Santander Customer Transaction Prediction

Songhan Lei, Wenyi Wang, Yutian Sun, Xuanyu Zhang

Presentation Video Link:
[Santander Customer Transaction Prediction | Kaggle Featured Prediction Competition]

Contents

Project Definition & Introduction

Santander is a global financial services company based in Madrid, Spain. Santander US has over 5 million customers and $155 billion assets. With a large amount of transaction data, Santander data science team is working on machine learning models to drive meaningful insights and predict the likelihood of their customers' behaviors, finally the whole organization can rely on those data science recommendations to achieve a long term goal: help its customers to understand their financial condition and choose the right products and services to achieve monetary goals.

Technically, the specific goal of this project is to build reliable binary classification models to predict if customers will make a specific transaction or not in the future based on their previous transaction records, irrespective of transaction amount.

The data source we used for this project is from Kaggle, both the training set and the testing set has 200,000 records with 199 anonymous variables. The data size for both training and testing sets is approximately 300 MB.

Solution Overview

The final updated model will be a binary classifier using LightGBM algorithm trained with some adjustments on the original dataset, including newly-added 200 magic features as well as the removal of 100,000 fake data in testing dataset. Sorting out fake data and feature engineering are the key to establishing a well-performing model. The huge similarities of basic statistics of training and testing data indicate duplication of the information. Also, we find that the distribution of unique values of specific features is quite different between training and testing dataset, indicating presence of fake testing data. To extract more informative elements from the dataset, we remove the fake data and create extra magic features. Our classification finally achieved an AUC of 0.901 on Kaggle's leaderboard.

Analysis

Methodology

The first step was to conduct a data preprocessing step to explore how the dataset looks and then conduct the data cleaning process, including checking missing values and detecting outliers. The second step was to do exploratory data analysis and try to extract some insights. Then, we worked on 199 unknown features to perform feature engineering and select informative features to prepare for the modeling step. We tried different algorithms and different feature engineering methods, and to compare the evaluation metrics on different models. Finally, we chose the best model based on AUC performance as our final model and submitted it on Kaggle.

Exploratory Data Analytics

From this part, we first checked the class distribution for our target variable, the result shows the majority (~ 90%) of transaction data belongs to class “0” , with “0” 179902 and “1” 20098 (Shown in Figure 1), which means most customers in the dataset are not willing to make a transaction. Therefore, the training dataset is highly imbalanced where the distribution across the label classes is biased, which will result in the poor prediction performance on the minority class. To tackle this problem, we used oversampling technique (SMOTE) to create new synthetic examples from the minority class and ensure the balanced class distribution between two classes. The overall record of the train dataset after oversampling is 359804, with both “0” and “1”

179902.

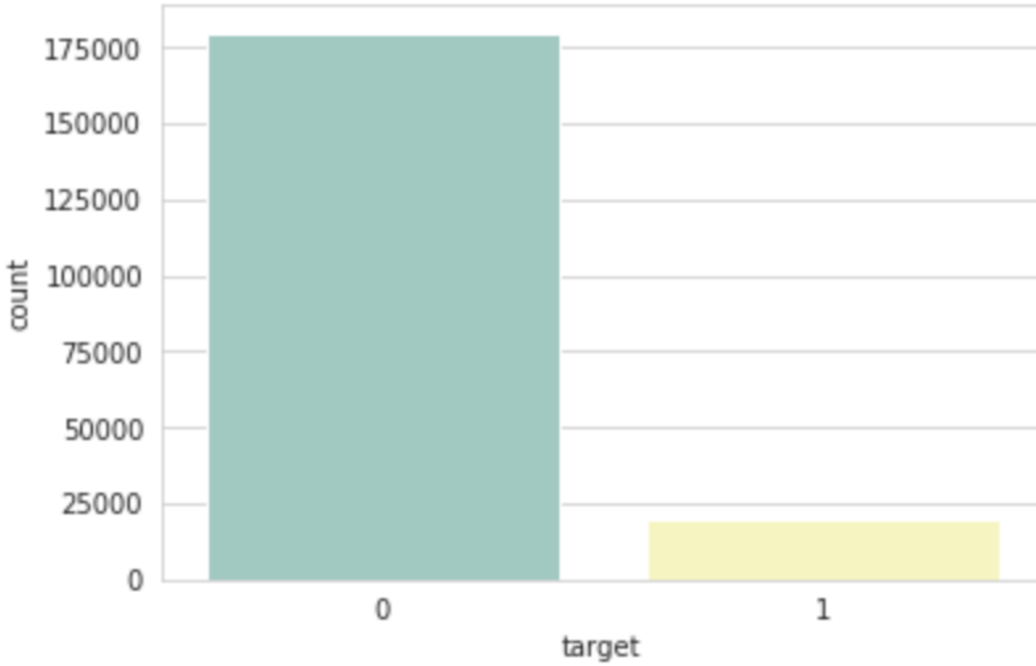


Figure 1

Next, we checked some basic statistics information for both training and testing dataset, we found an interesting insight: for the same feature between training set and testing set, almost all the statistics metrics are similar to each other, such as mean, standard deviation and different percentiles. This similarity is a signal that the testing dataset may have duplicated from the training dataset. (Sample shown in figure 2)

	target	var_0	var_1	var_2	var_3
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	0.100490	10.679914	-1.627622	10.715192	6.796529
std	0.300653	3.040051	4.050044	2.640894	2.043319
min	0.000000	0.408400	-15.043400	2.117100	-0.040200
25%	0.000000	8.453850	-4.740025	8.722475	5.254075
50%	0.000000	10.524750	-1.608050	10.580000	6.825000
75%	0.000000	12.758200	1.358625	12.516700	8.324100
max	1.000000	20.315000	10.376800	19.353000	13.188300

Statistic of train dataset

	var_0	var_1	var_2	var_3
count	200000.000000	200000.000000	200000.000000	200000.000000
mean	10.658737	-1.624244	10.707452	6.788214
std	3.036716	4.040509	2.633888	2.052724
min	0.188700	-15.043400	2.355200	-0.022400
25%	8.442975	-4.700125	8.735600	5.230500
50%	10.513800	-1.590500	10.560700	6.822350
75%	12.739600	1.343400	12.495025	8.327600
max	22.323400	9.385100	18.714100	13.142000

Statistic of test dataset

Inspired by the “List of Fake Samples and Public/Private LB split” solutions from YAG320 from kaggle, we checked the unique values for every feature between training dataset and testing dataset. The result shows the distribution of unique values is very different between training and testing dataset. Following this reference, we then made an assumption: the testing dataset synthesizes the real samples from the training dataset, which we could conclude part of the testing dataset is fake.

In order to find the real testing dataset and fake testing dataset, we followed the method shared by YAG320 to check the sample values for all 199 features. If more than one sample feature is unique, then the sample should be a real sample. We finally identified 100000 records(~50%) are fake samples while another 100000 records are real samples. Therefore during the modeling part, we chose the real sample data to perform modeling evaluation.

Feature Engineering

From this part, since there are duplicate values for each feature, in order to generate more informative features, we created one more new feature for each original feature to check if the value from that column is unique or not. So the first step of feature engineering is to create 200 more features, which are named *var_XX_unique_or_not*.

The second step of feature engineering was to select the useful features among those 400 features, we utilized *Mutual_info_classif* from sklearn to perform feature selection step, this key point of this method is to calculates mutual information value for each of independent variables with respect to dependent variable, and selects the ones which has most information gain. In other words, it basically measures the dependency of features with the target value. The higher score means more dependent variables. We set the threshold of 200, so we finally selected 200 more informative features as the model input.

Modeling

We totally select 4 models to predict whether customers will make a specific transaction in the future: LightGBM, Random Forest, Logistic Regression and K-nearest neighbors. The evaluation metric we choose is AUC.

Here are the results of those 4 models (Included 200 magic features in train dataset, removed 100,000 fake data in test dataset, conducting feature selection and oversampling)

Model	**AUC in Jupyter Notebook**
LightGBM	0.988
KNN	0.942
Random Forest	0.942
Logistic Regression	0.934

Under this current situation, our score on the Kaggle website based on LightGBM is 0.833.

Modeling Evaluation

Since we conducted several data engineering steps mentioned above, we want to check whether each step will increase the performance of models. Therefore, we decide to use our best model (LightGBM) to check the incremental or decremental effect of each step.

Feature Engineering	**Model Score (AUC)**	**Score on Kaggle**
Original train / test dataset	0.899	0.898
Include 200 magic feature in train dataset	0.903	0.899

Include 200 magic feature in train dataset + remove 100k fake data in test dataset	0.903	0.901
Include 200 magic feature in train dataset + remove 100k fake data in test dataset + feature selection	0.872	0.869
Include 200 magic feature in train dataset + remove 100k fake data in test dataset + feature selection + oversample	0.988	0.833

The results show that not the final step has the best performance, which may be due to overfitting after perfectly matching the distribution of "0" and "1". Therefore, we decide to use the dataset in the 4th step. Here is the result of each model under the 4th step.

Model	**AUC in Jupyter Lab**	**Score in Kaggle**
KNN	0.503	0.503
Random Forest	0.500	0.500
Logistic Regression	0.625	0.633

Here are some confusion matrix and classification reports of our models (dataset under 4th step):

Confusion Matrix of Logistic Regression

	1	**0**
1	1599	775
0	4427	53199

Classification Report of Logistic Regression

	precision	recall	f1-score	support
0.0	0.923	0.986	0.953	53974
1.0	0.674	0.265	0.381	6026
accuracy			0.913	60000
macro avg	0.798	0.625	0.667	60000
weighted avg	0.898	0.913	0.896	60000

Confusion Matrix of KNN

	1	**0**
1	78	405
0	5948	53569

Classification Report of KNN

	precision	recall	f1-score	support
0.0	0.900	0.992	0.944	53974
1.0	0.161	0.013	0.024	6026
accuracy			0.894	60000
macro avg	0.531	0.503	0.484	60000
weighted avg	0.826	0.894	0.852	60000

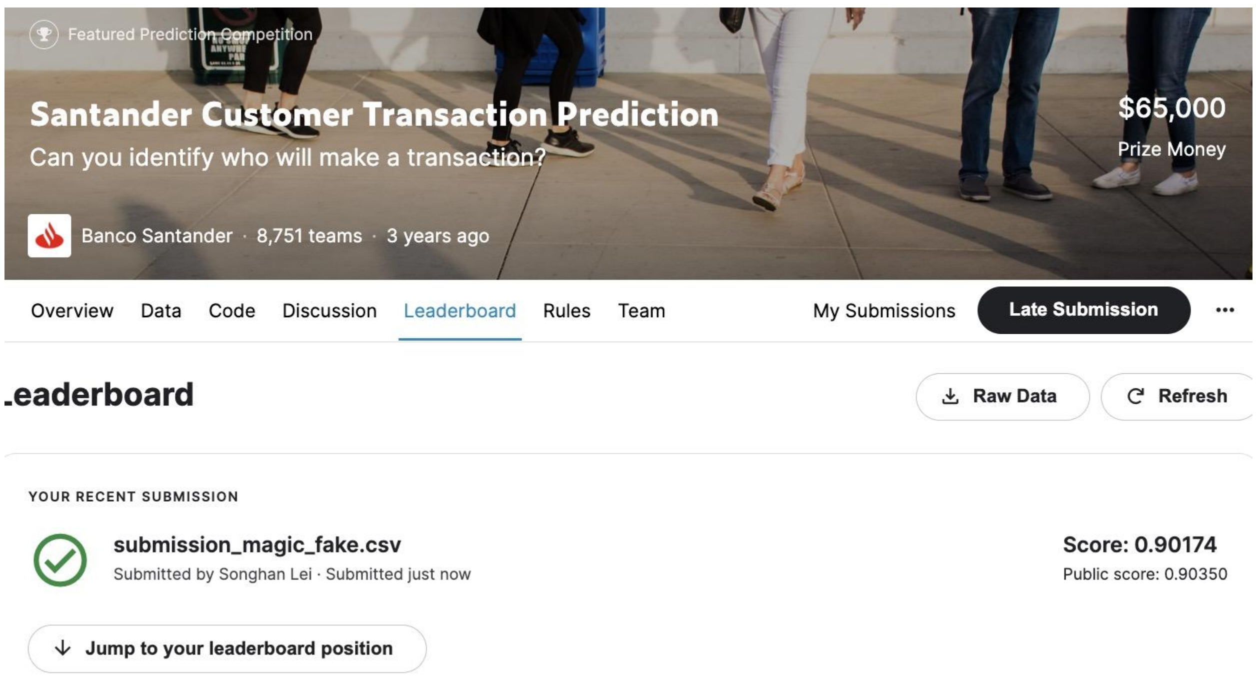
Confusion Matrix of Random Forest

	1	**0**
1	0	0
0	6026	53974

Classification Report of Random Forest

	precision	recall	f1-score	support
0.0	0.900	1.000	0.947	53974
1.0	0.000	0.000	0.000	6026
accuracy			0.900	60000
macro avg	0.450	0.500	0.474	60000
weighted avg	0.809	0.900	0.852	60000

Here is the screenshot of our best model result on Kaggle:



Reference

1. https://www.kaggle.com/code/yag320/list-of-fake-samples-and-public-private-lb-split