CS559 - Project Report: Santander Transaction Value Prediction

Team member: Junxin Xia, Bowen Lu

I. Introduction

According to Epsilon research, approximately 80% of customers are willing to do business with you if you provide personalized service. Banking is exactly the same. The digitalization of everyday lives, customers expect services to be delivered in a personalized and timely manner. The intention of our project goes beyond of determining whether or not a customer willing to making transactions with Santander service. We aim to predict the amount of customers' transaction values in a more concrete, but also a simple and personal way.

II. Dataset and Data Prepossessing

We are provided with two anonymized dataset: one for training (4459 records) and another for testing (49342 records). The training set has a column of "ID" (i.e. transaction ID) and a column of "target" (i.e. the real value of this transaction), while the testing set is given without "target" which needs to be predicted. The rest columns in both datasets are numeric but anonymized variables (4991 in total: 1845 of them have values in float type, and 3147 of them have values in integer type). They are candidate features for building the prediction model.

We draw a histogram (shown in Figure 1) of the target in training set, in order to figure out the distribution. The target follows a highly skewed distribution, therefore we make a log transformation (i.e. log(1+x)). The transformed value of the target seems more likely to be a normal distribution (shown in Figure 2).

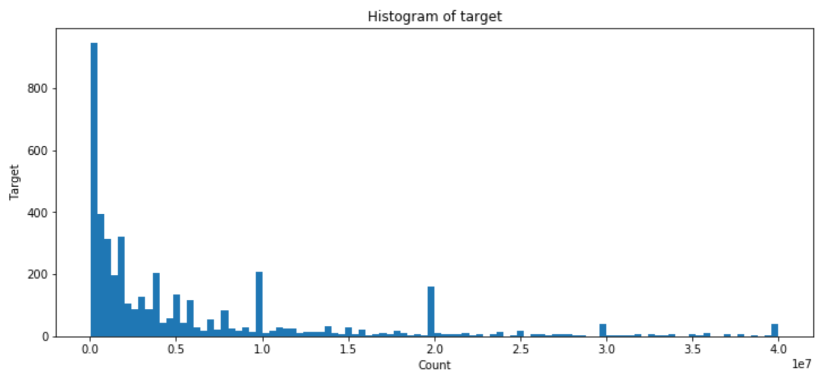


Figure 1

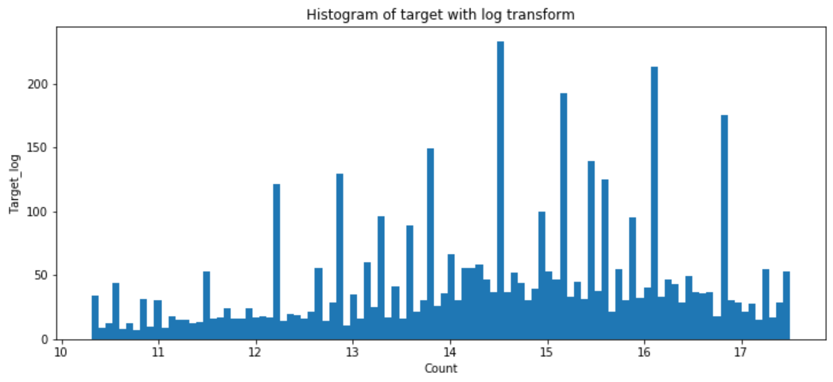


Figure 2

After a brief overview of the dataset, we found that the features are sparse since more than 95% of the values are 0, but luckily, none of the values is missed. Besides, the number of columns is more than that of rows in the training set, so it is vital to conduct feature selection and feature engineering properly.

First of all, we drop those features that only contain one unique value, which is detected by calculating the variance of each column. There are 256 features being dropped by this command. Also, five columns are discovered to be duplicates, so we drop them as well.

After that, we conduct feature scaling to standardize the range of features, with the method of min-max normalization. The variance of each remained feature is plotted in Figure 3. It is clear that only a small part of features have significant variance compared with the rest of so-called "trivial many" features.

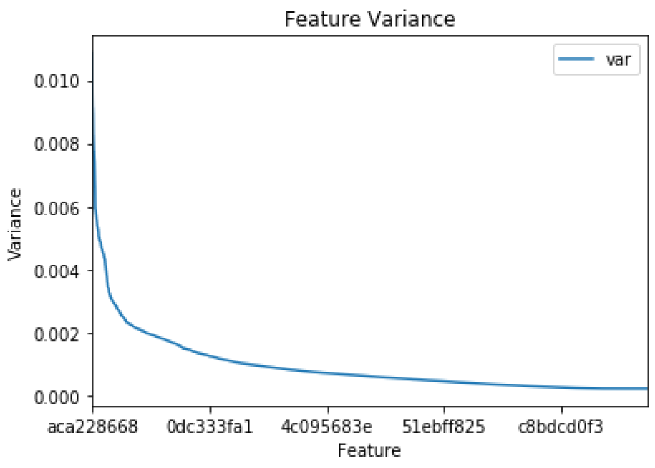
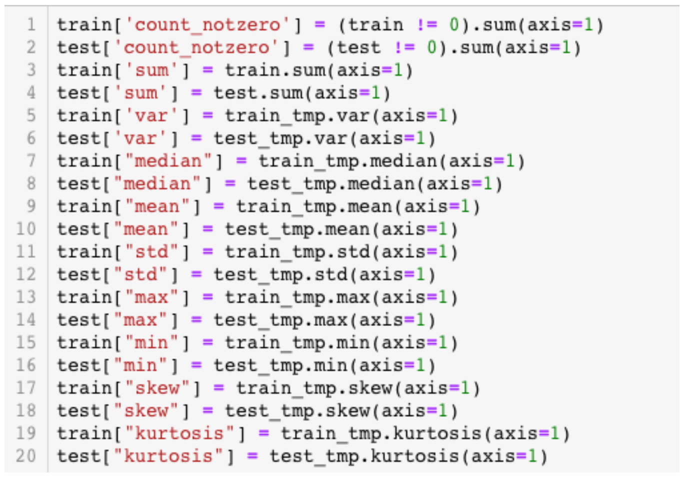


Figure 3

We also utilize Random Forest Regressor for feature selection. Random Forest Regressor is a meta estimator that fits a number of regression decision trees on various sub-samples of the dataset and uses averaging method to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the sample are drawn with replacement if using the bootstrap strategy. In addition, when splitting a node during the construction of the tree, the split that is picked isn't the best split among all features. Instead, the split that is picked is the best split among a random subset of the features.

As a result of this randomness, the bias of the forest usually is slightly improved, but its variance is decreased due to averaging. By doing so, random forest algorithms could compensate for the increase in bias, therefore yielding an overall better model. So we used Random Forest Regressor algorithms to fit into training dataset and see degree of importance each column has.

By putting all of results into dataframe, we sorted value by degree of importance and picked top 1000 out of it. Based on these chosen 1000 fairly important features, we conducted feature engineering using various statistical method like "sum, variances, median, standard deviations, min and max". Through this step, there are a few more columns added into training dataset and testing dataset.



```
train['count_notzero'] = (train != 0).sum(axis=1)
test['count_notzero'] = (test != 0).sum(axis=1)
train['sum'] = train.sum(axis=1)
test['sum'] = test.sum(axis=1)
train['var'] = train_tmp.var(axis=1)
test['var'] = test_tmp.var(axis=1)
train["median"] = train_tmp.median(axis=1)
test["median"] = test_tmp.median(axis=1)
train["mean"] = train_tmp.mean(axis=1)
test["mean"] = test_tmp.mean(axis=1)
train["std"] = train_tmp.std(axis=1)
test["std"] = test_tmp.std(axis=1)
train["max"] = train_tmp.max(axis=1)
test["max"] = test_tmp.max(axis=1)
train["min"] = train_tmp.min(axis=1)
test["min"] = test_tmp.min(axis=1)
train["skew"] = train_tmp.skew(axis=1)
test["skew"] = test_tmp.skew(axis=1)
train["kurtosis"] = train_tmp.kurtosis(axis=1)
test["kurtosis"] = test_tmp.kurtosis(axis=1)
```

Figure 4

III. Principal Component Analysis

Principal component analysis is a dimension-reduction method that can be used to reduce a large set of variables into a smaller set of variables, which still leads to contain most of the information. PCA is an unsupervised technique. The main goal of a PCA analysis is to identify patterns in data. PCA aims to detect the correlation between variables. If a strong correlation between variables exists, the attempt to reduce the dimensionality only makes sense. In a nutshell, what PCA does is to find the directions of maximum variance in high-dimensional data and project it onto a smaller dimensional subspace while retaining most of the information. So the main advantages of PCA are data decomposition(reduce memory, speed up learning) and data visualization.

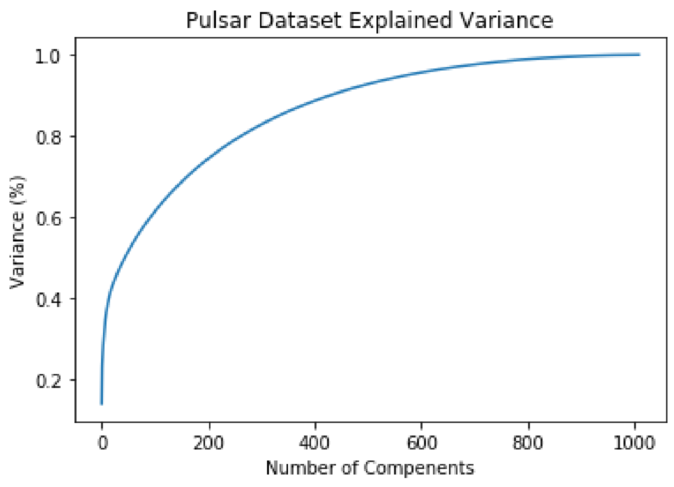


Figure 5

IV. Machine Learning model

After the feature is ready, we can using machine learning algorithms to fit the whole training dataset and make predictions.

1. Linear Regression (simple algorithm)

Linear Regression, intuitively is a regression algorithm with a Linear approach. We try to predict a continuous value of a given data point by generalizing on the data that we have in hand. The linear part indicates that we are using a linear approach in generalizing over the data. The cost function of Linear Regression is attached below. We normally use gradient descent to gradually minimize this function.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

2. Boosting (advanced algorithms)

Boosting algorithms is a series of ensemble algorithms that train a strong classifier by combining weak classifiers with iterated weights and run them in multiple time. In each iteration, the misclassified samples are given weight by how incorrectly they are classified. In our project, what need to be predicted is a continuous variable, and boosting is also capable to handle such regression problems. We use three of the most advanced boosting algorithms, which are XGBoost, LightGBM, and CatBoost, to predict transaction values.

XGBoost is one of the most popular machine learning algorithm these days dealing with structured data. XGBoost is an implementation of gradient-boosted decision trees designed for efficiency of compute time and memory resources. Some of the key algorithm features includes: sparse aware implementation, continued training, and block structure to support parallelization of tree construction.

LightGBM is also a gradient-boosted decision tree framework. It grows tree vertically (leaf-wise, split only one node and grow at a single level), while other tree-based algorithms grow tree horizontally (level-wise, split all nodes and grow at a single level). Leaf-wise algorithm can reduce more loss than a level-wise algorithm as the size and complexity of dataset keep increasing. LightGBM also performs with high speed and lower memory request.

CatBoost is a gradient boosting algorithm that work well with multiple categorical data, so that any explicit data prepossessing that converts categorical value to numerical value is no longer needed. It provides state-of-the-art results without extensive hyper-parameter tuning required by many tree-based algorithms, which also lowers the chance of overfitting.

V. Performance Measurement

The evaluation metric for this project is Root Mean Squared Logarithmic Error. Models with lower value of RMSLE achieve higher accuracy of prediction.

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

(n is the total number of observations in the training set. p_i is the prediction of target, and a_i is the actual target)

We calculate RMSLE of each machine learning algorithms as the number of principal components we select increasing. The performance result is shown in Figure 6.

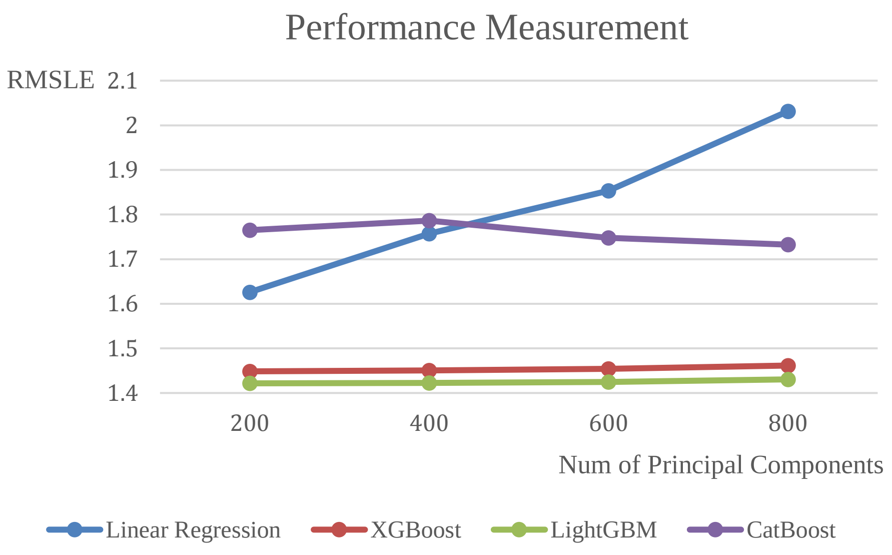


Figure 6

Advanced algorithms as XGBoost and LightGBM have lower RMSLE compared with Linear Regression, but CatBoost is an exception. The reason why CatBoost performs not as well as expected could be due to the fact that there is no categorical feature in the dataset, so the advantage of CatBoost has less impact on enhancing prediction accuracy as other boosting algorithms do.

Another discovery is that when the number of principal components we choose to project the features increasing, the performance of XGBoost and LightGBM seem to be robust because RMSLE not change significantly. In contrast, the increasing of number of principal components have significant impact on the prediction accuracy of CatBoost and linear regression. RMSLE of CatBoost decreases gradually and RMSLE of linear regression keeps increasing.