Kaggle Competition

Hyesuk Byun
University of Southern California
Los Angeles, CA 90089
hyesukby@usc.edu

Abstract

This paper presents an evaluation of various machine learning techniques applied for the Kaggle competition on Santander customer satisfaction as the author competed in it under the team name "hbyun" The goal of the competition is to develop a binary classifier to be used to classify unlabeled test data, and to maximize its Area Under the Curve (AUC) score, given a training data set consisting of 70k labeled samples. The biggest challenge for this competition is that the training data set is extremely imbalanced: negative samples comprise 96%, and positive samples comprise only 4% of the entire data set. In addition, the number of samples in the training data set is only about 70k.

First of all, in order to take the problem of imbalanced data, we have tried and evaluated SMOTE combined with undersampling. Second, we have experimented a variety of machine learning algorithms spanning logistic regression, SVM, neural network, and various ensemble learning methods, represented by gradient boosting. We present our experience of using hand-coded implementation of one of these algorithms, and how it is compared to using an off-the-shelf implementations of them, scikit-learn. Third, we present our experience of parameter tuning for the gradient boosting method we chose as our final learning algorithm.

Using high valued regularization terms we obtained from the parameter tuning process, we have built a binary classifier with a reasonably low bias & very low variance: our model's final rank on the private Kaggle leader board was 355 out of over 5000 competing models whereas its public rank was about 1700.

1 Problem Definition

The goal of the Kaggle competition on Santander customer satisfaction is to develop a binary classifier using a public training data set consisting of 76k labeled samples, and a developed classifier is evaluated using its Area Under the Curve (AUC) score from private, unlabeled test data.

1.1 Training Data Set

The biggest challenge for this competition is that the given training data is extremely imbalanced: negative samples comprises 96%, and positive samples comprise only 4% of the entire data set. Without proper preprocessing of the data, many existing machine learning algorithms tend to train a model that overfit for the *majority* class.

The training data has 369 features, and the meaning of each of the features is undocumented although the features' names are given in the training data set. Therefore, it is very difficult to perform feature engineering due to the lack of knowledge on the relation between the features and the label. Due to this problem, we pursue little on feature engineering for introducing an indicative new feature that may well work with a simple machine learning algorithm such as logistic regression.

The training data is sparse. We found that many samples (¿ 20) have zero-filled feature data and more than 20 of them have very little variance between samples. We consider this during pre-processing of the data.

Lastly, the the number of samples in the training data set is only about 76020. A small amount of training data can be detrimental to application of sophisticated machine learning algorithm such as neural network that is known to work very well with an enormous amount of training data up to a couple of billion samples.

1.2 Area Under the Curve (AUC) Score

If we use the count of correctly labeled samples as an evaluation metric for a binary classifier in the setting of imbalanced data, this can be problematic. For example, we can think of a fake classifier that classifies any given sample to be in the majority class. This classifier will show 96% of accuracy in our given training data set.

Area Under the Curve (AUC) Score can more fairly evaluate a classifier over imbalanced data by penalizing a model whose prediction is incorrect on samples from the minority class. Here, the "Curve" stands for an ROC curve, where the X-axis represents A/B where A is the number of mislabeled samples from the majority class, and B is the total number of samples in the majority class, and Y-axis represents C/D, where C is the number of correctly labeled samples from the minority class, and D is the total number of samples in the minority class. AUC score is defined as the area under this ROC curve. Intuitively speaking, the wider this area becomes, the more accurately we label *both* samples from the minority class, and the majority class without being biases by the sizes of the classes.

The Santander competition is using this AUC score as the main evaluation metric, so we need to build a classifier that correctly label samples from both classes equally.

2 Preprocessing

The given training data set consists of 76k samples with 360+ features. The dataset have feature values ranging from -9999999 to a few thousands. With processing of this dataset, we can expect training algorithms to work in a more stable way and run fast too. First, with proper preprocessing, we can guarantee numerical stability (not causing floating-point-related error while the computer performs floating point number calculations) as the training algorithm handles the data. Second, if we preprocessing the data so that we can remove unnecessary features, we can run a training algorithm faster on the cleaned-up data. In this section, we explain data preprocessing methods we experimented as we participated in the competition.

2.1 Min Max Normalization

This preprocessing transforms features by scaling each feature to a given range. This method scales and translates each feature separately such that it is in (0,1). For min/max normalization, we transform each feature data by:

$$X_{std} = (X - min(X))/(max(X) - min(X))$$

3 Standard Normalization

This preprocessing method normalizes each feature data separately so that the data in each feature has zero mean and unit variance. This normalization is mostly commonly used in various settings.

4 Removing Features with Low Variance

We found that many feature values in the training data set has very low variance (¡ 0.01 after standard normalization is applied). We can remove those features so that a learning algorithm can run faster on more compact data set without being affected by noisy feature data.

Table 1: Summary of preprocessing parameters and description

Sampling Part		
Name	Description	Value
SMOTE	Oversampling minority class sample	$4-10$
Undersampler	Undersampling majority class sample	ratio:1
Scaling and feature selction Part		
Name	Description	Value
Scaler	Standard scaling, min-max scaling	
Selector	Feature selector that removes all low-variance features.	$0.005-0.05$
PCA	Dimension reduction: principal component analysis	$2-20$
Clean-up	Remove constant feature column, change unreliable number (-999999)	
Additional features	Add number of zeros as new feature column	

5 Principal Component Analysis (PCA)

When a machine learning algorithm deals with data with many features, it may ignore some representative feature or relation between features. We apply principal component analysis (PCA) to generate *new features* as principal components generated by PCA, and add them as artificial features to the original dataset. Then, we can apply a learning algorithm on the enhanced dataset with an expectation that the algorithm can work more effectively.

6 Learning Algorithm

Many learning algorithms have their own strengths and weaknesses in terms of their accuracy and speed and their efficacy on different types of training data. We have experimented a variety of learning algorithms : 1) logistic regression 2) SVM 3) multi-layer Perceptron (neural network) 4) gradient boost. This section gives a brief overview of each of the algorithms, and why we tried them.

6.1 Logistic Regression

We have applied the simplest form of logistic regression using linear regression without any feature engineering except that we applied PCA during the pre-processing process. Using logistic regression, the learning algorithm accepts the training data and tries to fit a linear expression for the data using gradient descent. This can be the fastest algorithm among all other algorithms we tried, and it has a nice feature of showing how much accuracy improvement is actually made each step of gradient descent. However, without further feature engineering, it would be very hard to expect this to perform very well since most realistic training data cannot be modeled using only a linear expression.

6.2 Support Vector Machine (SVM) with Non-Linear Kernel (RBF)

SVM can be a very effective way to model a data set that cannot be modeled by a linear expression using kernels. However, we found that many SVM algorithms do not give its prediction in terms of the probability that each sample belongs to either the positive/negative class. In addition, an approximation algorithm to infer the probability can be inaccurate too.

6.3 Multi-layer Perceptron (neural network)

Neural network has been known to be very effective to model non-linear hierarchical training data especially when the amount of training data is enormous (e.g., billions of click-through data samples used at Google). In addition, it scales very well since it can run in parallel. However, it may be hard to apply on our training data set since its sample number is small (76k), and a massive cloud computing environment is unavailable to us to make it scalable.

6.4 Gradient Boosting

Gradient boosting is an ensemble learning method by which we create many weak models such as logistic regression, or decision trees, and combine them to create a stronger single model. Gradient boosting starts by creating some weak models, representing a single model combined. At each step of its learning process, it tries to minimize the combined model's mean square error (predicted_probability - real_label)**2 for all samples) by adding new weak models to fill the error between the current prediction and the real label. Since this error correction (boosting) is comparably similar to gradient descent, this algorithm is called "gradient boosting". This algorithm is known to be very powerful for applications where the number of samples in a training data set is not very many and SVM does not work well/doesn't fit for its purpose. As we explain in later sections, gradient boosting turned out to be chosen as our main learning algorithm for the competition.

7 Implementation & Evaluation

7.1 Implementation

We first tried logistic regression using matlab program. Basic logistic regression output scores of 79, however, it was less flexible in terms of changing parameters. Scikit-learn is free software machine learning library in Python. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN. Thus, it was very useful to test various kinds of algorithm to find the most suitable algorithm for our dataset. We finally chose gradient boosting, which is a principled method of dealing with class imbalance by constructing successive training sets based on incorrectly classified examples. We have implemented our learners in Python, mainly based on the Scikit-learn library. The following diagram shows the data flow of our implementation. The results of each implementation are summarized in Figure 3.

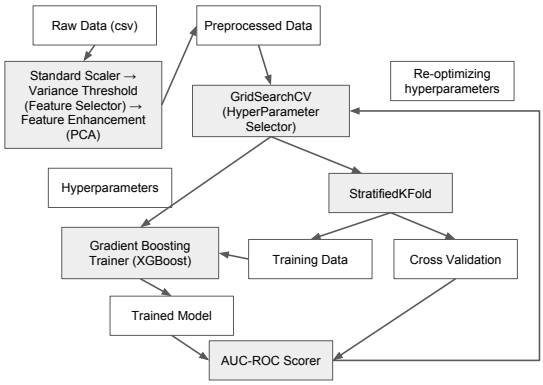

Figure 1: Work-flow diagram of the finalized framework from raw data to the classification output.

7.2 Evaluation

Although we are using built-in library, it was not easy since the XGboost has more than 20 parameters that need to be tuned. There is general approach to tune manually, however, manual tuning can lead to a wrong direction and especially we found that there can be a correlation between parameters, thus we cannot just compare two parameters at a time to find optimum set of parameters. Figure 2 shows an example relationship of two parameters. The subsample and colsample by tree is correlated, but not in a linear form whereas the relation between min child weight and max depth is weak. Thus we tried exhaustive grid search (GridSearchCV), but because of limiting computing power we couldn't proceed the billion possible combination of parameters. We ran RandomizedSearchCV, where a fixed number of parameter settings is sampled from the specified distributions. The number of parameter settings that are tried is given by n iter.

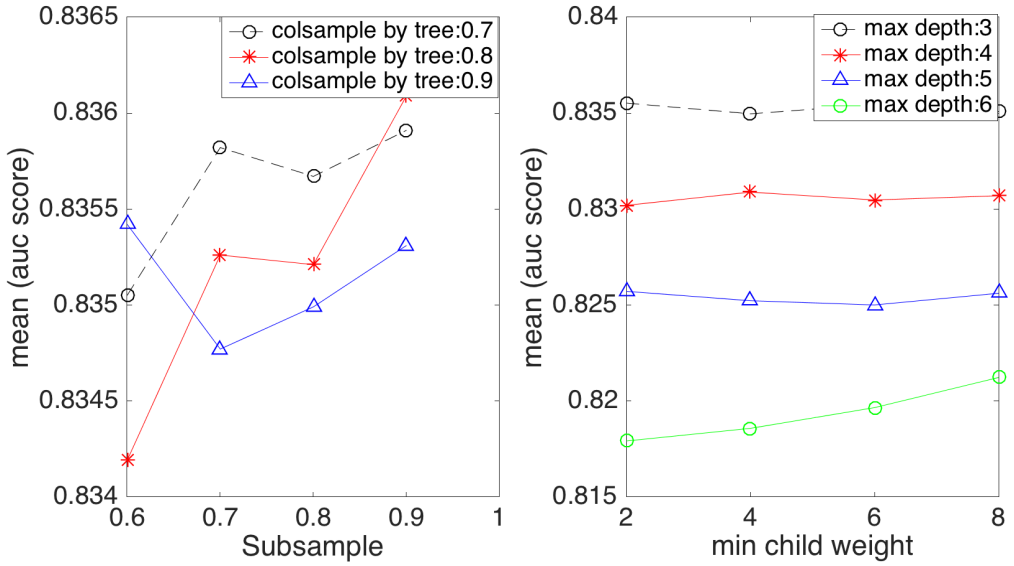

Figure 2: Bar chart plot summarizing classification AUC-ROC performance across classifiers

Table 2: Gradient boosting (xgboost) classifier parameters used

General Parameters		
Name	Description	Value
Booster	tree-based models	default
nthread	maximum number of threads	8
Booster Parameters		
Name	Description	Value
min child weight	minimum sum of weights of all observations required in a child	18
max depth	maximum depth of a tree	3
gamma	makes the algorithm conservative	0.0
max delta step	helps in logistic regression when class is extremely imbalanced	4
subsample	fraction of observations to be randomly samples for each tree.	0.8
colsample by tree	fraction of columns to be randomly samples for each tree.	0.8
colsample by level	the subsample ratio of columns for each split, in each level	0.6
reg lambda	L2 regularization term on weights	1
reg alpha	L1 regularization term on weight	110
scale pos weight	it helps in faster convergence in high imbalance data	24
learning rate	step size of convergence	0.05
Learning Task Parameters		
Name	Description	Value
objective	logistic regression for binary classification, returns predicted probability	binary:logistic
eval metric	metric to be used for validation data	auc
seed	The random number seed	auc

8 Results and Discussion

8.1 Results

Figure 3 shows the summarized auc scores for each classifier with multiple choice of parameter set. As it shows, we found the best parameters from gridsearchCV and applied them to gradient boosting

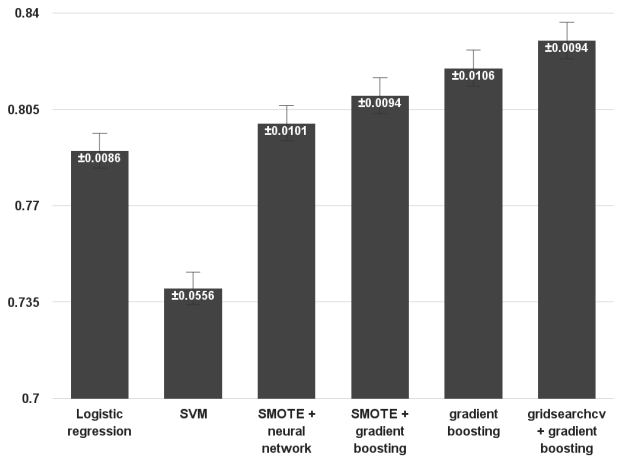

Figure 3: Bar chart plot summarizing classification AUC-ROC performance across classifiers

at the last stage of our evaluation. We ran a set of experiments to obtain highest CV score and public leader board score using ensemble of xgboost model.

8.2 Discussion

In Santander competition, training dataset was highly skewed. On the other hand, test set seemed well-balanced between negative class and positive class. Oversampling, undersampling techniques, and gradient boosting approach with high regularization terms in our model can prevent the model from overfitting to the majority of the data set. And the public leader board only shows the result from half set of test data, therefore we cannot only rely on public leader board score, thus we set the rule to choose our final two submissions: select the one that produced the highest 10-fold cross validation score, and the one that gave us highest leader board score. As we expected, actual auc scores dropped significantly from 0.84x to 0.82x due to the reason mentioned above. From the experience on this competition, we learned that the more complicated model cannot be better than the simpler one in a highly skewed real world data.

References

[1] Qazi, N., & Raza, K. (2012, March). Effect of Feature Selection, SMOTE and under Sampling on Class Imbalance Classification. In Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on (pp. 145-150). IEEE.

[2] Wasikowski, M., & Chen, X. W. (2010). Combating the small sample class imbalance problem using feature selection. Knowledge and Data Engineering, IEEE Transactions on, 22(10), 1388-1400. Chicago

[3] Padmaja, T. M., Dhulipalla, N., Bapi, R. S., & Krishna, P. R. (2007, December). Unbalanced data classification using extreme outlier elimination and sampling techniques for fraud detection. In Advanced Computing and Communications, 2007. ADCOM 2007. International Conference on (pp. 511-516). IEEE.

[4] Chen, T., & He, T. (2015). xgboost: eXtreme Gradient Boosting. R package version 0.4-2.