4.27.18
Group: P11

Xiaoshi Guo
Jinchi Li
Shawn Thompson

Data Use and Engineering

Initial Training Data Observations & Attribute Adjustments

The 7 default attributes given are: ip, app, device, os, channel, click_time, and attributed_time.

- "attributed_time" is an attribute which we will have value after the predicted class("is_attributed"). We should not have any value for this attribute for prediction purpose. Thus "attributed_time" cannot be used and is deleted from the training dataset.
- The approximate 277,396 unique ip attribute values are assumed to be randomly but statically defined (i.e. an address, or network, will be assigned one numerical value which will always represent that address) and as such will be treated as a factor, not an integer. This is partially justified by the image below which depicts smaller ip values having a higher frequency, on average, that of lower addresses. Essentially, It would be more likely that the first assigned IP values would have higher frequencies, simply because they have been around the longest.

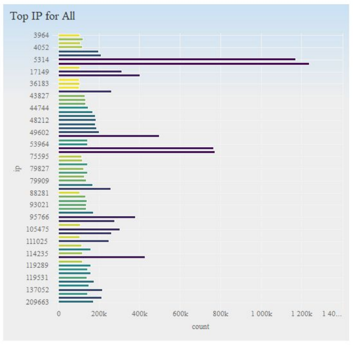
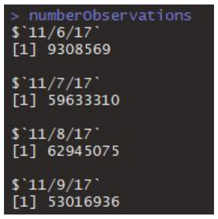


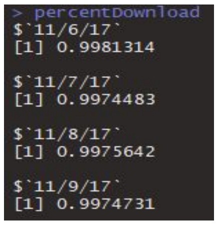
Figure 1: Frequency of ip occurrences, with a minimum of 100k observations, for the entire training data set

- There are 706 app values with "3", "12", and "2" being the most common. These are to be represented in the training data as factors. This is to ensure that no app is treated similarly to another because it is app attribute tag id was randomly close to another's tag id.
- There are 800 different os, but more than 50% of the click happens from os "19" and "13". This attribute will also be treated as a factor.
- There are 202 different channels with "280" and "245" being the most popular. This attribute will also be treated as a factor.
- There are 3475 different device type, but type 1 and type 2 device makes about 98% of the clicks.
- "click_time" is a time attribute which is in character data type. The range of the click time is between "2017-11-06 14:32" to "2017-11-09 16:00". It was converted to day and hour attributes (in integer type) to enable the comparison between different "click-time". This lead to a day attribute which represents only 4 of the days of the week and an hour attribute representing all 24 hours (0-23). Given the training data is sorted by time, it is easy to adjust these points.

- First off, it was determined that all days must be assumed to be the same and, as such, can be removed. If this assumption is not true, a larger dataset of all 7 days with all 24 hours each would be needed. At that point, one working with the data would still need to assume that the data aggregated in the training data is not affected by seasonal changes (this data would work all year long).



```
> numberObservations
$`11/6/17`
[1] 9308569

$`11/7/17`
[1] 59633310

$`11/8/17`
[1] 62945075

$`11/9/17`
[1] 53016936
```



```
> percentDownload
$`11/6/17`
[1] 0.9981314

$`11/7/17`
[1] 0.9974483

$`11/8/17`
[1] 0.9975642

$`11/9/17`
[1] 0.9974731
```

Figure 2: Total observations per day and the ratio of is_attributed per day

- Just as the day of click time, the hour of click time could be used for separating the dataset. Thus we must explore the ratio of downloads clicks over no-download clicks. These values and ratios can be seen below and show that the download ratio differs throughout the day. This indicates that hour should be viewed as a useful attribute moving forward. Another notable feature is that this ratio is very low for the 13th hour in every day. The Figure 3 displays the number of downloads (blue line) vs. number of clicks without downloads(red line) in the train_sample data provided by company. To handle the large scale dataset, we divided it into 4 subset based on their “day”. The first hour of day 1 and the last hour of day 4 is dropped since they are not complete. The ratio changes over hour in a day is plotted in *Figure 3*.

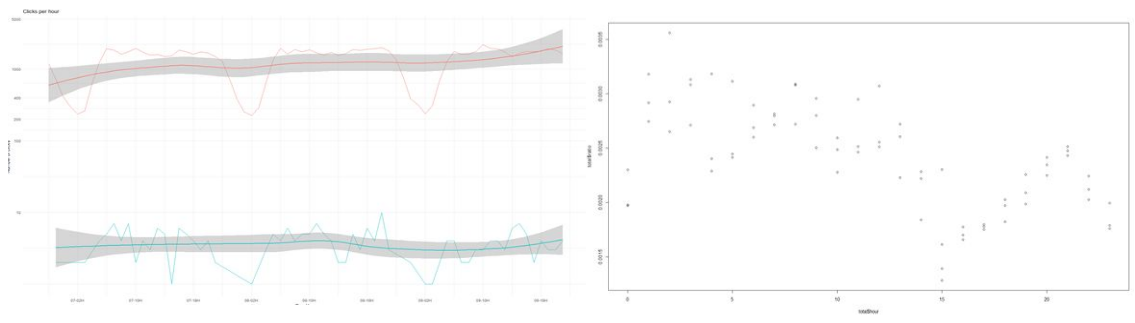


Figure 3: Count (left) and ratio (right) of clicks and downloads for all hours for the sample data set

At this point, the decision whether or not to create additional derived attributes had to be made. The best derived attribute type which we could think of were attributes that were stored as integers and used to count the number of times particular default attributes appeared together. This would include counts of the pairings of ip and all other attributes. However since our assumption was that the ip factor was randomly assigned and does not, in any way, correspond to the network type, we deemed it unfair to make a predictive model based on these counts. Even if we decided it to be fair, it would be extremely difficult to set a point in which this count becomes excessive (more likely to be fraudulent) as business and residential network types should be expected to produce dramatically different non-fraudulent count values. The second best created attribute type was discussed to be a “generalizing attribute” which would shrink the number of values for a specific attribute. An example of this would be changing “os” to have only “19”, “13”, and “other os” values. This type, however was regarded to not truly add anything to the data and, as such, was scrapped.

Generally speaking, an attribute selection method such as Boruta may be implored at this point. It would assist in creating the best possible model by helping identify which attributes are actually useful in determining a download. However, due to the robustness of random forest and gradient boosting models in regards to non-important attributes and the computing resources necessary to do this, we decided not to make use of any attribute selection methods. Technically, attribute selection would likely result in a model with higher accuracy though. Nevertheless the computing cost was not worth the minimal improvement in the model.

Data Size and Imbalance

The first hurdle that one must overcome to create a predictive model based on this particular dataset (TalkingData AdTracking Fraud Detection Challenge’s train.csv) is simply the size of the dataset. This dataset has over 189 million indexes of 7 default attributes with one class value (is_attributed). There are various ways that one could have approached this dataset. These include any of the following individually or in combination: attribute creation (in combination with attribute selection), random sampling of indexes, attribute selection, creating a sparse matrix of the data set to represent the factor variables, downsampling the data, and splitting the data based off of attribute splits. While not all of these were used, all of these options were considered and are discussed below with respect to the two model methods examined (random forest and gradient boosting).

- Attribute creation in combination with attribute selection was made use of twice in this data set. The initial use of this was to split the click_time attribute into two attributes, the day and hour value held within the click_time value. The intention of this creation and further selection was to remove these attributes from the model and was discussed previously.
- Random sampling was made use of for both examined methods. It is worth noting that random sampling was used differently for both examined methods, and the training data was only sampled after other input engineering was made use of. This will be discussed slightly further in splitting.
- Since the majority of the attributes should be considered as factors. It would be helpful to utilize the technique called one-hot-encoding which creates a new column for each of the factor attribute value. This would reduce the overall storage requirement of the training data, and was made use of by converting the data to a sparse matrix for the gradient boosting method.
- Downsampling of the training data would greatly reduce the size of training data as the majority class takes up 99.7529% of the 189 million training data observations. Downsampling would, more importantly, assist in dealing with this drastic imbalance. Essentially, this would allow for the minority class to be predicted correctly at a much higher percentage (a main goal of this project), but models built on downsampled data also drastically increase the variance of that model.

- Splitting the data based off splits in attribute values is a method which was made use of. The attribute which was split on was the (not used in the model) day attribute. All sampling was done from this day only, as it had the most data points out of all four days, and it had the closest click to download ratio to that of the entire data set (99.7529% for all to 99.7564% for Day 3).

Building Predictive Models

From the implementation of this project, the objective of the model is to ensure that a click which would result in a download would not be blacklisted (blocked) while still ensuring that the majority of those whom should be blacklisted, are blocked. For the purposes of building a model, this essentially means that the minority class of the data must be predicted as well as possible while still ensuring that overall accuracy stays as high as possible. A minimal accepted accuracy for the minority class will be discussed, but moving forward, this value can be adjusted upwards or downwards depending on the traffic and number of downloads observed.

Predictive Model Selection

Extreme Gradient Boosting (XGBoost):

Since we do not have access to more expertise domain of this dataset and out-of-box method is frequently used in Kaggle competition, XGB would be a good approach to start with. Due to the superior performance of XGBoost package in R in speeding up the method and handling relatively big dataset, it could help us to handle such a big dataset. On the other hand, the user can deeply customize the learning process through more than 30 parameters in XGB. Last but not the least, other participants who use XGB in Kaggle competition provides their solutions in Kernels, allowing us to compare our results with theirs, and consequently, make us have a better understanding of parameter tuning for XGBoost .

Random Forest (Ranger):

To begin with, random forest models generally give an out-of-bag error estimation which is almost indicative of the true error, and of the random forest generating packages, the Ranger package is the most rapid. Random forest models also tend to perform best on data within their training domain and not very well on data outside their training domain. For something like this project’s purpose, where the large majority of attributes are factors and the model is not truly expected to predict outside its domain, random forest would seemingly work well. As such, significant additions in observations of certain attribute values (such as a new ip) would justify remaking a model or adding additional models to predict a grouping of new attribute values. Essentially, not only would random forest predict poorly outside its trained domain, but also any model created on factor attributes, would not be able to properly make predictions on observations with non-trained on factor levels.

Dataset Used

Extreme Gradient Boosting (XGBoost):

- Random sample 10% of day 3’s data which give us 6*10^6 observation
- Add an attributes related to hour and convert other attributes into factors

- Divide the dataset into training and testing
- Since we want treat those attributes (except “hour”) as factors, we made use of some tricks like one-hot-encoding. Essentially, converting attributes into sparse matrices for use in XGB. DMatrix to satisfy the requirement for XGB package. However, the width of the new data matrix is huge, since there are so many levels for each attributes, and this wide matrix could make overfitting a serious problem for XGB algorithm training. We also saw the most kernel that are available in kaggle treat dataset as numeric value, so we will compare the performance of both numerical approach and factor approach.

Random Forest (Ranger):

- To get the training data, a random sample with replacement of size 1.33 million observations was taken out of day 3’s data subset. This sample was then sampled without replacement to create a test data set of 333,000 observations. The remaining 1 million observations were to be used as training data. The size of this was determined by ensuring a reasonable processing time during model creation while still keeping the training dataset as large as possible.
    - This dataset used the following factor attributes: ip, app, device, os, and channel.
    - This dataset used the following integer attribute: hour.

Tuning

Extreme Gradient Boosting (XGBoost):

There are many parameters in the function of XGB, which generally can be divided into general parameters, and parameters for different types of booster. Since we did not have a deep understanding of all the aspects of the XGBoost, we decided to start from two main groups of those parameters.

- The Gradient Boosting objective method that we started with is the logistic regression for binary classification (“binary:logistic”) with gradient tree booster. The linear function “gblinear” does not seem good for dealing with this factor dataset.
- For preventing overfitting, we first set the “max_depth” of tree to be 11 which is calculated by log(number of columns) when training from sparse matrix dataset (treating attributes as factor), and the “max_depth” for numeric matrix dataset (treating attributes as number) is set at 3. For other parameters related to Regularization, L1 regularization “alpha ” with c(0,4,10) and L2 regularization “lambda” with c(1,5) is used to deal with the overfitting problem. Besides those, the random sample ratio like “col_sample” and “subsample” also helped to prevent overfitting the training data. We set “col_sample” to be 0.8 and “col_sample” to be 0.6 when using sparse matrix dataset. When using numeric matrix dataset, we set “col_sample” to be 1 and “col_sample” to be 0.8 to enable fast converging.

- To deal with the imbalance of class value in XGB, “auc” was picked because we have heavily imbalanced data. Comparing with the root mean square error “rmse” and classification error rate “error”, “auc” would help the algorithm to focus on the performance of minority class. If we decide to only focus on accuracy, then “rmse” and “error” could be more appropriate. Another parameter which helped to deal with the imbalance data is "scale_pos_weight". This parameter will add weights to adjust the imbalance class. I set it to c(10,100,400). It turns out that the high ratio of this parameter will benefit the minority class.

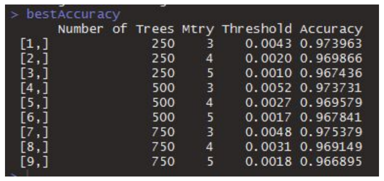
- In terms of speed of training, there are some parameters. For the step size shrinkage "eta", I tried out a list (1:5)*(0.1,0.01,0.001). The result was that these parameters could control the speed of converging but had little effect on the performance of the model trained. For the "early_stopping_rounds", I set it to be 150 which stops the training when the model is converged.

Random Forest (Ranger):

- Adjusted number of trees (num.trees) in the model with the following options: (250,500,750)
    - The number of trees in the model is increased to combat variance. However, setting this parameter to an overly high value would result in higher computing requirements and at a certain point would result in a higher variance (possibly due to overfitting). It's worth noting that the results, which will be discussed further in the following section, did not reach this higher variance point, and if one is willing to spend additional computing time, they may find that a random forest model built on a larger number of trees would perform slightly better.
- Adjusted the number of randomly selected attributes (mtry) to use in the model with the following options: (3,4,5). 6 would be the maximum, but using this would result in extremely correlated trees being used in the model.
    - The number of attributes (randomly selected) which were used to construct the trees were altered. This number would be reduced in an attempt to reduce the correlation between trees (more options of used attributes), but this may also increase variance of the model.
- The used prediction of this model was probabilities. This was done so that the threshold value for determining the value of is_attributed could be adjusted. This would allow for the completed model to be adjusted to predict the minority class much more frequently without adding the additional bias of downsampling the data.
    - The minimum "correct" prediction of the minority class was set at 85%. While this hurt overall training accuracy, this ensured that the most important case (a click resulting in a download) is predicted correctly very often.
- Finally, tuning parameters were optimized and selected based off most optimal training error. In retrospect, a better method of selecting ideal tuning parameters would have been to estimate true error via bootstrap and optimize off that. This, however, could have been skewed if the model's resampled training data did not contain a particular factor level that the independent test set of bootstrap did.

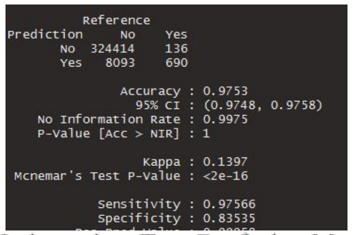
Evaluation of Model Performance

As stated previously, the models will be examined in such a way to ensure both high minority class accuracy while still keeping overall accuracy high. The methods of ensuring this slightly differed between models, so a direct and equal comparison between XGB and RF is not necessarily possible. It is, however, qualitatively make comparisons between the two models via their differing qualitative results. The XGB with numeric dataset perform very close to XGB with factor dataset but faster training time (5 min). However, the numerical approach XGB model find that “app” and “channel” are most important attributes, but factor approach XGB model thinks some particular “ip” is also interested as shown in figure 4. Some tuning parameters like “col_sample” in the XGB are introducing bias into our training process , but it also helps to build a model with less variance. Other parameters like “scale_pos_weight” in XGB and “threshold” in Random Forest are introducing bias into our model to help model focus on the minority class. From this and the below metrics, it was determined that XGB will be used as the predictive model for this data set. While XGB may be more computationally intensive, especially within tuning, its greater estimation of the minority class accuracy and lesser observed variance made it a better candidate of the two.



```
> bestAccuracy
      Number of Trees Mtry Threshold Accuracy
 [1,]             250    3    0.0043 0.973963
 [2,]             250    4    0.0020 0.969866
 [3,]             250    5    0.0010 0.967436
 [4,]             500    3    0.0052 0.973731
 [5,]             500    4    0.0027 0.969579
 [6,]             500    5    0.0017 0.967841
 [7,]             750    3    0.0048 0.975379
 [8,]             750    4    0.0031 0.969149
 [9,]             750    5    0.0018 0.966895
```

RF Tuning Parameter Optimization

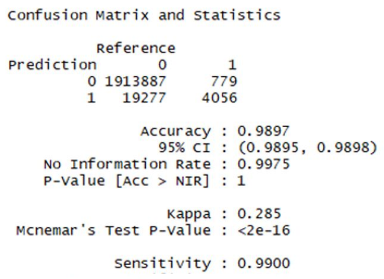


```
          Reference
Prediction     No    Yes
       No  324414    136
       Yes   8093    690

               Accuracy : 0.9753
                 95% CI : (0.9748, 0.9758)
    No Information Rate : 0.9975
    P-Value [Acc > NIR] : 1

                  Kappa : 0.1397
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.97566
            Specificity : 0.83535
```

RF Independent Test Confusion Matrix

Random Forest (Ranger):

- Final Tuning Parameters:
  - num.trees: 750 & mtry: 3
  - Threshold: 0.0048
- Training Accuracy: .975379
  - Minority class Accuracy: ~.85
- Estimation of True Accuracy (independent test set): .9753
  - Minority class Accuracy: .835
- Time to build: ~15 minutes

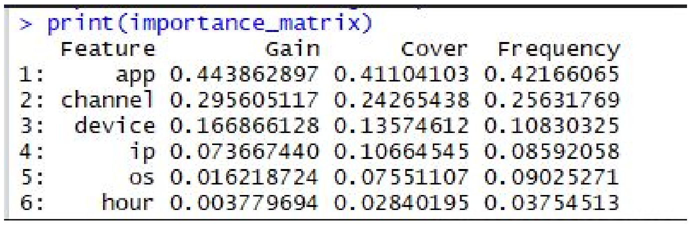


```
Confusion Matrix and Statistics

          Reference
Prediction       0       1
         0 1913887     779
         1   19277    4056

               Accuracy : 0.9897
                 95% CI : (0.9895, 0.9898)
    No Information Rate : 0.9975
    P-Value [Acc > NIR] : 1

                  Kappa : 0.285
 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 0.9900
```

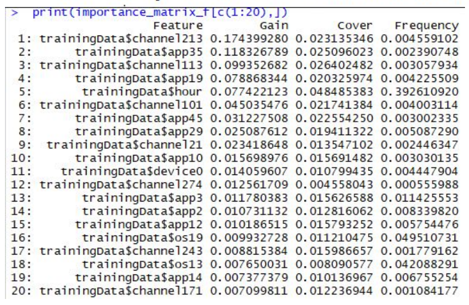
RGB Independent Test Confusion Matrix

Extreme Gradient Boosting (XGBoost):

- Training Accuracy: 0.9893
  - AUC:0.986
  - Minority class Accuracy:~.857
- Estimation of True Accuracy: 0.9897
  - AUC:0.966
  - Minority class Accuracy:~.839
- Time to build:~20 minutes



```
> print(importance_matrix)
   Feature        Gain      Cover  Frequency
1:     app 0.443862897 0.41104103 0.42166065
2: channel 0.295605117 0.24265438 0.25631769
3:  device 0.166866128 0.13574612 0.10830325
4:      ip 0.073667440 0.10664545 0.08592058
5:      os 0.016218724 0.07551107 0.09025271
6:    hour 0.003779694 0.02840195 0.03754513
```



```
>  print(importance_matrix_f[c(1:20),])
                      Feature        Gain       Cover   Frequency
 1: trainingData$channel213 0.174399280 0.023135346 0.004559102
 2:      trainingData$app35 0.118326789 0.025096023 0.002390748
 3: trainingData$channel113 0.099352682 0.026402482 0.003057934
 4:      trainingData$app19 0.078868344 0.020325974 0.004225509
 5:       trainingData$hour 0.077422123 0.048485383 0.392610920
 6: trainingData$channel101 0.045035476 0.021741384 0.004003114
 7:      trainingData$app45 0.031227508 0.022554250 0.003002335
 8:      trainingData$app29 0.025087612 0.019411322 0.005087290
 9:  trainingData$channel21 0.023418648 0.013547102 0.002446347
10:      trainingData$app10 0.015698976 0.015691482 0.003030135
11:    trainingData$device0 0.014059607 0.010799435 0.004447904
12: trainingData$channel274 0.012561709 0.004558043 0.000555988
13:       trainingData$app3 0.011780383 0.015626588 0.011425553
14:       trainingData$app2 0.010731132 0.012816062 0.008339820
15:      trainingData$app12 0.010186515 0.015793252 0.005754476
16:       trainingData$os19 0.009932728 0.011210475 0.049510731
17: trainingData$channel243 0.008815384 0.015986657 0.001779162
18:       trainingData$os13 0.007650031 0.008090577 0.042088291
19:      trainingData$app14 0.007377379 0.010136967 0.006755254
20: trainingData$channel171 0.007099811 0.012236944 0.001084177
```

Figure 4: Attribute Importance within XGB