Final Project Report – Team 11

Wenbo(Robin) Liu, Daniel Rodriguez, Shanchuan(Frank) LI

Step 1: Sampling and Reading in The Data

Initially we were worried that R would not be able to read in such a large training set, so we tried using different method including suing the hint code or using the Rsql package to randomly query the data. Alternatively, a good method we found useful to draw random rows from a large dataset is to use a Python using a package called "subsample", and it can subtract a one-million sample within seconds. A sample command is: "subsample -n 1000000 ProjectTrainingData.csv > sample.csv". But finally we were pleasantly surprised when *fread* was able to read the entire 130 million rows in a couple minutes. Initially, we thought to convert that data set into a matrix using *model.matrix* to handle the factors and input the data into models. When we attempted this we quickly realized two things: 1.) this was a terrible mistake because of all the factors in the data set, and 2.) there were new factors in the validation and test set we weren't truly accounting for. Once we realized *model.matrix* would not work we did some problem solving and came across a post on Stack Exchange that recommended we create a smaller training set that already included all the factors. Unfortunately, due to time constraints we had to use a different approach to handling new factors. This lead us into step two.

Step 2: Creating a Training Dataset and cross validation.

The next step after reading in the data was extracting a portion of the dataset to speed up our modeling. Because or memory allowed it, we extracted a million rows from the Training data and another million rows from the Test set for the Validation data.

From what we could see, in the 'id' column, we get one million unique records from our 1 million random sample, so we can say that this column will not have predict ability, and we decided to drop it. And the device_ip column has more than half a million unique variable, and we also decided to drop it.

After we removed the first column, we should begin from the third column since the original second column was our target variable ("click"). Additionally, we decided to make the second variable only hour, since we observed that there are only 9 days in the training data, but another 9 days in the test data. And we also think the hour made sense since there should be certain hours that people have more time and are more likely to click the advertisement.

Besides, according to our observation, site-id is highly correlated to site-domain, and app-id is highly correlated to app-domain, we decided to remove site-id and app-id.

We created a sample from range 1:nrow(Training Set) and drew one million indexes. This would become our real training set, since training on 30 million rows of data is unrealistic and unnecessary.

Then we considered our approach to do cross validation. We decided to have one training set, and one validation set simply because we have more than enough data to generalize how well our models perform. Doing k-fold validation and leave-one-out will be too computationally expensive and unnecessary.

Step 3: Coding the Test, Validation, and Training Set

After obtaining our test and validation sets, we set about remediating the problem of handling new factors in our model. To do this we created a list of factors within each column for all three data sets. We then set all the new factors in the validation and test set to having the value "other". As mentioned previously, given more time we would have created an ideal training set that contained all possible factors.

Step 4: Log Loss Function

After dealing with the new factors, we decided to right our log-loss function seeing as that would be our measure of performance. To calculate our log-loss we created a function that changed zeros to small fractions since one cannot compute the log of zero. Next, we implemented the standard equation for log loss using our predicted values (y-hat) and the actual y-values.

Step 5: Naïve Bayes Model

With our loss function and test set ready, we set about creating our first model. To test the waters, we started off with a simply Naïve Bayes model using the *naivebaye*s package in R. At first we modified naivebayes.r provided on Canvas, and we successfully made some predictions. However, it was rather time-consuming, and we had to move to using naivebayes package. Using log-loss as our performance measure we produced a value greater than .6, an underwhelming number.

Step 6: Lasso Regression

Fueled by our ability to create a prediction with Naïve Bayes, we set our eyes upon a loftier goal of using a lasso regression. Through our previous data exploration, we discovered that there was high correlation between specific features of the data set. This ultimately lead us to try and fit a lasso regression model to effectively handle multicollinearity within the data. We used a lambda grid ranging from –2 to 3 with 100 steps. After finding the optimal lambda, our model resulted in a log-loss of .43, a much better loss than our Naïve Bayes model, but still not good enough. This is something we expect because even though Lasso regression does feature selection for us, the dimension of feature space is still quite high and we do not expect a hyperplane could do a very good job at dividing the space. Instead, we would expect something non-linear to perform well.

Step 7: Adaboost

Yes, you read correctly! We used Adaboost, the buzz-word/model of the Kaggle community as of late. We thought this model would be daunting to use, but with the *fastAdaboost* package in R it was a very seamless process. Considering its complexity and our training size we tried out different parameters, and eventually decidedto use 3 classifiers. This model resulted in a log-loss of .38, our highest performance.