Stats 154 Final Project Report

Minsu Kim, Albert Lee, Yunhao Zhang

1. Introduction

The training set we are originally given is a refined dataset[1] with 50,000 observations, each of which has binomial sentiment information (1 for positive and -1 for negative sentiment) and word counts from 1,000 predictors. The test dataset has 100,000 observations with the same 1,000 predictors extracted from the original 1,578,627 tweet dataset. The sentiment of the tweets in the refined dataset is fairly evenly distributed, 50.11% positive, and 49.89% negative, respectively.

the goal of the project is to predict the binomial sentiment of individual tweets from the test dataset. However, the Kaggle competition provided the first half of the test dataset (50,000 tweets). The performance is measured by accuracy, which is equivalent to 1-misclassification rate.

The team is currently *Rank 1* on the leader board. As we are writing this report, we are still running models in hope of higher accuracy.

2. Feature Engineering

Before we were provided a bag of words by the GSI, we implemented our own version based on natural language processing tutorial provided by Kaggle, and compared its quality with the bag of words from the refined dataset. In order to measure the quality of these two versions, we tried random forest with the same parameters on both bags of words several times, and obtained prediction accuracy of 76% and 72% from refined dataset and our own version respectively. Similarly, we obtained prediction accuracy of 78% and 76% from Extreme Gradient Boosting. From this result, we concluded that the provided dataset was already good enough and hence, feature engineering was not necessary. After implementing the GSIs version of a bag of words, we experimented several different versions of a bag of words by increasing the number of predictors from 1000 to 7000, and examined whether more features improve prediction accuracy. Because we have enough observations, we believed that increasing the number of predictors by 1000 each time does not result in overfitting or curse of dimensionality. The following is the improvement in prediction accuracy generated by Extreme Gradient Boosting (Xgboost).

[1]Thanks to Hoang's python codes that helped converting the tweets to*bag of words*

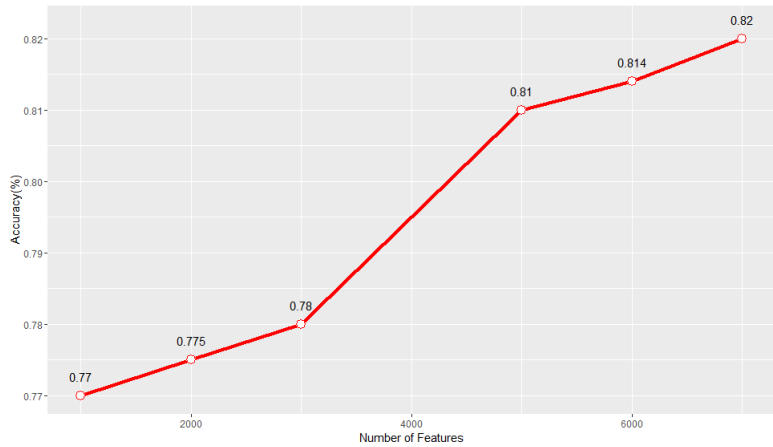

Figure 1: The Effect of Increasing Number of Features on Prediction Accuracy

The figure above apparently indicates that prediction accuracy increases with the number of features. Therefore, we decided to put more effort on model selection, parameter tuning, and model stacking with different number of predictors.

3. Models

3.1. Basic Models

As a preliminary step, we ran 4 different individual machine learning models on the 50,000 observations of refined dataset to check their respective accuracy without hyper-parameter tuning. The performance was measured through cross validation, in which we assigned 40,000 (80%) randomly chosen observations as training set, and the remaining 10,000 observations as hold-out dataset.

1. XGBoost: A very popular model in Kaggle Competitions. We used the version provided by GraphLab, and wrote a scikit-learn wrapper around it for consistency with the rest of the models.

2. GBM: This is a representative of *Boosting*.

3. Random Forest: This is a representative of *Tree* method and *Bagging*.

4. Sparse Logistic:It is simple! We also assume that not all features (words) will be useful in terms of prediction.

The respective accuracy from these models on a hold-out set are[2]:

XGBoost performed the best on the hold-out dataset, so our team tried the model on the actual test set in Kaggle, which gave 76.9% accuracy. This confirmed XGboost performed as well in the actual test set as in the hold-out set, and the hold-out set is representative of the actual test set. The result led to two following approaches: hyper-parameters tuning of

[2]These are the results after optimal tuning

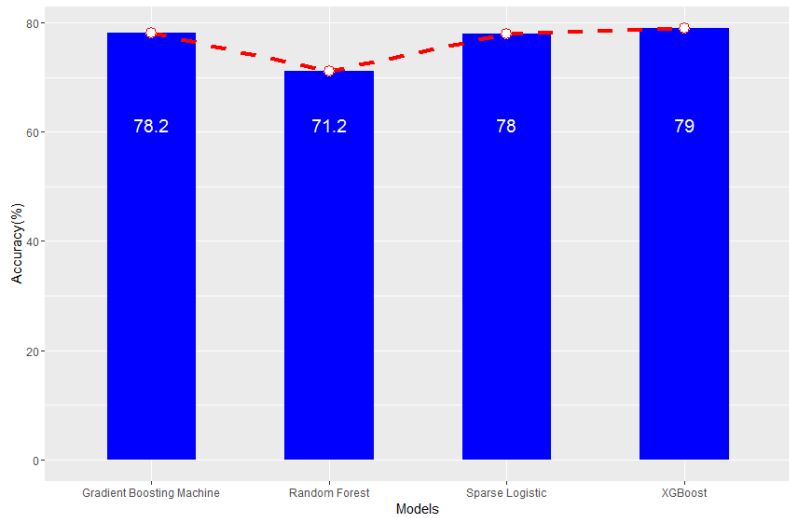

Figure 2: Accuracy of Different Basic Machine Learning Models

XGBoost and ensemble method.

3.2. Hyper-Parameter Tuning & Data Processing

In order to find the optimal parameters, we decided to utilize the grid-search method[3]. However, due to its high computational cost, we thought it was infeasible to try all possible parameters with the different number of features. Thus, we chose XG Boost for grid-search because it results in the best prediction accuracy and runs relatively fast. We ran a grid-search on datasets with 1000 and 2000 predictors and examined whether optimal tuning parameter between these two are different. It turned out that optimal parameters for both datasets were the same except for maximum depth and the number of estimators. Also, we adhere to xgboost and further tuned maximum depth for each dataset. The following is the hyperparameters we tried.

In order to run this grid-search, we utilized AWS (Amazon Web Service) C4 High-CPU Eight Extra Large[2] service on EMR. Finally, we obtain optimal parameters in the following.

Table 1 shows the (local) optimal parameters for each dataset.

[3]132 Compute Units (ECU), 36 cores. For more specification of the service, visit: http://www.ec2instnnces.info/?selected=c3.8xlarge.

Hyperparameters	Grid	Description
Learning rate	[0.01, 0.05, 0.1]	Step size
n_estimators	mean.auc.test	Mean auc score
max_depth	[7, 9, 11, 13, 15, 17]	Maximum depth of tree
subsample	[0.5, 0.7, 0.75]	Subsample proportion
colsample_bytree	[0.1, 0.5, 0.6]	Column sample proportion

Table 1: List of Hyper Parameters and Grids Tested for Parameter Tunning

	Learning rate	n_estimators	max_depth	subsample	colsample_ bytree	auc
1000 features	0.05	3200	11	0.75	1	0.844
2000 features	0.05	4000	11	0.75	1	0.85
3000 features	0.05	4549	13	0.75	1	0.86
5000 features	0.05	5979	15	default(1)	default(1)	0.88
6000 features	0.05	6000	15	default(1)	default(1)	0.885

Table 2: Hyper Parameter Grids and Number of Features Tested for Parameter Tunning

3.3. Ensemble 3.3.1. Averaging

We want to combine XG Boost and Sparse logistic models by applying optimal weights[4] on their predicted probabilities. (Again we set up the grid search that loops over different weights, calculate and compare the resulting accuracies.) The optimal weights for XG Boost and Sparse Logistic are *0.8*and *0.2* respectively, and the accuracy on the hold-set is still about 0.78, indicating no significant improvement from XG Boost.

3.3.2. Stack Generalization

We experimented 2-layer stacked generalization, motivated by its superior performance at various Kaggle competitions such as *Amazon Employee Access Challenge*. We also learn the method from *Wolpert 1992*. Setting the threshold for Positive at 0.5, we obtain the final predictions. However, the accuracy achieved is still around 79%, indicating no significant improvement from benchmarks. The following chart is the logic flow of our implementation:

[4]That gives the best accuracy

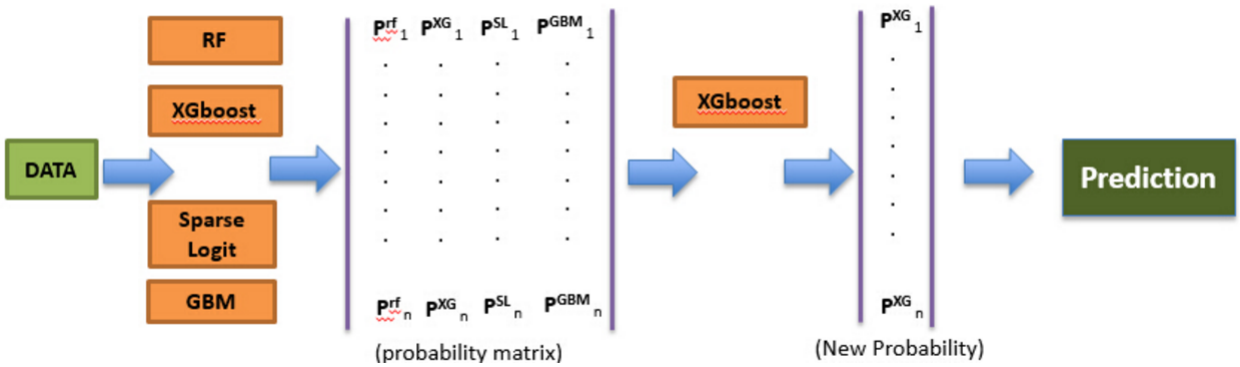

Figure 3: The Pipeline of the Stacked Generalization

4. Conclusion

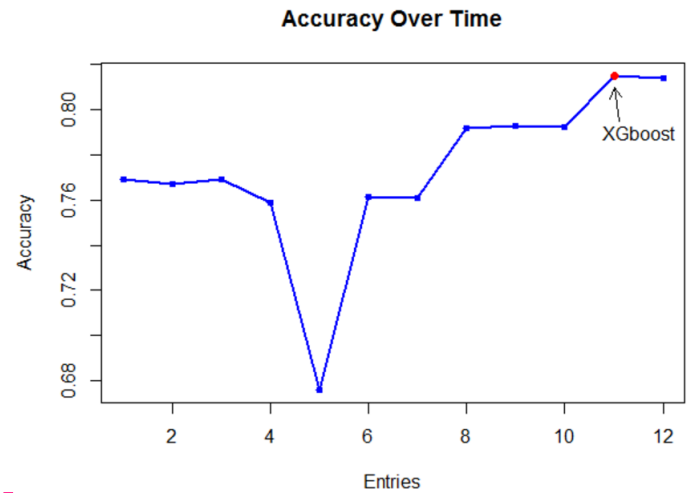

Figure 4: Change in Accuracy over Time

Since *XG Boost* seems to have the best performance so far, we apply *XG Boost* to the full data set, which contains 1,578,627 rows (tweets) and at least[1] 5000 columns (features) [5] . As data size increases, we expect our final prediction result to be better than the current accuracy 81.51% on Kaggle test submission. Finally, we provide a plot of accuracy for each test entry.

One important takeaway from this project is that prediction accuracy increases with the size of representative training data set.

[5]All accuracies presented eelow arb tuned to the respective model's optimum

References

1. Wolpert, D., Stacked Generalization., Neural Networks, 5(2), pp. 241-259., 1992
2. Hastie, Trevor, Robert Tibshirani, and J. H. Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction: With 200 Full-color Illustrations. New York: Springer, 2001. Print.