Predicting Clicks Using Maching Learniing

Katie Wallis, Philip Fang, Aman Jiddewar & Alex Su

**­­­**

**Data Understanding & Data Cleaning**

For this project, we were tasked with predicting whether a customer would click on a given ad. Looking at the original data, we had 32 million instances consisting of 24 variables designating various categories. The number of instances is too large for data cleaning, therefore we randomly sampled 100,000 so that they represent all of the categories while not bringing too much burden to run. Some of the 24 variables were simply identifiers for the data. As a result, we removed any attribute that had ended in “\_ip” or “\_id” because these attributes wouldn’t provide our model with information. After removing these variables, we were left with 18 variables that primarily corresponded to anonymized categories. Because of the categorical nature of the variables, the data attributes needed to be transformed into dummy variables by factoring them, which lead to n-1 new binary variables. Because the categories were anonymized, it was difficult to bring in any business domain knowledge to our understanding of the categories. To better understand them, we decided to analyze how each category’s “Click through Rate”, meaning the percentage of time an ad was clicked on. From this we found that overall, the ads had a click through rate of 0.168 for the sample, but some categories’ click through rates varied significantly. We then ranked each category by their perspective frequency and used the average click through rate of 0.168 as a cutoff. Categories with click through rates below the cutoff would be combined into one category called “other” while categories with higher frequencies and representative CTR were kept in the data. Finally, we removed a few of the variables where the “other” category held a large portion of the variable.

**Model Building**

Ultimately, we created three predicative models to solve the challenge: a classification tree, a random forest and lasso regression. To build and optimize models, we knew we would need a validation dataset, which would give us an estimate of performance as we tuned our model. We created this validation set by withholding a small portion of the training dataset each time the model was fit. After each model was fit, we used this validation data to choose between models and to select different tuning parameters such as tree depth for our Classification Tree and alpha in our regression models. Before we could start building and fitting the models, we needed to make sure that the dummy variables (described above) were consistent across the training and validation data. The file titled “Matching Levels” fixes this problem by creating a “factor list” that contains all the variables present in the training data and compares the variables in the validation data to that list. If a variable exists in the validation data that does not exist in the training data, that variable is treated as an NA in the algorithm. This process is also repeated again at the end for the final test data once the best model is chosen.

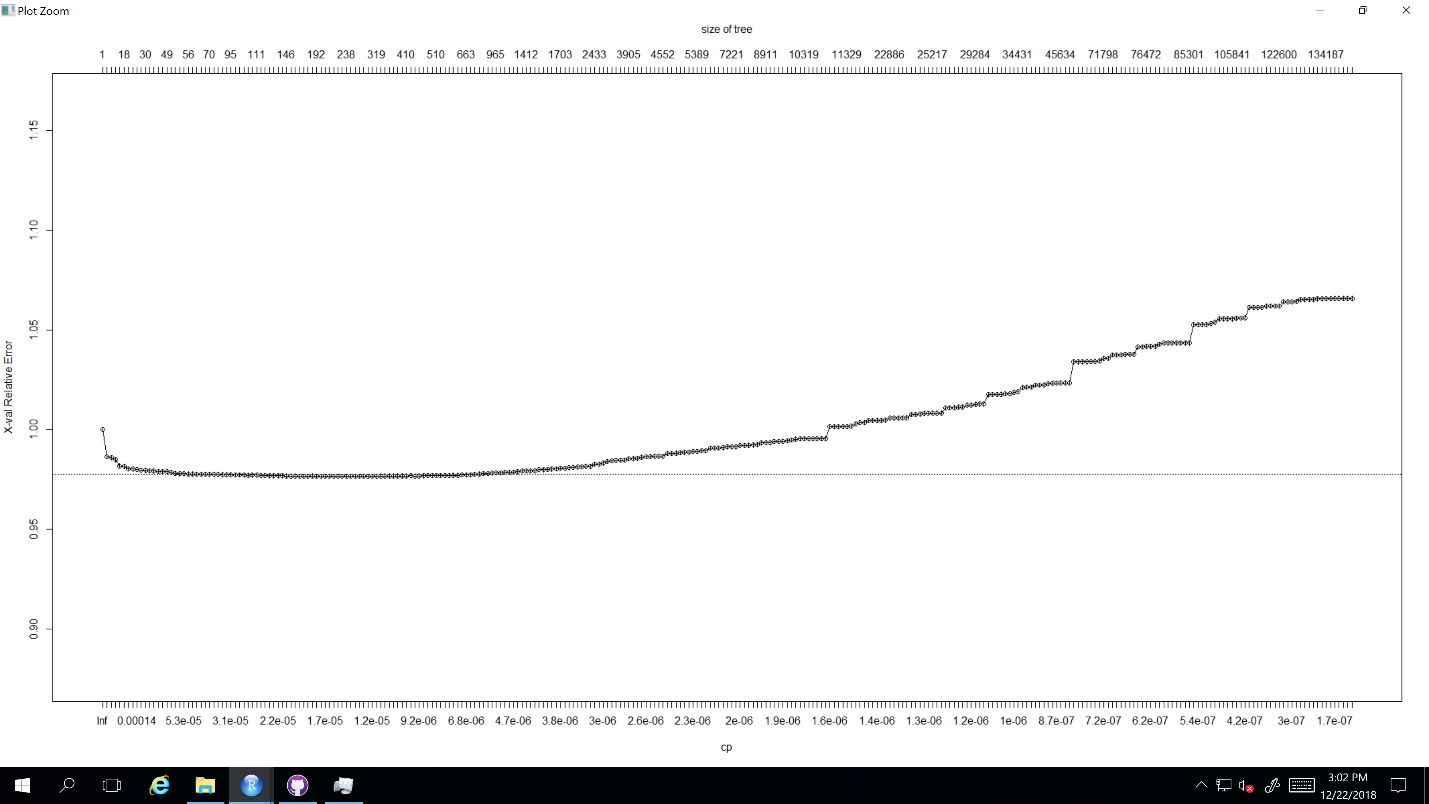
**Lasso Regression**

Lasso Regression is a type of classifier that combines a classic regression function with the L1 penalty which penalizes a function using the absolute value of the length of the coefficients. In so doing, Lasso Regression performs regularization, which combats the common problem of overfitting in models by penalizing more complex models. Because this type of regression uses the L1 penalty as opposed to the L2 penalty which is used in ridge regression, the penalty function actually performs feature selection as some coefficients are set to 0. The degree to which the model is penalized for being complex is determined by the parameter alpha. For this model, we needed to determine what value of alpha would lead to the best model. To do this, we created a grid of possible alpha values in r and comparing the results of each alpha value on the held out validation data. Through this process, we ultimately decided on an alpha value of XXXXXXX which we found to best balance the model in terms of complexity and accuracy.

**Decision Tree**

The next model we tried was a decision tree. The decision tree classifier works by creating a set of “And” rules that split the data into more pure subsets. The goal of the decision tree to take each instance down a path that follows these rules until it gets to a place where all of the instances that follow that path are of the same classification. This classifier is very effective and easy to understand and implement. Just as the L1 penalty was needed in Lasso Regression to prevent a model from becoming overly complex, decision trees have the option of doing pre or post pruning to combat overfitting. For instance, with pre-pruning, the model can stop splitting into smaller subsets if the information gain of that split isn’t enough or if there aren’t enough nodes in the resulting subsets. For post-pruning, the model may be cut back to a certain depth. This pruning is extremely important. Without pruning, a decision tree would grow to be very deep, complex and overfit to such an extent that subsets could be created for a single instance. That level of complexity would be picking up on noise in the data and wouldn’t be indicative of overarching patterns in the data.

We created our model using the rpart function, which is a part of the rpart package. This function allows us to make a decision tree with certain set parameters. You can see in the code, we created a very large decision tree and set parameters with the rpart.control function. Once we had created a large complex tree, we used the prune function (which is also a part of the rpart library) to prune the tree in a way that optimizing model performance on the out of sample validation data. Below in the graph, you can see the relationship between the models performance and the size of the tree. At first when the model is very simple, adding more complexity to the model is beneficial as it allows the model to pick up on patterns in the data. However, you can see at a certain point, higher complexities lead to much worse performance as the model becomes overfit and picks up on noise within the training data. The sweet spot comes at the minimum of this graph where complexity of patterns being discovered is balanced with the problems of overfitting.



**Random Forests**

Two random forest models were built to predict whether someone had clicked on an ad—One with bagging and one without. Random forests are ensemble methods of decision trees. Because they include many trees within them, they tend to be more robust. Random Forests work by having many different decision trees that all vote for their predicted classification. Ultimately, the classification with the most votes from the decision trees becomes the prediction of the random forest. To force the decision trees to differ from one another, random subsets of the attributes are considered at each split rather than having all of the attributes be considered each time. Because different attributes are being considered each time, the trees within the forest are forced to go down different paths through the data. Additionally,

**Model Selection**

|  |  |  |  |
| --- | --- | --- | --- |
| Model | Log Likelihood | AUC | Parameters |
| Decision Tree (using rpart)(Full train data rows) | 0.402 | 0.726 | minsplit = 1, minbucket = 1, cp = 0, usesurrogate = 0,  folds of cross validation = 5 |
| Logistic Regression |  |  | Alpha = |
| Decision Tree (using rpart)(100000 rows) | 0.414 | 0.675 | minsplit = 1, minbucket = 1, cp = 0, usesurrogate = 0,  folds of validation = 5 |
| Random Forests with Bagging(100000 rows) | 0.411 | 0.681 | Mtry = 22  Ntree= 500 |
| Random Forests without Bagging(100000 rows) | 0.412 | 0.684 | Mtry = 5  Ntree = 500 |

We choose to use rpart for final model building, because of limitations in random forest. Random forest gave us error of “cannot create vector of size 12GB”. The results of rpart were not far from random forest as shown in above table. Hence we used rpart and pruned the tree for CP 5.78265e-07, which had least error on validation set for 5 folds cross validation.