Predicting Clicks for Online Advertisements Using Machine Learning Models

Machine Learning Final Project Report  
Team 1 - Adil A Kumar, Daniel Byun, Jiaqi Chen, Ivy Zhou

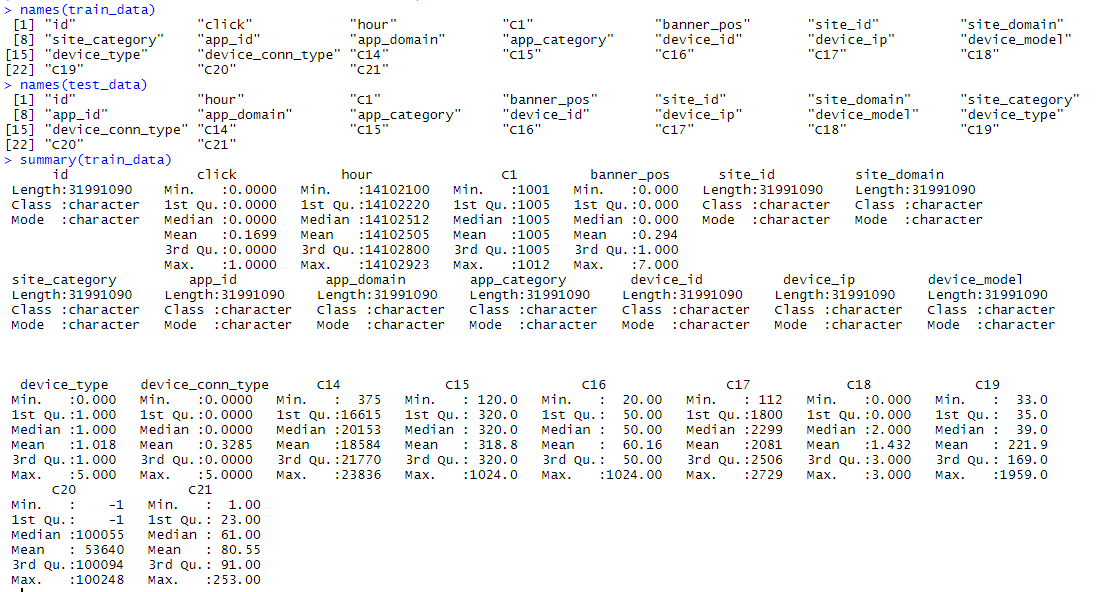
DECEMBER 2019

**Introduction**

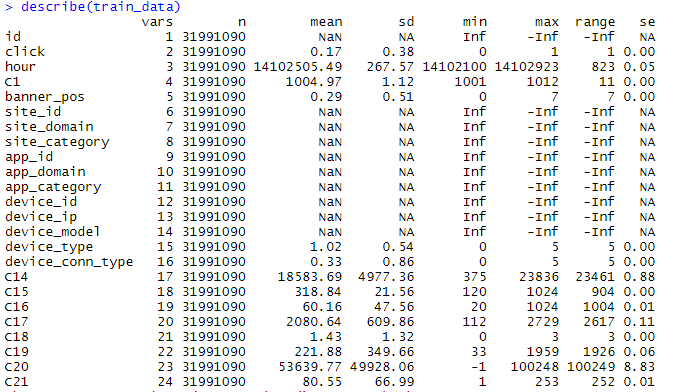
Online advertisement is now widely applied by businesses to deliver promotional messages via the internet. For our project, we trained multiple machine learning models using 30 million online advertisement records from October 21, 2014 to October 29, 2014 in the aim of predicting clicks for those advertisements.

**Exploratory Data Analysis**

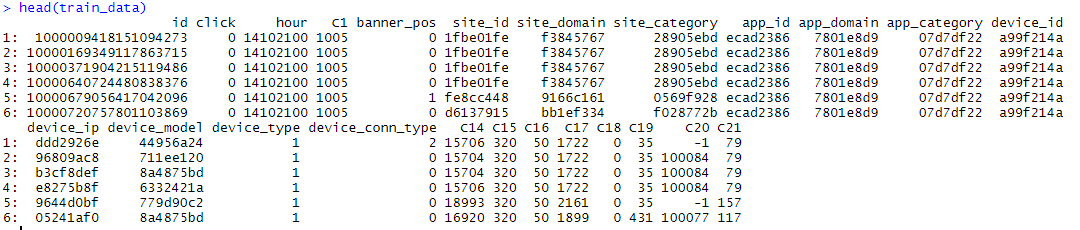
First, in order to get a basic understanding of our data, we ran a summary of the whole training dataset.

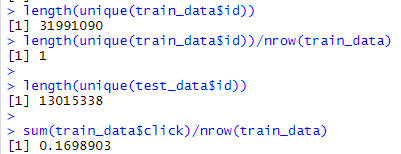


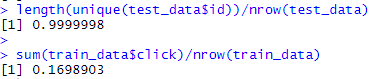
From the given output, we understand that all variables are categorical, except for hour. Clearly, since we have mixed types of data, we need do some data cleaning and transformations later.



According to the output of the describe command, the entire dataset contains 31991090 observations and 24 variables. The head function gives an idea of what our data looks like.

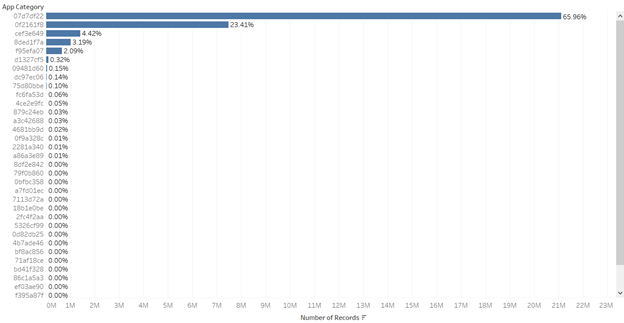


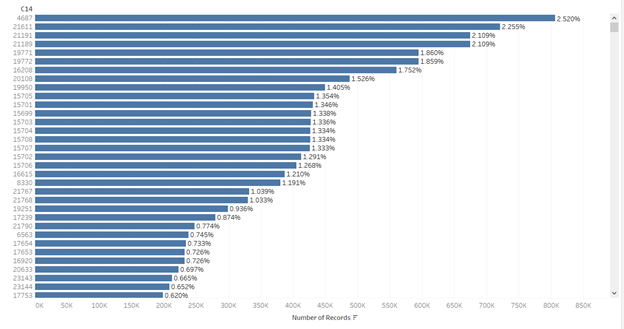




To get an understanding about the id column, we checked the number of unique ids in training data – it was equal to no of rows. However, this was not the case in test data, which is in line with what is given in documentation. Finally, we check the target variable distribution, and note that it is a skewed distribution, indicating only 17% of the data involved users clicking the ad.

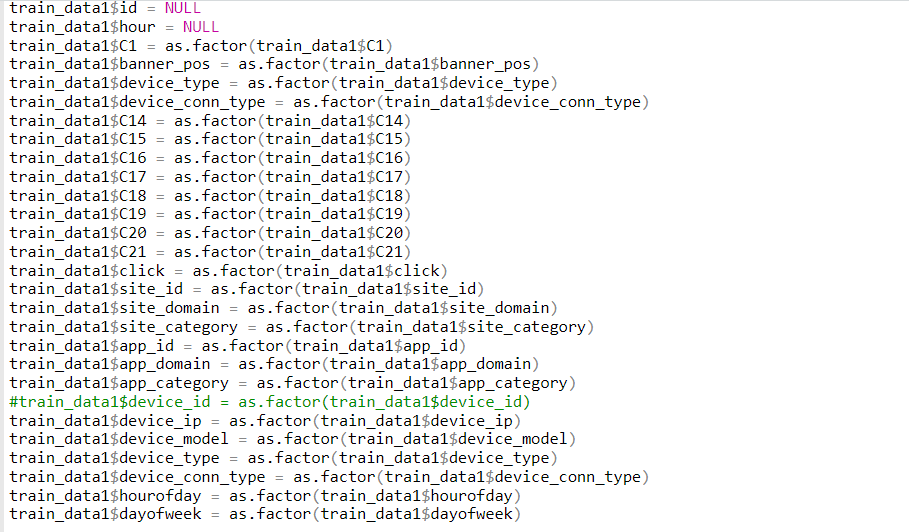
To get a better idea about the level distribution for each categorical variable, we used tableau. We created bar charts to show % of rows in the data that fell under a level for each categorical variable. We have submitted a word extract from tableau containing the same. We could clearly see that some variables had few levels that were representing a huge chunk of the data (pareto principle) while some variables had many levels with similar % of rows. Based on this we used code to keep only the most frequent levels and grouped the less frequent levels into other.



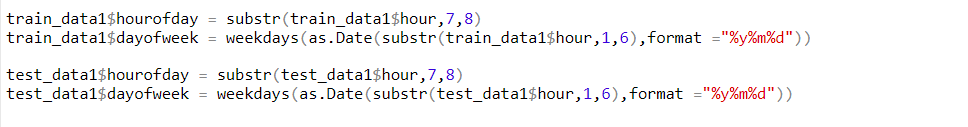


**Data Preparation**

As mentioned earlier, each observation in the dataset has a unique id so we decided to drop this variable since it has no predictive value to the model. We also converted all variables into factor type since many of them were not meaningful numbers to our model.



To better indicate the specific time point of an observation, we also extracted hour of day and day of week from the hour column to make the best use of the time data and dropped the hour column.

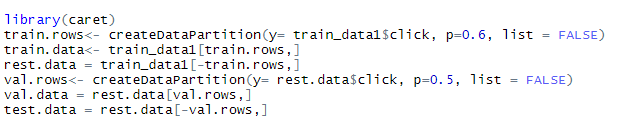
From our Exploratory Data Analysis graph about variable categories, we noticed that each categorical variable has a different distribution and hence we would need to do the grouping based on different cutoff criteria for each variable while using the same overall approach. Based on the % of rows each level in a variable, we set a different cut off to group minority levels as others. After we finish the categorization of levels in training data, we also applied the same approach and threshold on the test data.

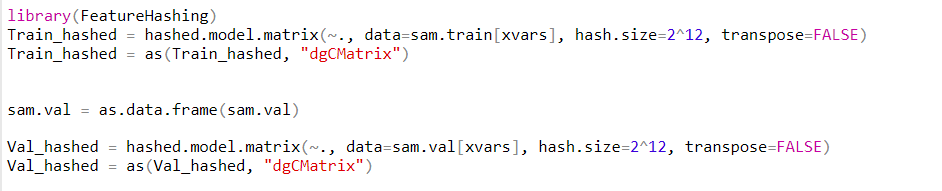






Once we cleaned our predictor variables, the next step was to split the training dataset. We decided to use a 60:20:20 split to split the data into train, test and validation. We split the data to make sure the target variable distribution was consistent in train, validation and test sets.



Initially, we tried running models on the above split training data (20 mil rows) and we encountered computational issues in R with no useful outputs. Then, we decided to use feature hashing on the training data of 20 million rows. Feature hashing is a way of modeling data sets containing large amount of factor and character data. The method will transform features to vector and applies a hash function to the features and uses their has values as indices directly. In this way, we don’t need to convert all level of factors into dummy variables which would create numerous additional columns in this case. 

It takes about 3 hours for us to get the logistic model using hashing approach, so we decided to use hashing on all our datasets since they had a clear advantage in improving efficiency and saving processing time.

With the help of Professor Eastons’ post on “loving bigness of data”, we decided to use a random sample of 1 mil rows to train models and 1 mil rows to validate the models. Both these datasets were independent of each other and had no overlap since we sampled them from our above partitioned train and validation datasets.



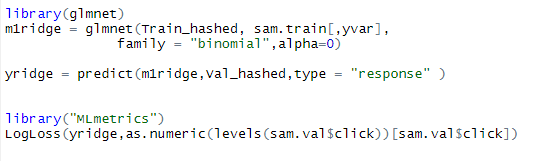
We decided to run all models on those two smaller subsets and identify the best models. Then, we can apply the best model on the training dataset of 20 million rows, identify the best model parameters using the validation dataset of 6 million rows and determine model performance on test dataset of 6 million rows.

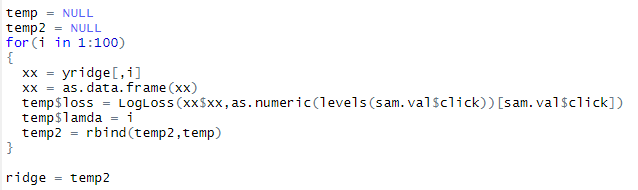
**Baseline Modeling**

First, we started by training models on our random sample of 1 million rows and validating performance on our second random sample of another 1 million rows using the data without additional variable transformation.

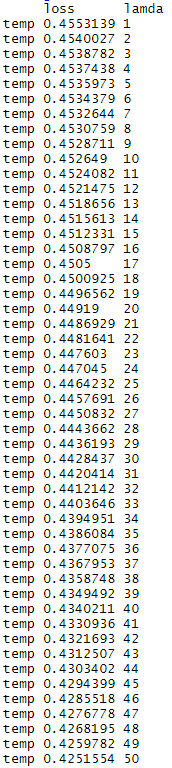
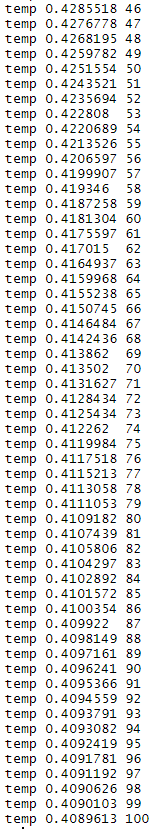
Penalized Logistic Regression - Ridge

Ridge regression uses L2 regularization to weight and penalize residuals when determining the parameters of a regression model. As we mentioned earlier, we applied feature hashing to the dataset and get a matrix of training and validation dataset. Then, we decided to apply this to glmnet() function because it requires a vector input and matrix of predictors. For ridge regression, alpha would be specified as 0 in this case.





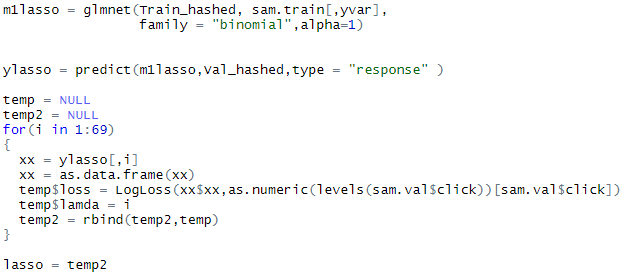
Additionally, ridge regression also has a tuning parameter lambda for the complexity penalty. In our case, we used the apply the hashed training data to glmnet() function and use the validation data to identify the optimized lambda. Based on the results shown below, our best log loss occurs when lambda equals to 100 and the corresponding log loss is 0.4090.

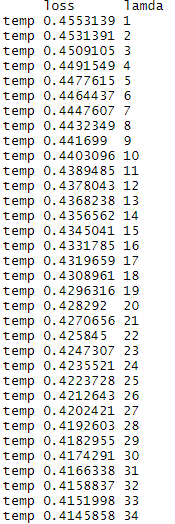
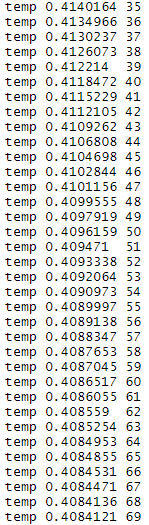
 

Penalized Logistic Regression - Lasso

Lasso Regression performs L1 regularization, which adds a penalty equivalent to the absolute of the magnitude of regression coefficients. We used similar approach as Ridge Regression and set alpha equals to 1 in this case.

We also use validation data to tune parameters and found that our optimized lambda is 69 with a log loss of 0.4084, which is slightly better than the log loss generated from ridge regression.



Extreme Gradient Boosting – Tree Learning

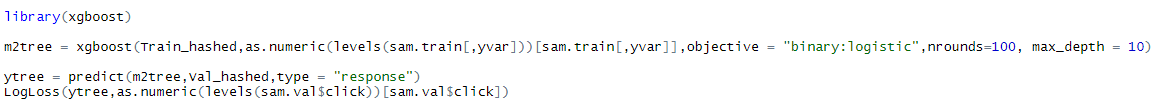
We used the xgboost package in R to implement a tree based learning algorithm based on gradient boosting framework. The main advantage of xgboost is parallel computation which helps it run 10 times faster than classical gbm. It also works with sparse matrix, which suits our purpose.

Parameters tuned with values:

Max depth – No of splits in tree (default up to 100)

Nrounds – no of passes the model makes on the data. Each pass tries to reduce the error in prediction (default up to 1000)

We tested multiple values of max depth and nrounds on validation data and found ideal values that prevented overfitting and minimized log loss. The final values we chose for the parameters was max depth=10 and nrounds=100. We achieved a log loss of 0.4016641. This was the best log loss we achieved across all models.



Extreme Gradient Boosting – Random Forest

We chose to implement random forest with xgboost instead of the traditional randomforest package due to the speed advantage of xgboost. We used the following parameters:

num\_parallel\_tree - No of trees (default up to 1000)

colsample\_bytree - % of columns to sample for each tree (0.1 to 0.9)

subsample - % of rows to sample for each tree (0.1 to 0.9)

We tested multiple values of num\_parallel\_tree, subsample and colsample\_bytree on validation data and found ideal values that prevented overfitting and minimized log loss. The final values we chose for the parameters was num\_parallel\_tree = 1000, subsample = 0.5 and colsample\_bytree =0.5. We achieved a log loss of 0.506724.

Other models considered

With numerous observations and categorical variables, we decided not to use KNN model due to computational issues

We also did not work on neural network model due to computational limitation on the dataset with so many observations.

**Transformations**

After applied the hashed training and validation data into three models as mentioned earlier, we also want to see if variable transformations will give us a better result.

For this dataset, majority observations are categorical variables and the three competition models that we are using do not assume normality, so it is not meaningful to do log or square root transformation on those data. Then, we decided to further analyze data in a qualitative approach to see what kind of transformations we can apply to it.

Transformation I

Firstly, we believe whether a target customer is busy or not can be an important predictor of whether he or she will click on ads. As a result, we decided to use different time periods as a representation of being busy. We classified day of week as weekend (Saturday and Sunday) and weekday (Monday through Friday) as well as hour of day as sleep time (11pm – 8 am), work time (9am – 5pm) and night time (6pm – 10pm) to see if we can identify time points as an effective predictor of clicking on adds or not. After doing this variable transformation, our optimized ridge regression log loss is 0.4092, lasso regression log loss is 0.4086, and xgboost log loss is 0.4019. All three results are worse than validation performance on hashed raw data.

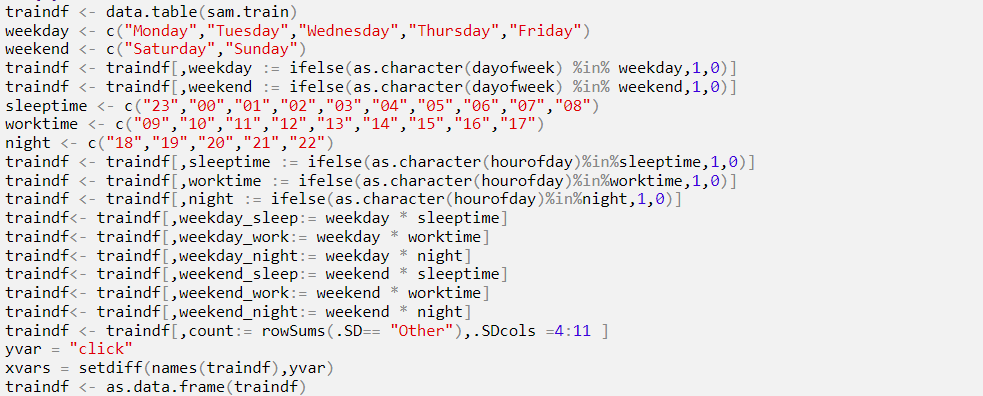
Transformation II

Additionally, we believe busy hours may vary depends on if the day is weekday or weekend. Normally, people may be busy in working hours on weekdays while be busy at night during weekends. As a result, we further created an interaction between being weekday or weekend and three time points of a day by multiplying them. In this case, the log loss of ridge regression is 0.4092, log loss of lasso regression is 0.4087 and that of xgboost is 0.4019.

Transformation III

Another important factor we think is whether the customer is visiting through a “rare source”. Since we categorize those minority sources as “Others” previously, we decided to include a count of number of “Other” sources that a specific observation has. Using this transformation, the log loss of ridge regression is 0.4092, the log loss of lasso regression is 0.4087 and the log loss of xgboost is 0.4018.

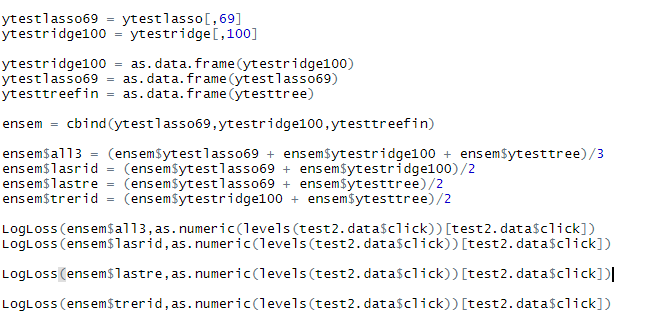
As we can see from the outputs of all three-variable transformations, the log loss does not improve by using variable transformation, so we decided to use hashed raw data for modeling.



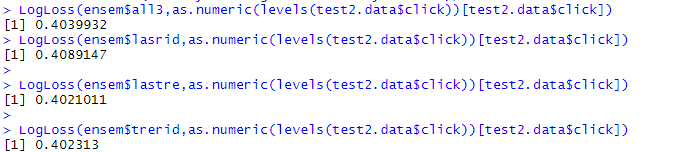
**Modeling on full training data**

Based on our baseline models, we decided to drop use of Random Forest and decided to run lasso, ridge and tree boosting on full training data (20 million rows) without transformations. We then found best parameters using validation (6 million rows) and calculated log loss by predicting on test data (6 million rows).

Using the same approach as mentioned earlier, we applied the feature hashing method to training and validation dataset. The results are attached in appendix b. From the validation results on full dataset, we can see that extreme gradient boosting tree generate the optimized log loss of 0.3989.

We also tried to see if ensemble method would give better log loss values. For the ensemble approach we average the prediction probabilities of model combinations (e.g. lasso and tree, lasso and ridge, tree and ridge) and want to see if results can be improved using this approach. 

However, as the output below indicates, none of the ensembled combinations can generate a log loss better than the results of extreme gradient boosting tree.



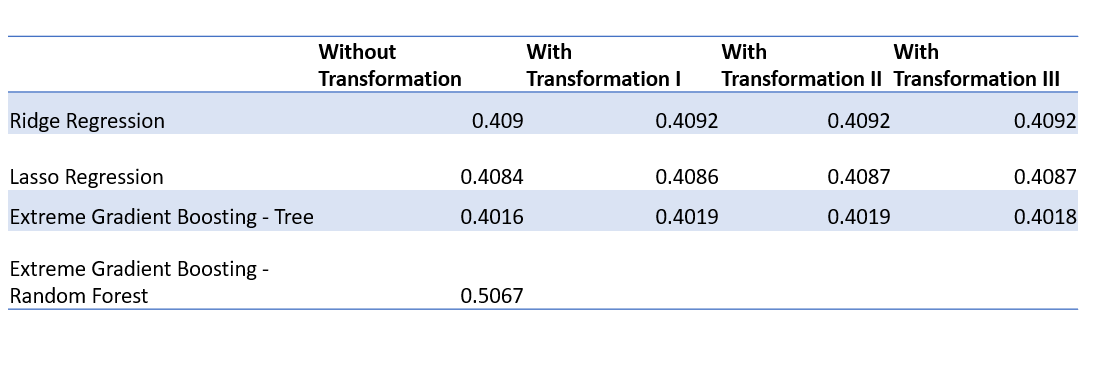
To sum up, we decided to use extreme gradient boosting on the full training set of 30 mil rows and use this model to predict on the final test dataset with unseen y values.

**Code files**

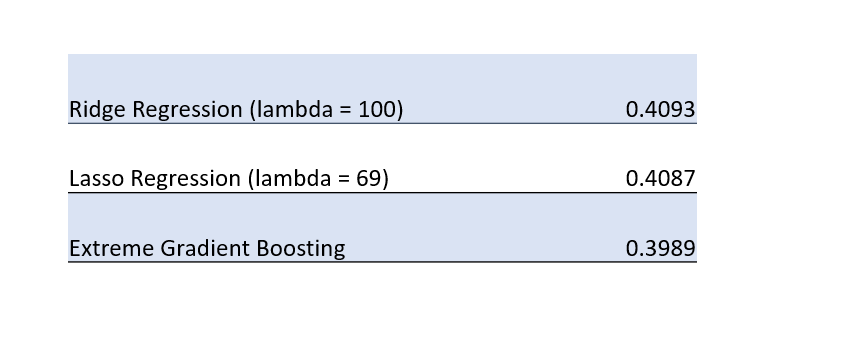
“Final Script” is neatly commented and can be run completely to get final model results we achieved. “variable transformation demos” file shows our transformation demonstrations.

**Appendix**

1. Validation log loss using different models and variable transformations



1. Log loss on three models using full training dataset



**Citation**

https://canvas.emory.edu/courses/63023/discussion\_topics/318438

http://amunategui.github.io/feature-hashing/

https://medium.com/value-stream-design/introducing-one-of-the-best-hacks-in-machine-learning-the-hashing-trick-bf6a9c8af18f

https://drsimonj.svbtle.com/ridge-regression-with-glmnet