STATS 542 Final Report: Analysis of Kaggle Springleaf Data

Nathan Walter (walter9)
(Dated: December 2015)

I. INTRODUCTION AND MOTIVATION

Hot or cold. Dark or light. Cat or dog. Since the beginning of time humans have sought ways to classify our world. Putting everything into order has always been a prerogative. With technological advances, we seek to have these same classifications made by computers. From the electronic sorting of mail to the likelihood an individual will develop a certain disease, statistical analysis of large datasets plays an important function in today's world. Boiling down a large data set into small pieces which can be used to make informed decisions is one of the best ways analysis of large data sets is used to improve society.

This project is attempting to use all of the principles learned in this course on some real data. Compared to the previous data we have analyzed, this data set is larger and less well posed. This means that any number of elements could be useless features or empty space. The data from this project is supplied from a corporation using a previously finished Kaggle contest asking for classification results on some data supplied by the Springleaf Corporation. Typically, these competitions have cash prizes to the team who can create the best models. In this case the Kaggle site is used merely to benchmark our work against that of the previous competitors.

II. PROBLEM FORMULATION

The data being analyzed was acquired online from Kaggle [1]. The problem is to develop a model to predict customer response to mail offers from Springleaf. The problem is a binary classification problem. The trained data is composed of 145232 observations of 1934 variables. Of the 1934 variables, one variable is the observation ID, which contains no predictive power, as each observation has a unique ID. The train data contains one response variable, labeled target. Thus, there is 1932 predictive variables and one response variable. The test data is composed of 145232 observations with the same 1932 predictive variables and one ID variable. The 1932 variables are composed of numeric variables, Boolean (binary) variables, character variables, categorical, variables, and dates/times. The data is composed of around 0.5% NA or missing values. Missing values are considered NA. The response variable is composed of 0 and 1s.

After reading many suggestions on the Kaggle forums, we decided to attempt two different strategies on this data analysis. One is using a very simple model, in this case linear, and more complex forms of feature selection. The other method is starting from the best given model from the forums, using the package xgBoost, and tweaking different parameters to see if the results can be improved. All given results in this project will be given in terms of the score recorded by the Kaggle competition site.

III. DATA PREPROCESSING

In this section, we discuss the measures used to pre-process the data so that the data used is well posed. First, we remove the useless variables from the data set, so the ID variable is removed, and the target variable is moved from the predictors. Next, we check the variables for variables with only one value, for example one variable had only NA as every entry. We remove these variables because they have no predictive power. Next, we check for variables that are dominantly NA. We remove variables that have composition of 15% or greater NA entries. We thought that 15% was a sufficient amount of missing values to corrupt the models we used. Further,

we removed the variables VAR_0212, VAR_0227, and VAR_0228 These were removed based on the condition that these variables were composed of 80-90 % unique values, and other than one or two NA values, VAR_0227 and VAR_0228 were identical. We removed these variables because they appeared to have no predictive power and appeared to be some sort of ID not an actual indicator of anything. After this initial variable removal, we had 1917 variables remaining, thus we had removed 15 extraneous variables.

Next we assume all character variables are categorical variables and convert them to numerical levels instead of characters, this is because packages such as R's biglm cannot handle character variables or levels, so we need numerical entries. Further, we note which variables are dates/times and which variables are categorical. We also convert dates/times to categorical variables. Last, we convert all remaining NA,NAN,INF values to -1 for numerical variables.

IV. METHODS AND PACKAGES

In this report, we use the full least squares model. This method is the standard least squares method but includes all predictors. The other methods are used to reduce the number of unnecessary predictors, so this method is a bench mark to compare to. The least squares method approximates the response as

$$\hat{Y} = \beta_0 + \mathbf{X}\hat{\beta} \tag{1}$$

where $\mathbf{X}$ has p (predictors) columns and n (observations) rows. Thus, with this method, there will be one intercept term and p coefficients. This equation is solved such that the residual sum of squares is minimized, resulting in minimal error between the predicted values and the actual response values. This is solved by computing the coefficients as

$$\hat{\beta} = (\mathbf{X^T X})^{-1}\mathbf{X^T Y} \tag{2}$$

under the assumption that $\mathbf{X^T X}$ is non-singular. Because of the size of the data set, the standard R package "stats" function lm cannot be used on the full data set without either a powerful computer or long computation time. Thus, we use the package biglm for linear regression.

Further, we implement lasso and ridge regression on the data set. Lasso and Ridge work similarly as full least squares except that they penalize the least square regression. Rather then simply minimize the residual sum of squares, ridge regression adds a term based on the L2 norm. The error for ridge regression becomes

$$|\beta_0 + \mathbf{X}\hat{\beta} - \hat{\mathbf{Y}}|^{\mathbf{2}} + \lambda|\hat{\beta}|^{\mathbf{2}} \tag{3}$$

Thus, the values of the coefficients become

$$\hat{\beta} = (\mathbf{X^T X} + \lambda\mathbf{I})^{-1}\mathbf{X^T Y} \tag{4}$$

and the error minimized for lasso regression is

$$|\beta_0 + \mathbf{X}\hat{\beta} - \hat{\mathbf{Y}}|^{\mathbf{2}} + \lambda|\hat{\beta}| \tag{5}$$

The added term penalizes the least squares. λ is a varying parameter that determines how strong the regularization is, a value of 0 is the normal least squares. As a result of the penalty, ridge and lasso regression results in a different fit then least squares but at the cost of slightly more error. Lasso works similar to ridge but instead of penalizing the L2 norm, Lasso penalizes based on the L1 norm of the coefficients.

For this report, the values of lambda used will be lambda.min and lambda.1se. Lambda.min is the lambda that results in the least cross validated mean error, and lambda.1se is the largest value of lambda within one standard error of the error of lambda.min. Thus, lambda.1se adds more regularization at the cost of more error. For ridge and lasso the R package glmnet was used. For ridge the parameter α is 0, and for both ridge and lasso we used the binomial option for family.

We do not preform any forward or background AIC/BIC searches or K-NN methods because we were worried that with the large size of the data and the large number of predictors that these methods would suffer both computationally and in model accuracy.

The R package readr was used to read the train and test data into r.

The extreme boosting package for R xgBoost is an extremely powerful tool for classification on

large data sets. The xgBoost package is designed for boosting on large scale tree systems. This combines what we have learned about boosting with the learning scheme of trees to create a quick and efficient classification algorithm. This package is great becasue little to no input information is needed.

PCA, principle component analysis works by using singular value decomposition to determine the principle components in the data set. Then, based on these principle components, perform linear least squares regression with these components, then transform the model back to the original predictors. This method thus works by only studying the predictors that are not heavily correlated with one another. This method is similar to noise reduction in signal processing and file compression. PCA can be used to determine components that explain the data very well. Thus, PCA is used to eliminate predictors that are not strong explainers of the data. For this portion, the function princomp from the package stats is used. From this function, we determine the principle components that explain 95% of the data, and transform the training and test data into these components.

V. ISSUES

Because of the categorical variables, biglm would often return the error "NA/NaN/Inf in foreign function call." While we could not find the exact reason for this issue, we did determine that if all categorical variables were removed biglm would run. As a result, in order to perform linear regression on the full predictor space, we just used R's lm function, while this was much slower than biglm, we were able to obtain an solution.

VI. RESULTS

First we submitted a null model. The null model was composed of all zeros for every test idea. This resulted in a solution score of 0.500. From this we learn that half of the solution is 0's and half is 1's. Another important observation we made from the sample submission file is that the submission is expected to be in decimal form not in binary classification as the training targets were. Further, when we converted the sample submission file to 0s and 1s based on if the response was greater than 0.5, the submission score decreased by 20%. Therefore, the criteria used by Springleaf for whether to send mail is not 0.5. So the submission file is really the probability that a certain person will reply to the mail offer, and Springleaf uses a criteria other than 0.5 to decide if they should send the offer or not. We learned that Springleaf uses an AUC to determine the final score of the submission, and therefore the submission should be a decimal number not a binary number.

Next, we performed linear regression on the full data set using the function lm and predict.lm from r. The resulting model received a submission score of 0.76098, which is 5% from the top score on the Kaggle leader-boards. This showed us that even a simple linear model can perform rather well on large, messy, real data sets.

We also tried using linear regression on the non-categorical and non-time predictors. For this, we used biglm since it is faster for large data set problems such as this. The resulting model received a score of 0.76016, which is comparable to the full model. Thus, removing the categorical variables had little influence on the model prediction for linear regression.

We then tried linear regression with lasso penalization. The resulting model had submission scores ranging from 0.500 to 0.75068. The 0.500 submission resulted from the lasso fully penalizing the coefficients so only the intersect remained, and the 0.75068 was the solution with lambda resulting in the minimum train error. However, this clearly resulted in the submission score being worse than the full model.

We then tried linear regression with ridge penalization. The resulting model had submission scores ranging from 0.500 to 0.75129. The 0.500 submission resulted from the lasso fully penalizing the coefficients so only the intersect remained, and the 0.75129 was the solution with lambda resulting in the minimum train error. However, this clearly resulted in the submission

score being worse than the full model. This model had comparable results to the lasso regression.

We next performed PCA on the data set without the categorical variables or time variables, only the numerical variables, and selected the principle components that explained 95% of the variability. We then ran the linear regression on the reduced variable set. The kaggle score on the principle components was 0.7603. From this, we hypothesized that when dealing with large data sets such as this, removing the variables with only 5% of the remaining variability does not effect the model much. Our guess is that with the number of degrees of freedom in the data set, removing a small number of seemingly useless variables will not change the parameters calculated in the linear model by very much.

The xgBoost package was hailed by many on the Kaggle forums as the best for tackling this particular problem, so naturally it seemed like a good place to start using more complex models. Since this method is new to us, it made sense to try to run with all the default settings first. Using only the default settings, the Kaggle score was 0.7624 which in its own right is pretty good.

Thus far, we have experimented with the use of the missing command which tells the algorithm what missing values look like. This did not improve our score which means that the missing values either have little effect on the results or the algorithm handled them properly. We believe the algorithm properly handled the missing values becasue there are many forums posts talking about dealing with missing quantities. Next some variable preprocessing was done according to some suggestions posted on the Kaggle forums. The ID column was removed along with variables var_0212, var_0227, and var_0228. These were chosen for removal becasue they seemed to mimic the ID column or had no unique values. Again, the xgBoost algorithm performed exactly the same giving a score of 0.7624.

To attempt to improve our score two of the parameters in xgBoost were examined. The first was the learning rate of the algorithm. We expect the algorithm to prefer a small learning rate which will take a very long time to run. Five values of the learning rate were examined. Those values are 1, 0.5, 0.1, 0.01, and 0.001. As the learning rate decreased the speed of the script also decreased so the time until completion increased. In reality though, the Kaggle score did not favor the smallest learning rate, but rather it favored a rate near 0.3 (default). The scores from the five runs are 0.71399 for 1, 0.73880 for 0.5, 0.74804 for 0.1, 0.72206 for 0.01, and 0.66672 for 0.001.

The other parameter examined is the max tree depth. Tree depths of 12, 17, 22, and 27 are examined. As tree depth increases the score is expected to improve, but the algorithm seems to take longer with additional tree depth. The scores obtained are 0.74306 for 12, 0.74602 for 17, 0.74653 for 22, and 0.73981 for 27. This shows that a tree depth of about 22 is the best depth for the algorithm. Additionally it seems the defaults for the xgBoost algorithm seem to perform the best becasue all these scores are worse than when the algorithm is set to the default.

In an attempt to further improve our Kaggle score, several different ensembles were examined. The first was a simple average. The average was tried between groupings of two of the data as well as groupings of three for particularly responsive sets. Groupings tried consisted of biglm and lasso for a score of 0.76829, xgBoost and biglm for a score of 0.77633, xgBoost biglm and lasso for a score of 0.77573, as well as xgBoost biglm and ridge for a score of 0.77584. Other combinations were not attempted becasue they did not seem likely to improve our score. Of all the ensembles attempted the simple average of biglm and xgBoost provided us with the best score of 0.77633. This is likely becasue the submission files occasionally had conflicting predictions and their average was a better choice than either individual value.

Another method ensemble attempted tried to polarize the data by taking the highest or lowest value for a particular data point. This method worked by first taking the simple average of the submission variables. This average is not returned, but rather used to make a decision on

whether or not the highest or lowest score will be taken. A cutoff is compared to the average value, and if the average is higher the highest value is chosen and vice versa. Cutoff values of 0.7, 0.6, 0.5, 0.4, 0.3, 0.2 and 0.1 were examined. This method is designed to make the number of data points near the Kaggle decision boundary small if the cutoff is properly chosen. This method did not end up working as well as the simple average and often did not improve on the solo scores of the methods. The one benefit is though playing with this cutoff point we were able to discern the Kaggle cutoff value which is around 0.2.

Method	Score
Null Model	0.5000
Sample Binary	0.6056
Sample Submission	0.7576
Full Linear	0.7610
Linear No-Categorical	0.7601
Lasso	0.5000 - 0.7507
Ridge	0.5000 - 0.7513
PCA Full Linear	0.7603
xgBoost	0.7624
Simple Average	0.7683 - 0.7763
Other Ensemble	0.5241 - 0.7020

VII. CONCLUSIONS

From our results, we drew several conclusions. First, we notice the importance of preprocessing data when dealing with real world data. Thus far, we have only dealt with well posed clean data for our homeworks, which can be easily modeled and immediately modeled. This problem taught us the importance of extracting extraneous variables out, handling missing and na values. The data initially is unusable because of the missing values. Further, some people have posted that they received scores of around 0.60 for linear regression on the full non-categorical predictors. That is a massive difference from our results. This further shows that while the model can vastly change the submission accuracy, the methods of preprocessing and data handling also plays a crucial role in data science.

Next, we conclude that while simple, the linear regression model is an extremely powerful model even in real life messy data such as this data set. Previously, we had used the linear regression model on data sets that were well posed for a linear model. But in the real world there is no guarantee that the relation is linear, yet our full linear model scored fairly well on the leader boards, thus showing the power of a simple linear model.

One portion of the report that we still struggle to understand is why lasso and ridge regression performed so vastly worse than the full linear model. We had hypothesized that these models would perform better.

Using the xgBoost method seemed to be the best solo method attempted. This method without any parameter tuning achieved a score of 0.7624. Some of the parameters were examined and optimized to produce this score these parameters include the max tree depth and the learning rate of the system.

Additionally we attempted two methods of ensembles. The simpler method which was a simple average seemed to perform better than any of the methods by themselves. The other ensemble method attempted to reduce the number of values near the decision cutoff. This method was a failure it performed worse than any of the methods by themselves.

Lastly, as we mentioned in the midway report, we attempted different methods of handling the NA/NAN/INF values, such as using the mean and median value of the variable to replace these values instead of using -1 as we did. However, we found that regardless of what value we replaced the NA/NAN/INF with, the models were not effected by very much. We concluded that because there is so many data points in the data set the value replacing the NA/NAN/INF value does not effect the bulk data points, so the predictions are remain mainly the same.

[1] Kaggle, "Springleaf marketing data," (2015).