Tim Mango

Predicting Income with Census Data

Introduction:

The goal of this paper is to predict whether individuals have an income equal to or greater than $50,000 a year based on information gathered from the census.

Dataset:

The data set used in this paper is from the UC Irvine Machine Learning Repository. A large data set was selected from the library to be the basis of a classification problem. I chose the largest data set possible in an attempt to prevent overfitting by my selected model. A data set summary can be viewed in R the file using the view(dfSummary(dataset)) command. The data set has 42 columns, 199,523 rows, and no missing data. Since there is no missing data, tree based methods are a good option for this data set.

Model selection:

In my group project, Predicting Bankruptcy with Financial Statement Information, we used a Naïve Bayes classifier and a Random Forest method to try and predict companies that would go bankrupt. The data used for the group project was also from the UC Irving Machine Learning Repository. The creator of the Bankruptcy data set, Sebastian Tomczak, wrote a separate paper comparing models and their performance on predicting bankruptcy with the Bankruptcy data set.

While my groups' models were very successful, they were outperformed by Sebastian Tomczak's best performing model, the Extreme Gradient Boosting model. Extreme Gradient

Boosting is used for supervised Machine Learning problems and is one of the highest performing models in Kaggle competitions. This paper uses a tree base method in the XGBoost package that operates similarly to Random Forest, but is boosted for better results. The model is built with the XGBoost package to minimize computational cost.

Data Preparation:

In order to maximize functioning of the XGBoost package the final data should be separated into a matrix of training data that excludes income labels and a matrix of testing data that excludes income labels. All columns in the matrix must have a numeric value.

Hot commands can be used to turn factor variables into numeric variables. A hot command will take a factor variable with n categories and then create a column of dummy variables for each category of the factor variable.

Finally, the training data and training labels are put into a xgb.DMatrix for training, named dtrain, and an xgb.DMatrix for testing, named dtest. This is the final form of the data and ensures that the XGBoost model will be able to run as quickly as possible.

Validation:

This model is computationally expensive and can take a while to train, retrain, and improve performance. An example of this retraining process is shown below. I started with a tree based model that has a max tree depth of 3 and 1 boosting iteration. The boosted tree based model had a max tree depth of 8 with 100 boosting iterations. The boosting was commanded to stop if there

was no further improvement after 8 consecutive iterations. The boosted model continued to improve for all 100 iterations.

Results:

The results of the performance of the two models are shown below using the same training data and the same testing data. The boosted model reduced the test error rate by 29.41% when compared to the original model's test error rate. It is worth noting that the boosted training model ended with an error rate of 0.02824 on its 100^{th} iteration, but only achieved a testing error rate of 0.04344. This could be a sign that the boosted model may be overfitting the data. However, the boosted model's performance on the testing data was still better than the un-boosted model. The final boosted model predicts if an individual will have an income of $50,000 or more annually with 95.66% accuracy.

Figure 1:

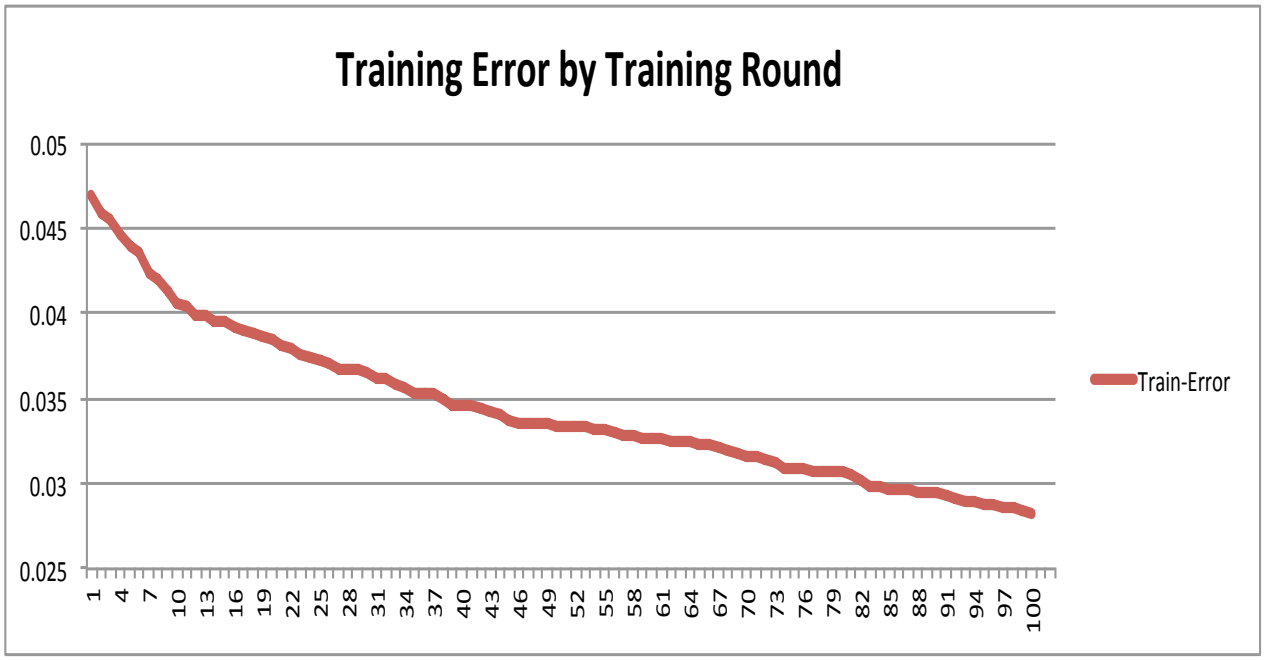

Figure 2:

Model	Test Error Improvement Percentage: 29.41%	Accuracy Improvement Percentage: 1.36%
Model	0.05622	0.9438
Boosted Model	0.04344	0.9566