Machine Learning On Home Credit Default Risk

Peng Ye
Department of CSE
HKUST
pyeac@cse.ust.hk

Abstract

Home Credit Default Risk[1] is a Kaggle contest supported by Home Credit. The aim is to predict their applicants' repayment abilities through data mining and machine learning tools. This report analyzes the dataset provided in the competition, performs some feature engineering and runs different machine learning methods. Finally a simple averaging ensemble strategy is used to achieve a score of 0.782, showing that machine learning can be powerful and help us make good use of the data.

1 Introduction

In Home Credit Default Risk competition, a dataset containing users' telco and transaction information is given. The objective is to predict whether an applicant will be capable of repaying a loan. Accurate predictions can identify clients who are capable of repayment so that loans can be given. And applications from untrustworthy lenders will be rejected so that the company will not lose money.

Machine learning methods are powerful in analyzing data and making predictions. A question naturally arises: *is it possible to apply machine learning methods to the dataset, to help us predict clients' repayment abilities?* To answer this question, in this project five different machine learning models are built to make predictions based on the dataset. Some feature engineering skills are performed to make the data more suitable for model training. The final score shows the models' abilities to make accurate predictions.

One may also ask: *can we train models on only part of the features while achieving nearly the same performance?* In this report we also study feature selection and dimensionality reduction approaches. They are used to reduce the number of features fed into the models. Results suggest that one can use much fewer features to predict while achieving nearly the same accuracy.

In this project we perform several machine learning methods, compare the experimental results and provide some analysis. The code of this project can be found on github[2]. The remaining parts are organized as follows: section 2 describes and analyzes the dataset provided in the competition, section 3 describes the models used in the experiments, section 4 shows the results on the main table and merged table after feature engineering, section 5 applies feature selection and dimensionality reduction and analyzes the effects, section 6 states the conclusion of this report.

2 Dataset overview

The dataset has 7 tables:

[1] https://www.kaggle.com/c/home-credit-default-risk/
[2] https://github.com/isdkfj/MATH5470-project

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

- The main table `application` contains information about each loan application at Home Credit. This table is split into two files: train (with label) and test (without label).
- Table `bureau` contains all client's previous credits provided by other financial institutions that were reported to Credit Bureau.
- Table `bureau balance` contains monthly data about the previous credits in the bureau.
- Table `POS CASH balance` contains monthly balance snapshots of previous POS and cash loans that the applicant had with Home Credit.
- Table `credit card balance` contains monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
- Table `previous application` contains previous applications for loans at Home Credit of clients who have loans in the `application` table.
- Table `installments payments` contains repayment history for previous loans at Home Credit.

The training data contains 307,511 records and the objective is to predict whether the clients will repay loans on time for 48,744 loans in the testing data. The total number of origin features is more than 200 and distributed in different tables. So careful analysis of the connection between tables is of great importance to utilize the data. And since the label is a binary variable (0 for will repay on time and 1 for not), this is a supervised binary classification task.

The evaluation metric of this competition is Area Under the Curve (AUC). This means the submission file will not contain the definite 0-1 predictions, but probabilities between 0 and 1 instead. The reason why the organizer uses AUC is that the label distribution is unbalanced - in the real world, most people will repay loans on time while only a small group of people are untrustworthy. We can also see this from the training data: only 24,825 out of the 307,511 loans don't get repayment on time, even less than 1%.

3 Model description

In this project we will use 5 different models and compare their performance:

- **Logistic regression** It uses $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ to estimate the output probability. It optimizes the cross entropy loss $\frac{1}{N}\sum_{i=1}^{N}(y_i \log(h_\theta(x_i)) + (1-y_i)\log(1-h_\theta(x_i)))$ to fit the trainning data.
- **Random forest** It generates many decision trees by bootstrap sampling and uses ensemble to predict.
- **Ridge regression** It can be seen as adding a regularization term on the objective function of linear regression: $\min_\theta \|X\theta - y\|_2^2 + \lambda\|\theta\|_2^2$. Note that the output of this model is indeed not a probability - the output value may not be in the range $[0, 1]$. So after obtaining the output, we will clip the value to make it a valid prediction.
- **Neural network** The network structure we used has three hidden layers with 400, 100, and 20 neurons each. The activation of each layer is ReLU except for the final layer, where a sigmoid function is applied.
- **Light Gradient Boosting Machine (LightGBM)** It is a gradient boosting (GBT, GBDT, GBRT, GBM or MART) framework based on decision tree algorithms. Due to its great performance, fast training speed, and low memory use, it is the most popular model used in many Kaggle contests.

We implement the first three models from the Scikit-Learn[3] package, use Keras[4] to build the neural network, and use LightGBM's official python package[5].

[3] `https://scikit-learn.org/`

[4] `https://keras.io/`

[5] `https://github.com/Microsoft/LightGBM`

4 Experiment

In this section, we will apply the machine learning methods mentioned above to the dataset and see how the performance will be. In the beginning, we will use only the `application` table for training and making predictions. After that, we will try to join the tables provided in the contest together and perform some simple feature engineering. Then we run the models on the merged table and see how much AUC increase can be obtained.

4.1 Main table results

As a starting point, let's ignore the other 6 tables and focus on the main table - `application` table. The training table has 122 columns including the transaction id and target variable. Unfortunately even for this table only we cannot directly run the models.

First, there are 67 columns that have missing values. Some of them have even nearly 70% missing values. So here we use median imputation strategy to deal with missing values - calculate the median value based on non-missing values and then replace the missing values with the median.

There are 16 categorical features in this table. If a feature has only two categories, we can easily convert the values to 0 and 1. For features that have more than two categories, we use the one-hot encoding approach to convert the objects into values. There are two things that we should be careful about: some categorical features can have missing values, so we should do one-hot encoding first (ignore the missing values) and then apply median imputation. Some categories may only occur in the train table, making the number of columns of the train and test table inconsistent. So we should drop the columns that only occur in the train table.

Since different features may have different magnitudes, to make sure that features with large values won't significantly affect our prediction, the values of each column will be normalized to range $[0, 1]$ before being fed into models.

After the above preprocessing, we are able to run our models on the main table. We use a 10-fold cross-validation to evaluate the performance. The result is shown in table 1.

	Logistic Regression	Random Forest	Ridge Regression	Neural Network	LightGBM
AUC Score	0.684	0.691	0.729	0.734	0.746

Table 1: Results on main table.

4.2 Table merging and feature engineering

After obtaining the results using the main table, the next step is of course merging tables. Let's merge the tables one by one.

The `bureau` table contains clients' previous credit records. For every row in `application`, the client can have many credits before the application date. So it may have many rows in `bureau`. These rows have the same loan ID as in `application`. To extract features from this table, we first make credits with the same loan ID as a group and record the number of previous credits as a new feature. Then for numeric features, we calculate the sum, average, minimum and maximum values as new features. For categorical features, we first apply one-hot encoding and then calculate the sum and average for each column as new features. After this procedure we have created a lot of features for each loan ID. Then we join this new table with `application` to finish the table merging.

The `bureau balance` table contains monthly balances of previous credits in Credit Bureau. This time we make rows with the same bureau ID as a group and then apply the same feature extraction method as in `bureau` to extract features. Then we add loan ID from `bureau` by identifying the bureau ID, group by loan IDs and then apply feature extraction again. Finally we can join this table with `application`.

The `previous application` table contains previous application history of clients. We can apply the same aggregation approach as in `bureau` and then merge it to `application`.

The `POS CASH balance` table contains monthly balance of previous POS and cash loans of the clients in Home Credit. For this table we first aggregate previous credit IDs and then aggregate loan IDs. After that we can join it with `application`.

The `installments payments` and `credit card balance` have the same structure as `POS CASH balance`. So we can apply the same aggregation approach to get the joined table.

After the merging process, we can find that there are many columns that have many missing values. By conducting some experiments we found that it would be useful to remove some columns that have too many missing values. In the experiments we remove all columns with more than 70% missing values either in train or test table. There are 451 such columns and after alignment between train and test table, we got a train table with 989 features (including the loan IDs and the target).

Then again we apply the median imputation and 0-1 normalization technique to this merged table. Till now we have done all the merging and feature engineering, we are able to proceed the model traning process.

4.3 Results on merged table

After table merging and feature engineering, we have got a much larger table. We then apply the five models to this table. The performance results are shown in table 2.

	Logistic Regression	Random Forest	Ridge Regression	Neural Network	LightGBM
AUC Score	0.710	0.696	0.761	0.770	0.781

Table 2: Results on merged table.

From the results, we can see that all models have gained increases in AUC scores. The best model, LightBGM, achieves an AUC score of 0.781.

For the final submission, we use a weighted averaging strategy - compute a weighted average of probabilities output by different models. By doing so we achieve a private score of 0.782.

5 Feature selection and dimensionality reduction

Although we have merged the tables and had a satisfactory result given by our machine learning models, we are still interested in the following question: *can we train models on only part of the features while achieving nearly the same performance?*

In this section, we are going to try feature selection and dimensionality reduction to see if it is possible to reduce the number of features.

5.1 Feature selection by co-linearity and feature importance

Co-linearity is a property we can use to reduce the number of features. Because if two features are strongly linearly correlated, we can retain only one of them because the other one can be easily deduced. We will remove a variable if its correlation coefficient with another variable is greater than 0.95. We found there are 228 columns that can be deleted.

After that, we perform feature selection through feature importance. We train LightGBM (we choose it because it has the highest performance) on the table. From the trained model we can extract every feature's importance. If a feature's importance is 0, we know that the boosting doesn't use these features. So we can remove it. By removing all features with zero importance, we obtain a table with 610 features.

We then apply the five models to these selected features. The performance results are shown in table 3. Compare the results with table 2, there is only a very tiny difference (which can be seen as random noise) between the AUC scores. This means we have successfully reduced the number of features while maintaining the performance.

From the feature importances showed in figure 1 we can also see that three external source features are most important for us to make predictions.

	Logistic Regression	Random Forest	Ridge Regression	Neural Network	LightGBM
AUC Score	0.714	0.705	0.759	0.768	0.779

Table 3: Results on selected features.

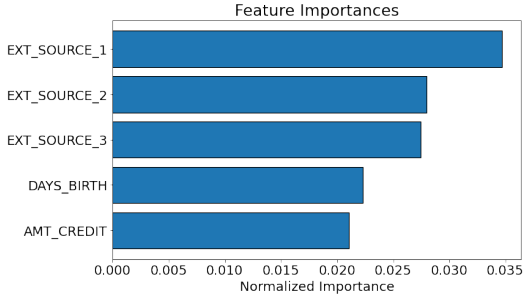

Figure 1: Normalized feature importances.

5.2 Dimensionality reduction by PCA

The most popular way of dimensionality reduction is Principal Component Analysis (PCA). In addition to feature selection, we also try PCA to see whether it is capable of reducing the dimension. After applying PCA, we retain the same number of principal components as selected features (610). The results are shown in table 4. From the table, we can see there is a huge decrease in AUC (especially in logistic regression and random forest), which implies PCA may not be suitable for this dataset.

	Logistic Regression	Random Forest	Ridge Regression	Neural Network	LightGBM
AUC Score	0.636	0.627	0.752	0.767	0.769

Table 4: Results on PCA features.

Although even only the first principal component accounts for more than 95% of explained variance, if we plot the target distribution of the first two principal components as in figure 2, we can see the points still gather together, suggesting that it would be hard for our models to predict through this dimension.

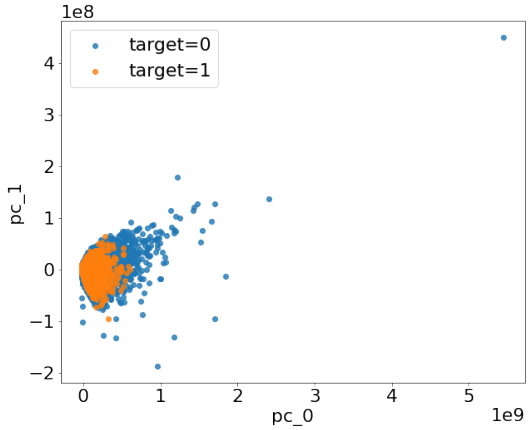

Figure 2: First two principal components.

6 Conclusion

In this report, we first analyze the dataset structure. Then we design five models to run on the dataset. As the dataset contains 7 tables, we also perform the merging and feature engineering operations before running the models. We use a simple averaging strategy to ensemble the predictions and achieve a private score of 0.782.

In addition to running the models, we also try feature selection and dimensionality reduction methods to reduce the number of features. Based on the results we obtained, feature selection through co-linearity and feature importances can significantly reduce the number of features while maintaining the same AUC. On the other hand, PCA can not achieve satisfactory AUC compared with feature selection.

Acknowledgments

I read some Kaggle notebooks before finishing this project. The ideas and coding skills in the notebooks inspire me a lot. The links to the notebooks are listed below:

- `https://www.kaggle.com/code/willkoehrsen/start-here-a-gentle-introduction`
- `https://www.kaggle.com/code/tottenham/10-fold-ridge-from-dromosys-features-lb-0-768`
- `https://www.kaggle.com/code/aantonova/aggregating-all-tables-in-one-dataset`

References

[1] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, Tie-Yan Liu. "LightGBM: A Highly Efficient Gradient Boosting Decision Tree". Advances in Neural Information Processing Systems 30 (NIPS 2017), pp. 3149-3157.