CSCI 567 Spring 2016 Mini Project

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this report, we describe our approach to building data models to predict if a customer is satisfied or dissatisfied with their banking experience at Santander Bank. The raw dataset and problem statement is taken from Santander Customer Satisfaction competition on Kaggle [1]. From front-line support teams to C-suites, customer satisfaction is a key measure of success. Santander Bank asked Kagglers to help them identify dissatisfied customers early in their relationship. In this competition, we had to work with hundreds of anonymized features to predict if a customer is satisfied or not. First, we introduce the problem and describe the dataset. Then, we identify the crucial pre-processing steps to transform the data. Next, we explain the Feature Engineering techniques and also the machine learning models we used for this problem. Finally, we discuss the results along with some insight into how the performance can be further improved.

1 Introduction

In this problem, we attempt to predict if a customer is satisfied or dissatisfied with their banking experience at Santander Bank. The dataset is anonymous and contained a large number of numeric variables. The "TARGET" column is the variable to predict. It equals one for unsatisfied customers and 0 for satisfied customers. The task is to predict the probability that each customer in the test set is an unsatisfied customer. Our first basic approach was to start with a simple logistic regression model and check its performance. But eventually, our research led us to ensemble models [2].

2 Data

The data consisted of multiple sets of values corresponding to the features. Every feature was identified by a unique id. A set represents the values for the different features for a particular customer with that sets' expected value.
The training dataset consists of values for 76020 customers. Each of these has 371 observable features. This set is the observed values for different features for the customers. Our test data consists of values for 75818 customers. Two of the important features are:
• ID - unique value for every customer.
• Target - 0 if customer is satisfied and 1 if customer is not satisfied.
The test dataset is same as the training dataset, but the Target field is missing. We were estimating this field.

3 Feature engineering

In the given data, there was a lot of it missing or zero. This meant that we needed to select only those features which are important and which would help us make a good prediction. So the first step we

took in feature engineering was to remove all those features which had zero variance, because all such features don't really provide us with any useful data about the customer. Second, we found all the features that had duplicates, i.e., identical columns. We eliminated all the duplicates and maintained only one of the features.

3.1 Monotonous feature transform

We discovered three types of features in general. The first type has binary distributed values represented by zeros and ones only. The second type has distribution with sharp peak on zero and very stiff hump aside from it and the third has $lambda$-like distribution decay, with sharp peak near one and long tail towards positive direction of value axis (see Fig. 2). For the latter two cases we have decided to conduct logarithmizing procedure according to the following formula:

$$\forall i \in [1, ..., N]\, x'_i = \log\left(x_i - \min\left(x_i\right) + 1\right) \tag{1}$$

Where N is the number of features. As a result we got the value histogram bins separation for nearly 10% of cases which allowed us to extract more data from our features. We have also demonstrated the important feature '*var38*'(number 370) takes a closer to Gaussian pmf instead of λ-like decaying distribution that it had earlier. This, however, does not guarantees the quality improvement of our classification. Noteworthy, that such methods like XGBoost[5,6] and random forests[4] use decision trees and compare relative feature values, rather then inferring some numerical dependencies, like in case of linear or kernel regression.

3.2 Feature selection

Additionally, we have tried variety of feature selection procedures. We started with uncorrelated methods that included Pearson correlation, Mutual information(MI) and maximal information coefficient (MIC). We also tried linear regression approaches, like Lasso and Ridge regression. Finally, we invoked the tree based methods Stability Selection[7] and Recursive Feature Elimination.

3.2.1 Uniform correlation

The Pearson Correlation does not treats non-linear dependencies, so we discarded it right away. We used MIC to evaluate the correlation of key features like '*var15*', '*var36*' and '*var38*'. The 'minepy' toolkit was used to build a 307 by 307 matrix of non-zero unique feature correlations.

3.2.2 Linear methods

The linear Ridge and Lasso regression needed to have some extra tunning, we were not available to get a meaningful results using them. This in conjunction with the fact those methods are not the best in terms of feature selection made us to move on to better methods.

3.2.3 Random forest based methods

Random forests regression became our most used method. We used 2500 short 4-level depth classifiers to have enough averaging on the feature ranking. We have used the resulting weights to transform feature matrix that we used further. The resulting amount of features was 211 and the loss in AUC(area under receive operation curve) was around 0.8%. This method seemed to be the best in terms of computational cost and reliability to cut off the non-relevant features, despite of its overly aggressive reweighing(see Fig. 1).

3.2.4 Stability and recursive method

Stability methods and recursive techniques are claimed on the top of other approaches in terms of results quality. However, in out case, they were hard to use, since iterating though every feature and classifying the reduced dimensions would take days, if not weeks of computing even using significantly decreased number of classifiers and tree depth.

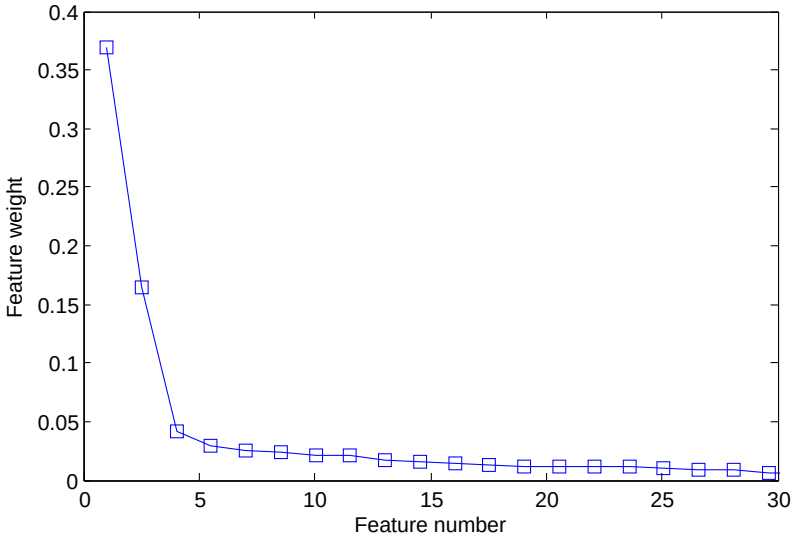

Figure 1: Random forest regression feature weighting. We can see a characteristic drop-down in weights after intitial few features.

4 Models and algorithms

We used three models in our research: logistic regression, random forests and XGBoost. We spend more time on the latter model ,since it was giving us substantially better results, compared to the other two. Due to time limitations we were not able to invoke all feature selection techniques. The classification and tuning hyper parameters were computationally expensive and took around one hour for every run. We were using 'sklearn' library, a Pyhton-written package that includes most of up-to-day tools for statistical research.

4.1 Logistic regression

We have tried logistics regression as a preliminary step for our calculation. We started with Our best AUC value was 55.7862. The precision was limited due to the necessity of introducing 1.0 L2 penalty due to tendency towards overfitting and very noisy data. With using PCA for the dimensionality reduction to 200 and 100 features we obtained the AUC result of 57.3154 and 58.1877. Since the limited capacity of the method for treating the non-linear effects we have not spend much time on it and moved to higher order methods: Random Forests and XGBoost.

4.2 Random forests

The random forests technique is based on the decision trees [2], but it uses 'bootstrap aggregating'-type approach for averaging many different estimates. It decorrelates the base-level learners by learning trees based on a randomly chosen subset of input variables. The overall approach can by described by the following formalism:

$$f(x) = \sum_{m=1}^{M} \frac{1}{M} f_m(x) \tag{2}$$

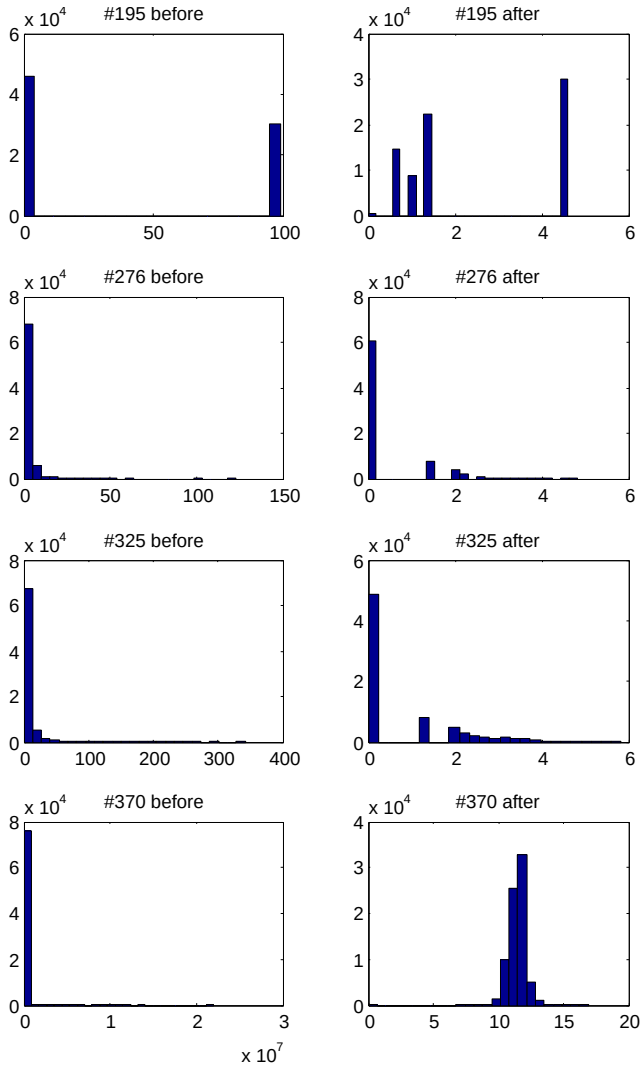

Figure 2: Comparing features distributions before and after logarithmizing

Where f_m is the m-th tree. Another name for this approach is 'bagging' introduced by Breiman et al. in 1996.

We tried 100, 200, and 300 estimators for random forests approach with the results with results of 75.0085, 0.764076 and 0.770102.

4.3 XGBoost

This was our best model which gave the highest score of 0.842629 on the public leader-board. The 'sklearn' XGBoost wrapper was the only was able to handle missing values. It handled missing values by automatically finding its best imputation value based on the reduction on training loss. We believe that this was the deciding factor in this model being the best because a lot of training data had missing values and this model was able to treat it appropriately. However, this result was achieved with no appropriate feature selection. We would trust our other result of 0.837902 result more, since it was obtained after feature selection and should be less over-fitted.

5 Results

To generate the sample solution, you will need all the necessary machine learning packages: 'sklearn', 'Numpy' and 'scipy' for Python installed. After that executing the script script.py will generate the required submissio'n file. We achieved our current best scores using XGBoost library and obtained a Public leader-board score of 0.842629 and a Private leaderboard score of 0.824047.

6 Perespective improvements

It should be noted that our private results were way worse than public. As a lot of other teams we encountered severe over-fitting. While doing many-fold cross validation helps to reduce the over-fitting we should have tried to use some L1 and L2 penalties and try to vary them to figure out, at which value we have a reasonable trade-off between AUC loss and over-fitting. Another thing we can try is to implement recursive feature selection, using some reference, more high-performance method, like SVM and then use for main final classification. Another idea that came to us was running K-means with 3-7 classes over positive and negative points, allotting equal portion of each class into each fold during cross validation. Lastly we can add another classifier to our 'bag'. We consider the optimal start in this direction to be an XGBoost with 500-600 classifiers and 5-6 depth trees and convolution neural net of 5-8 depth.

[1] Kaggle - Santander Customer Satisfaction - *https://www.kaggle.com/c/santander-customer-satisfaction*

[2] Kaggle Ensembling Guide - *http://mlwave.com/kaggle-ensembling-guide/*

[3] Breiman, Leo (2001). Random Forests. *Machine Learning* 45 (1): pp. 532

[4] Kevin P. Murphy (2012). Chapter 16. Additive basis funciton models. *Machine Learning. A Probabilistic Perspective.* MIT Press

[5] Hastie, T.; Tibshirani, R.; Friedman, J. H. (2009). 10. Boosting and Additive Trees. *The Elements of Statistical Learning (2nd ed.).* New York: Springer. pp. 337384.

[6] Friedman, J. H. Stochastic Gradient Boosting. (March 1999) Stanford, Department of Statistics (web)

[7] Nicolai Meinshausen, Peter Bhlmann (2009). Stability selection. *Journal of the Royal Statistical Society: Series B* 72(4) pp. 417-473