the chart below, we can see that the client's previous repayment status would largely determine whether they will default in the future. At the same time, the amount of given credit is also a good indicator. With this result, we can inform the bank to pay extra attention to clients with low credit ratings and who have defaulted previously.

Feature	Importance
LIMIT_BAL	0.081531
PAY_AMT1	0.079949
PAY_AMT2	0.075737
BILL_AMT1	0.073659
AGE	0.070687
BILL_AMT2	0.067451
PAY_AMT3	0.065522
BILL_AMT3	0.063988
BILL_AMT6	0.062904
BILL_AMT4	0.062461

Table 1. Feature importace

7. Hyperparameter Tuning

Before going into hyperparameter tuning, we planned to perform 4-fold cross-validation on the training data to see if one specific fold can produce a better score. We then picked the fold that has the best performance and used it during the later hyperparameter tuning for each model. However, most scores of each model are no better than the scores before. This implies that 4-folds cross-validation does not help boost the performance, so we rejected the method and directly went to tuning processes.

Fig 8 shows the compared performance of 4-folds cross-validation. Rf_best, xgb_best, lgbm_best and catboost_best are models with specific folds trained. We can see their performance is no better than the one of the models built in previous processes.

	fit_time	score_time	test_score	train_score
decision tree	0.487 (+/− 0.019)	0.017 (+/− 0.002)	0.572 (+/− 0.008)	1.000 (+/− 0.000)
random forest	6.591 (+/− 1.493)	0.195 (+/− 0.096)	0.714 (+/− 0.008)	1.000 (+/− 0.000)
XGBoost	2.269 (+/− 0.025)	0.035 (+/− 0.006)	0.728 (+/− 0.004)	0.767 (+/− 0.001)
LightGBM	0.691 (+/− 0.012)	0.047 (+/− 0.001)	0.728 (+/− 0.003)	0.869 (+/− 0.001)
CatBoost	10.722 (+/− 1.256)	0.053 (+/− 0.003)	0.732 (+/− 0.003)	0.833 (+/− 0.002)
rf_best	4.921 (+/− 0.065)	0.129 (+/− 0.007)	0.726 (+/− 0.007)	0.990 (+/− 0.001)
xgb_best	2.052 (+/− 0.045)	0.031 (+/− 0.002)	0.727 (+/− 0.005)	0.764 (+/− 0.001)
lgbm_best	0.743 (+/− 0.034)	0.064 (+/− 0.002)	0.728 (+/− 0.002)	0.845 (+/− 0.001)
catboost_best	1.109 (+/− 0.036)	0.048 (+/− 0.002)	0.731 (+/− 0.006)	0.769 (+/− 0.001)

Figure 8. the compared performance of 4-folds cross-validation

We selected 4 models to perform hyperparameter tuning on them. For each model, we chose parameters in an appropriate range to maximize the performance. We used randomized search instead of grid search because we can have a broader range of values to make random combinations. In addition, random search finds the best parameters in fewer iterations. After finding the best estimator for each model, we ran the cross-validation on the models to see their scores. In Fig 9, most models are expected to boost their performance through hyperparameter tuning. And the CatBoost model has the highest score on the validation data.

	fit_time	score_time	test_score	train_score
decision tree	0.572 (+/− 0.037)	0.024 (+/− 0.015)	0.568 (+/− 0.007)	1.000 (+/− 0.000)
random forest	9.112 (+/− 3.327)	0.267 (+/− 0.250)	0.715 (+/− 0.004)	1.000 (+/− 0.000)
XGBoost	2.199 (+/− 0.015)	0.030 (+/− 0.002)	0.733 (+/− 0.004)	0.768 (+/− 0.001)
LightGBM	0.645 (+/− 0.032)	0.044 (+/− 0.002)	0.730 (+/− 0.004)	0.869 (+/− 0.002)
CatBoost	10.161 (+/− 0.643)	0.051 (+/− 0.002)	0.736 (+/− 0.003)	0.834 (+/− 0.003)
xgb_best	2.222 (+/− 0.051)	0.031 (+/− 0.001)	0.733 (+/− 0.004)	0.768 (+/− 0.001)
lgbm_best	0.542 (+/− 0.223)	0.048 (+/− 0.022)	0.734 (+/− 0.004)	0.771 (+/− 0.001)
catboost_best	2.518 (+/− 0.338)	0.058 (+/− 0.020)	0.737 (+/− 0.003)	0.780 (+/− 0.001)
rf_best	25.301 (+/− 0.308)	0.571 (+/− 0.015)	0.732 (+/− 0.005)	0.991 (+/− 0.000)

Figure 9. The performance after hyperparameter tuning

8. Model Evaluation

We first evaluated our model performance based on the roc_auc scores. All of our tuned models have very similar performance, with an auc score of around 0.72. LGBM and CatBoost have slightly better performance than random forest and XGBoost classifiers.

Since the dataset is imbalanced, auc score might not be the best metric to reflect the actual model performance. Therefore, we also tested out our models using f1 score, which is a harmonic mean of recall and precision. However, all our tuned models perform poorly with the highest f1 score being 0.467 (LGBM, XGBoost).

9. Future Work

Due to the limited time on this project, there are still many things we could try in the future to improve our model performance. For example, we could apply various feature selection methods such as the wrapper and the filter methods to keep only the important variables. We could also tune more hyperparameters on each model with f1 as the main scoring metric. In terms of data preprocessing, we recognize that tree-based models perform poorly with one-hot encodings because they will create many binary sparse features so that the algorithm sees them as independent variables whenever a split is performed, making the gain from each split trivial. Instead, we could try to use ordinal encodings or apply non-sklearn frameworks such as the H2O framework, which does not require OHE to process categorical variables.

Model	AUC Score	F1 Score
RandomForest	0.723	0.417
LGBM	0.726	0.467
XGBoost	0.724	0.467
CatBoost	0.726	0.045

Table 2. Test result, overall passing rate 54%

10. Reference

Theos Evgeniou, Spyros Zoumpoulis. (n.d.). Classification for Credit Card Default. http://inseaddataanalytics.github.io/INSEADAnalytics/CourseSessions/ClassificationProcessCreditCardDefault.html