Introduction to FinTech Final Project Report

Group 24 / TEAM_4415

B09703028 呂紹嵊 B09902023 蔡孟錡 B09502156 楊景淳

1. Abstract

In this project, we aimed to design an algorithm to train a model that could detect credit card fraud through a transaction record. The training dataset is given based on real-world customer transaction records, which contains a variety of different features such as payment amount, the date and time the transaction was made, and location etc. To measure the performance of our prediction, our valuation method is to maximize the f1-score. First, we treat the task as a classification problem under the *Scikit-learn* framework, and deal with data preprocessing using various methods. Second, we use XGB Classifier to fit the data. Last, we use grid search cross-validation to evaluate the model. Our final score in the competition is 0.870894 in public and 0.582047 in private, ranking 36 and 35 out of over 200 contestants respectively.

2. Pre-processing dataset

Our data preprocessing process can be divided into three parts listed as follows.

I. Missing Data Handling

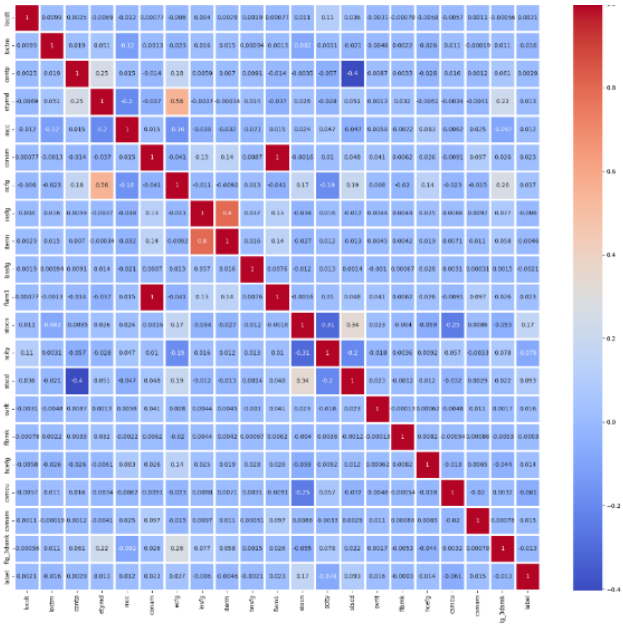
We first listed the missing feature of each data, and since the data is very imbalanced, we decided to delete the rows with more than 5 features missing while not being a fraudulent charge in order to maximize the information that we can get from the data given. We also deleted the *stscd* column since it had most of its data missing.

To tackle the missing data problem and prevent our model from overfitting at the same time, we first created a winsorized version of the dataset. Then, we tried to fill in the numerical missing values using linear regression within the labels that have absolute correlation above 0.11.

The regression formula is listed as:

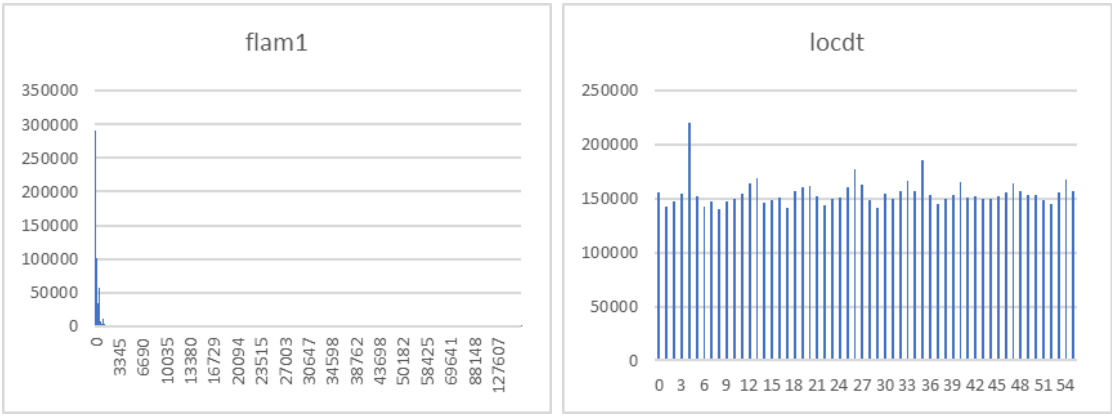
$$x_j = \alpha + \Sigma\beta_i x_i + \epsilon,\ where\ |\rho_{ij}| > 0.11$$

Take the s*tocn* variable for instance, by plotting out the correlation chart shown below, we can see that *csmcu*, *stscd*, *scity*, and *ecfg* have high correlations with our target variable, so we use these values to predict the missing values.



II. Feature Scaling

For numerical features, we first plotted out the distribution of each feature and divided them into two groups. One is those with a long tail to the right, including *iterm*, *flam1* and *csmam*. We use logarithm transformation for these kinds of features. The other is those without evident distribution patterns, including *locdt, loctm* and *conam*. We used standardization for these kinds of features. Chart a and b are the original distribution of *flam1* and *locdt* as representative of two groups.



a. *Logarithm transform*

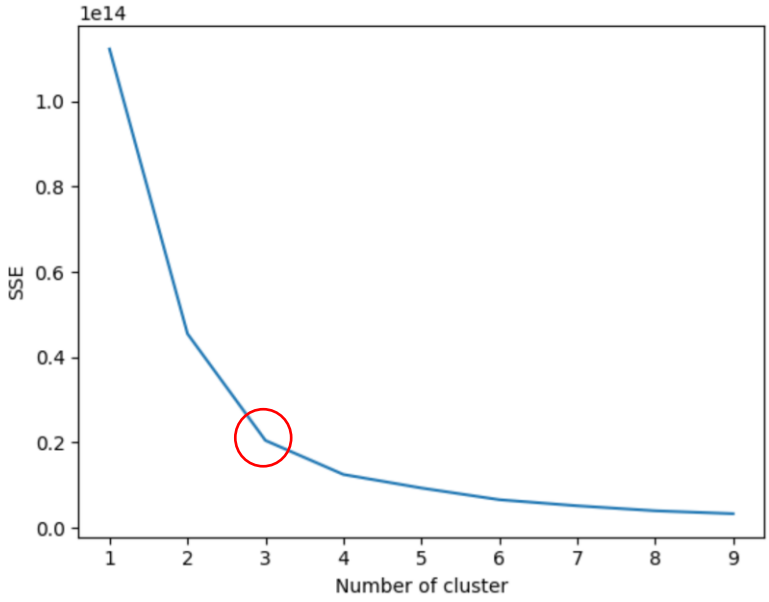
b. *Standardization*

Second, for the categorical features, we divided them into 2 groups, one with categories less than 10, and the other with categories more than 10. We used one hot encoding for the first type of features and Catboost encoding for the second type of

features with an eye to reducing the complexity. Note that we have done some extra preprocessing for features *stocn, ecfg,* and *mcc,* as we think these features have a bigger impact on the outcome of the results. We used K-means clustering for these three features, which will be discussed in the next paragraph.

III. K Means Clustering

For the three important features, we calculated each features' percentages for different card number, determines the optimal number of clusters using the elbow method (see below for SSE visualization example), and performs K-Means clustering on the card number based on their feature's percentage. We then add a new column indicating the cluster assignment for each card number withing that feature, namely the *oversea_percentage, online_transaction, and MCC_code* clusters. After implementing the method mentioned above, we sorted the *stocn, ecfg,* and *mcc* features into 2, 2 and 3 clusters respectively.



Elbow method for mcc feature (k =3)

3. Model Selection and Tuning

I. Model selection

We use XGB Classifier from xgboost as our main model. XGB Classifier is known for its efficiency due to its parallelization and regularization techniques. It utilizes the gradient boosted trees algorithm and incorporates various optimizations,

making it faster than traditional gradient boosting algorithms. It can handle a large number of features and samples efficiently, thanks to its parallel processing capabilities. On top of that, XGB Classifier offers moderate interpretability. While it doesn't provide direct feature coefficients like logistic regression, it offers feature importance scores that indicate the relative importance of each feature in the model's predictions. The most significant disadvantage of XGB Regressor is that it takes time and patience to find the best parameters.

II. Loss function

With the objective of our XGB Classifier set as 'binary:logistic' and lambda set to 1, the resulting objective function of our model is the logistic regression's loss function with L2 regularization to prevent overfitting issues.

The objective function is shown as follow:

$$Obj = \sum_{i=1}^{n} [y_i \ln(1 + e^{-w^T x_i}) + (1 - y_i) \ln(1 + e^{w^T x_i})] + \lambda ||\mathrm{w}||^2$$

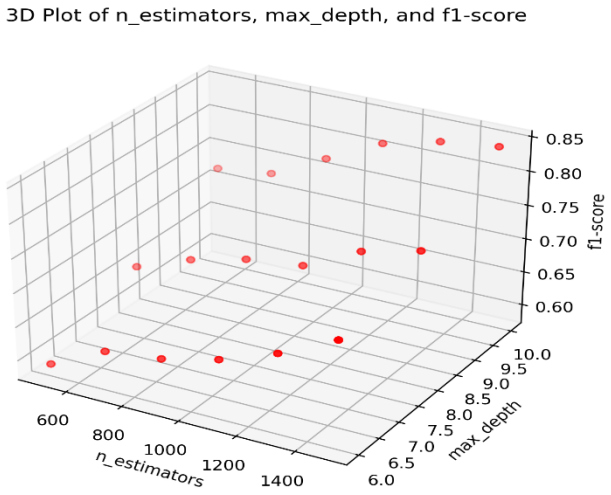
III. Hyperparameters

There are several hyperparameters in XGB Classifier including learning rate, max depth, n estimators, gamma, subsample and so on. We use *GridSearchCV* from *sklearn.model_selection* to find the best parameters for our dataset. First, we set range for some selected hyperparameters as the following chart showed. Next, *GridSearchCV* will go through 5-fold cross validation to get the best parameter and the corresponding best f1-score within the given range. We find out that the best score, 0.84492, occurred at the last search. There is another observation that with different *tree_method* doesn't make much difference on the score. So, we adjust the *param_grid* based on our observation and try to tune the combined dataset of training.csv and public.csv. However, the best score of this attempt is 0.83759. To find the best set for our model, we look into the CV score trend of each parameter. We first use a larger learning rate of 0.1 and get the best *n_estimator* to be 300. Second step is to find the *max_depth* which is 10. The next step is to find the best gamma, which turns out to be 0.1. The last step is to find the best subsample which is 0.9.

After finding all the parameters, we changed the *learning_rate* to a smaller one, 0.05, to find the best *n_estimators* again, which is 2000. With this set of parameters, we get a public score of 0.870894, which is better than all the attempts beforehand. All the details of *param_grid* settings, best parameter set and best score of each attempt of grid search are organized into a chart in the next section.

4. Results

All in all, our final approach is XGB Classifier trained with dataset preprocessed as mentioned above and the hyperparameters is to browse through multiple comparisons. Since the original dataset had some missing data and imbalanced in some columns, preprocessing the dataset can largely improve our model's performance. One can discover the improvement through comparing the public score we got with and without preprocessing as listed in the leaderboard part below.



I. Validation set

Dataset	param_grid	Best Parameter Set	Best Score
Dataset_1st training.csv	'n_estimators': [500-1500], 'max_depth': [6, 8, 10], 'learning_rate': [0.01,0.05], 'subsample': [0.8,0.9], 'gamma': [0, 0.1], 'tree_method': ['auto', 'hist']	n_estimators: 1500 max_depth: 10 learning rate: 0.05 subsample: 0.9 gamma: 0.1 tree_method: 'hist'	0.84492

Dataset_1st training.csv + Dataset_2nd Public.csv	'n_estimators': [1400, 1500, 1600], 'max_depth': [10, 12], 'learning_rate': [0.05,0.06], 'subsample': [0.88,0.9], 'gamma': [0.08, 0.12],	n_estimators: 1500 max_depth: 12 learning rate: 0.06 subsample: 0.88 gamma: 0.08	0.83759

II. Leaderboard

i. XGBoost with depth 8 on original dataset: 0.597668 (public leaderboard)

ii. Add onehot encoding: 0.632092 (public leaderboard)

iii. Final model: 0.870894 (public leaderboard) /
0.582047 (private leaderboard)

5. Extra Model Comparison

Apart from the XGboost model, we also implemented different models to compare and contrast the difference between them.

I. Logistic Regression

We first selected Logistic Regression from *sklearn.linear_model* as our baseline model, getting 0.5319 as our public score. The reason we chose this is that it is simple to implement and efficient, especially when dealing with large datasets. It only took about 24 seconds to train a model and evaluate f1 using cross validation on CPU. It is also generally scalable as long as the number of features doesn't grow significantly. Moreover, logistic regression provides good interpretability. The coefficients associated with each feature can be directly interpreted as the impact of that feature on the prediction. Nevertheless, due to the linear decision surface of logistic regression, it can only be applied to linear data, which isn't the case of our dataset, so it may not produce a highly accurate prediction.

II. Support Vector Machine

We selected Support Vector Classification from *sklearn.svm*, which had been introduced in class, as our third model. The efficiency of support vector classification depends on the kernel chosen. Because training an SVM model involves solving a

quadratic programming problem, which can be time-consuming, especially when using non-linear kernels. It also takes lots of time or costs a lot finding the best parameters for the kernel. As for scalability, SVM is not suitable for large data sets due to the training complexity. SVM can be less interpretable compared to logistic regression. It uses support vectors and the distance between them to make predictions, which may not have a direct relationship with the original features. Even though the low efficiency, support vector regression has high accuracy, especially in high-dimensional space and when there is a large gap between classes, which is suitable for our dataset.

We tried out all the kernel types, found that rbf is most suitable for our data set (ran locally after stage 1), getting 0.6032 as our public score, which is better than logistic regression, but worse than XGB Classifier.

6. Workload

B09703028 呂紹嵾: Code framework, Data Preprocessing, Model Training Setup.

B09502156 楊景淳: Model Tuning, Report Framework.

B09902023 蔡孟錡: Cross Fold Validation, Missing Data Imputation.

7. Reference

- Chih-Jen Lin, Hsuan-Tien Lin. An ensemble of three classifiers for kdd cup 2009: Expanded linear model, heterogeneous boosting, and selective naïve bayes.
- Machine Learning Technique YouTube videos by Professor Lin
- https://blog.csdn.net/m0_37870649/article/details/82707197
- https://github.com/CubatLin/TBrain-E.SUN-AI-Open-Competition-Fall-2019-15th-place-Feature-Engineering
- https://blog.csdn.net/han_xiaoyang/article/details/52665396