Kaggle my Baggle

Alex Kellogg, Bill DeRose, Jacob Fiksel, Lingge Li

December 4, 2013

1 Introduction

Loan defaults played an integral role in the collapse of the financial industry in 2008. In the midsts of a housing bubble, banks wanted their share of the profits and so engaged in misdirected and extremely risky lending practices (subprime mortgages, for example, which involve lending money to borrowers with lower credit ratings). As more banks participated in this type of lending, the competition for credit grew fierce and institutions further relaxed their standards of credit worthiness. Banks lent too much money to the wrong people. Lenders finally faced the consequences when many of those subprime loans were not repaid; in the ensuing months the United States faced the worst recession since the 1930s.

In the wake of the subprime mortgage crisis banks and regulators began paying special attention to the risks associated with lending. Enter Kaggle's "Give Me Some Credit" competition where we are provided with data on 150,000 loans. Among the information provided about the borrowers are revolving utilization, monthly income, debt ratio, age, number of dependents, number of times late in paying loans, and number of real estate loans. With data in hand, the task is then one of binary classification; our goal is to predict a borrower's probability of default.

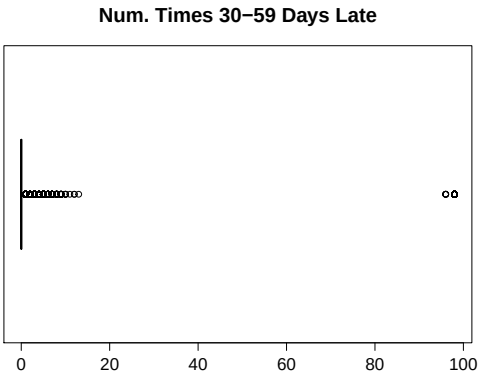
2 Clean Up

The first, and arguably the most important, thing we did was explore the data and attempt to draw meaning from the variables. This led to our discovery of outliers in the data and, in some cases, alerted us to corrupt data. We also noted the presence of many `NA` values, an issue we certainly needed to address if we wanted reliable predictions.

2.1 Outliers

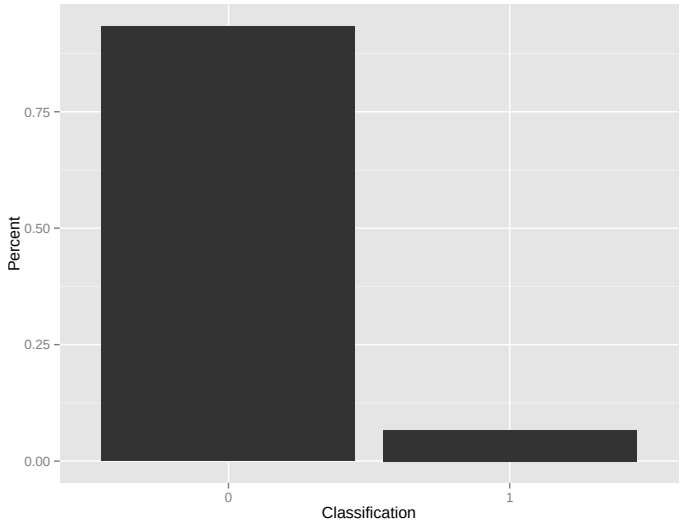
After plotting our data and examining summary statistics, we noticed a number of outliers in the data. For example, someone had age 0 and another was making several million dollars in monthly income (what industry is that in, please?). The person with age 0 is easy to deal with, but in other cases we had to more judiciously choose a path of action. Debt ratio was off the charts for observations where monthly income was `NA` . We proceeded to set both values to `NA` if either was missing and leave the rest to imputation.

Another anomaly was the presence of nonsensical values 96 and 98 in the predictor variables. After doing some digging, we realized that these quantitative values were used to code for qualitative values such as "failed to answer or "not applicable". We simply replaced these observations with `NA` and let the imputation deal with the rest.

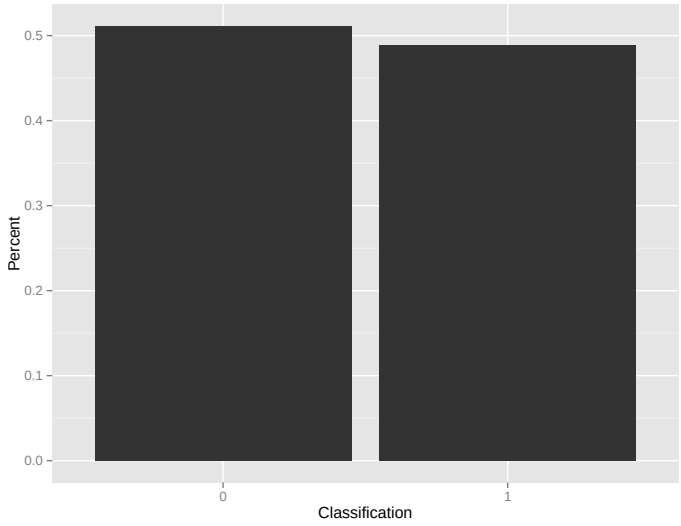


2.2 Skew

When we began our work we were pleasantly surprised to see one of our classifiers correctly predict 93% of the observations. Upon inspection, this result is nothing to be excited about; in fact, it is not much better than guessing at random because at least 93% of the data are all 0's.



We noticed a large number of false negatives (the worst kind of error in this case), and decided that it would behoove us to train more of the data on people who did indeed default. Thus we decided to selectively sample our training set so defaulters and non-defaulters were more equally represented:



As noted in our concluding remarks, we did not have time to find an optimal default-to-non-default ratio. Our results were based purely on the belief that the two classes should be equally represented in the training data.

2.3 Imputation

Many individuals in the data set were missing important predictor values. Instead of ignoring blank values, and thus potentially ignoring values that could be key in classifying individuals, we decided to choose between two imputation methods: K-nearest neighbors imputation, and gradient boosting imputation.

To evaluate the two methods, we focused on computation time and perceived accuracy of the methods. The K-Nearest Neighbor computes a distance matrix, and finds a specified number of closest observations to the individual you are imputing the data for. It then finds the mean of the variable you are imputing from those individuals. Unfortunately, the algorithm in R was unable to compute a distance matrix for a data set of the size we are working with, which would force us to use much smaller subsets to impute the data, thus

decreasing the accuracy of our imputations. Gradient boosting, the method that we used for our data, treats each missing value as a regression problem and uses boosting to create regression trees and predict the value as a weighted sum of the predictions. The weights are created to minimize a loss function, thus improving accuracy.

While using regression to find missing values for our data set may not have been the best model, gradient boosting allowed us to use the training data to impute the missing values in the test data, and was significantly computationally less expensive than k-nearest neighbors imputation.

3 Classifiers

In general, we did not use one single classifier. We realized that for any classifier, there were myriad parameters that we could try to tweak to get things exactly as we wanted. Instead, we opted to use the classifiers "out-of-the-box" and combine their results in an intelligent way. Below, we discuss the specifics of each classifier and the reasons for using them as such.

3.1 Naive Bayes

The naive bayes algorithm is a good example of a simple classifier we used to compute our final probabilities. It uses a strong assumption that each class contributes independently to the final classification, and thus is "naive." For each data point it computes the posterior probability of the individual being in a certain class by taking the product of the prior probabilities and the likelihood functions of the data. It classifies the individual into the group with the highest posterior probability. The advantages of this algorithm is that it is computationally fast, and does not require a large training set.

3.2 Random Forest

From our lectures and research, we knew that random forests would perform well given the number of observations and dimensionality of our predictor variables. In addition, they are not phased by the presence of missing data; this was a concern of ours early on before we had given though to imputation techniques. Random forests are themselves powerful ensemble learners that use forests of decision trees to vote on the final classification label. Given their ability to scale with data, excellent predictive abilities and easy of use, random forests were an obvious choice for us.

3.3 Boosting

Our final classifier is also an ensemble learner that uses the adaBoost algorithm with decision trees as the weak classifiers being trained. Again, the simple ease of use and predictive power drew us towards boosting as a solution.

By default, the `ada` package uses trees from the `rpart` library. We played around with different trees, including stumps, to use in the learning step. Ultimately we settled on using trees with 16 terminal nodes, a result backed by our submission results and not by any sort of cross validation.

The adaBoost algorithm is sensitive to outlying data and may be prone to overfitting. In fact, we observed artifacts of overfitting when our classifier would perform very well on our training data and extremely poorly on the data Kaggle uses to verify our results. We attempted to remedy this issue by combining all classification results using a genetic algorithm.

3.4 Genetic Algorithm

4 Conclusions

All in all, this project was extremely enjoyable, and presented the challenge of analyzing a vast amounts of data in figuring out the best way to predict defaults. Although the more successful methods were addressed in the various sections of this paper, we learned a lot of valuable lessons in our numerous errors.

At first, we didn't pay much attention to the data we were dealing with, figuring that our classification methods would be able to sift through all the outliers and come to a reasonable estimate. However, after a quick set of box plots, we quickly changed our minds about the significance of our dataset, and went a little over the top by setting values of data that we found moderately sketchy to `NA`, or adjusting them down to "more reasonable" values. After testing our results, we learned that too much data manipulating was a bad thing, and we backed off a bit once more.

In terms of implementation of the code, this project gave us a great opportunity to work with a particularly interesting data set. As a result, we had to seriously consider computational complexity,(running KNNImputation, for instance, was ruled out under this criteria), a concern new to us. Learning to use imported libraries proved to be an interesting challenge: for the majority of the work in this class, we had been implementing our own algorithms in R, and working with pre-made packages required a new approach in order to get our classifiers.

As a group, we are extremely pleased with our performance – having jumped from around 600th place to 149th was an incredible leap made in just a couple of days! However, as we concluded coding, we realized that with more time, there were some additional things we would have liked to try. We believe that the values that we train our data on are

extremely relevant to correct classification, and thus would have liked to focus a bit more energy on imputation: although we are satisfied with how `gmbImpute` worked for us, when we browsed the imputed values of our data, we noticed some seemingly strange outputs. Clearly, we can't be sure that these imputed values were off, but when comparing them to `KNNImpute` (which, mind you, only could handle very small subsets of the data), we seemed to be able to better rationalize the `KNNImpute` data than the imputed values we ended up using using.

We would also have liked to spend more time dealing with the skewness of our data. As was mentioned earlier, 96% of our data was from non-defaulters. In dealing with this, we explained that we chose to make a vector of probabilities to give more weight to observations of defaulters. However, we chose these probabilities arbitrarily; in order to even our data up, we usually trained our classifiers on data that was split 60-40 between defaulters and non-defaulters. Thus, if we had more time, we would be interested in solving for the optimal weights.

That being said, we are very pleased with our results. Here's a link to our github repo, if you want to check out our code: `https://github.com/billderose/KaggleCredit.`