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 engaged in this type of lending, the competition for credit grew fierce and institutions further relaxed their standards of credit worthiness. Banks were lending too much money to the wrong people. Lenders finally faced the consequences when many of those subprime loans were not repaid and in the ensuing months, the United States faced the worst recession since the 1930s.

As a result of this default spiral, banks and regulators have begun paying special attention to the risk involved in lending. Enter Kaggle's "Give Me Some Credit" competition where we are provided with data from 150,000 loans. Among the information provided about the borrowers we have availible the 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 binary classification with the goal of predicting a borrowers' probability of default.

2 Data Cleanup and Scrutiny

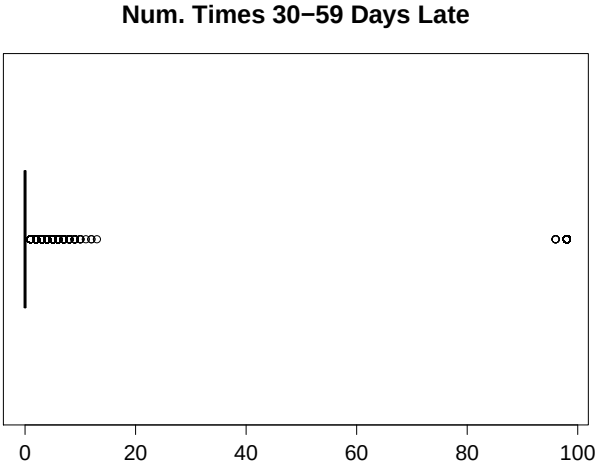
The first, and arguably the most important, thing we did was examine and interpret the data. It did not take long to realize that there are a plethora of NA values, an issue we would certainly need to address in order to have better predictions (see Imputation Section). After a little more scrutiny in the form of creating box plots of each of the respective variables, we noticed a number of outliers. For example, someone was aged 0 and another was receiving several million dollars in monthly income (what industry is that in, please?). As a group, we decided to look into each of the variables and really think about the feasibility of the outlying values that we were observing. In doing so, we came to a couple of major realizations that helped boost our Kaggle score. The first key observation, and certainly the most significant of the three, was that 96% of our data had a value of 0 in the field which we were to predict. Thus, each time we trained our data, the majority of the training was done on observations that would ultimately yield a value of zero (ie we could be 96% correct by guessing 0s for all values). We were noticing a large number of false negatives (predicted not to default, but did indeed default 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 weight the observations with a value of 1 in serious delinquency in 2 years more heavily than those with a 0. The way we did this was by creating a vector of probabilities, and assigning observations with a value of 1 a higher probability of being selected than those with a value of 0. In manipulating these weights, we were able to take subsets of our data (with resampling) with 40% of this training data having defaulted. Next, we observed a dependency between debt ratio and monthly income (the former is a function of the latter). We had noticed, in both the box plots and a scatter plot of debt ratio vs monthly income, that low and NA values of monthly income were highly associated with large debt ratios. Our main concern was with

the NA values which produced a large percentage of the outliers, and so we proceeded to replace the debt ratios given to NA for every given monthly income value of NA. This got rid of a large number of the number of outliers.

3 Clean Up

3.1 Outliers

We begin our exploration of the data with summary statistics and plots. One of the first anomalies we noticed was the presence of the 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.



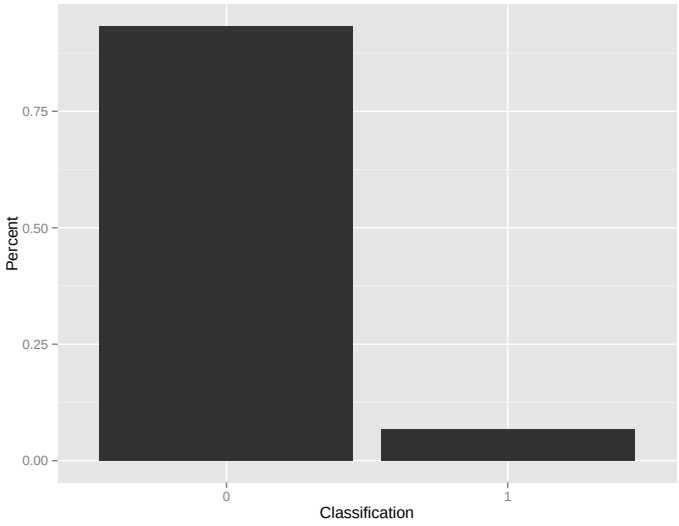
3.2 Imputation

Some individuals in the data set did not have values for certain values. Instead of ignoring blank values, and thus potentially ignoring values that could be key in classifying

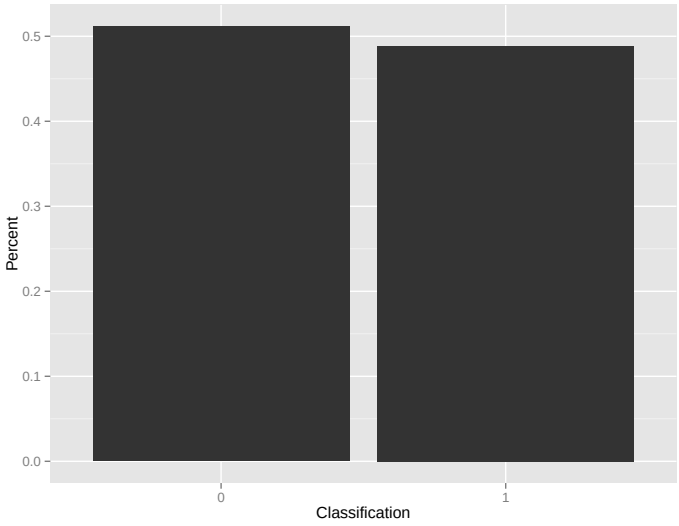
individuals, we decided to choose between two imputation methods: K-nearest neighbors imputation, and gradient boosting imputation. To evaluate the two methods, we focused on computation time and perceived accuracy of the methods. The K-Nearest Neighbor computes a distance matrix, and finds a specified 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.

3.3 Skew

When we began working on the classification problem we were pleasantly surprised to see one of our classifiers correctly predict 93% of the observations. Upon inspection, this number is not much better than guessing at random because at least 93% of the data are all 0's.



As we continued our work, we realized that our false positive rate was extremely high. We were not able to classify defaulters very well. We thought that this might be the result of too few defaulters in the training observations. To remedy this 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 pick an default to non-default ratio. Our results were the result of our belief that the two classes should be equally represented in the training data.

4 Classifiers

In general, we did not stick to a 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 as they come"out-of-the-box" and combine results in an intelligent way. We wound up using a naive bayes classifier, a random forest, a support-vector machine, and boosted trees to predict defaults. We then fed those predictions to a genetic algorithm to do the "intelligent" part and figure out how to best weight each prediction.

4.1 Naive Bayes

A simple algorithm we used to contribute to our final classification was a naive bayes classifier. 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.

4.2 Random Forest

Given the size of our data, we wanted a classifier that would scale well. From our lectures and research, we knew that random forests would perform well given the number of

observations and predictor variables. In addition, random forests 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. We also knew that random forests are very good at these types of problems and wanted to at least include them in our meta-ensemble.

4.3 Boosting

Our final classifier in the ensemble is itself an ensemble learner. We made use of the adaBoost algorithm which uses boosted regression trees. We used the `ada` package whose `ada` function we used right out of the box. We played around with the types of trees used to train the classifier. By default, the function uses random forests from the `rpart` library. We tried stumps and a few other types of trees and ultimately settled on using trees with 16 terminal nodes. We did not make use of any cross-validation to make this decision, but instead relied on empirical results.

4.4 Genetic Algorithm

5 Conclusions