**CKME136- Capstone Project**

Literature Review and Data Descriptions

Sheena Chauhan

Talking Data Ad tracking fraud detection Challenge

<https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data>

**Introduction**

Click fraud is the act of generating fraudulent clicks on pay-per-click advertisements which results in artificially inflating web traffic and potential loss of revenue to the advertiser.

Talking data provided a dataset of around 200 million clicks for a Kaggle challenge to determine the users that download a mobile application after clicking on a relevant advertisement. The aim of TalkingData mobile big data platform to reduce this fraud by blacklisting the devises and known IP addresses involved in fraud click. The sample dataset provided consist of 100,000 randomly selected rows of training data

Techniques for classification analysis will be employed on the R platform (RStudio) to tackle this challenge.

Below are few questions that are posted to assist with this challenge:

1)     Is there specific type of mobile applications which are downloaded more often by the users?

2)     Is there specific day and time when mobile users download those specific applications?

3)     Is there any relation between the mobile devices used and fraudulent clicks done?

4) Which clicks were not fraudulent clicks

**Literature Review**

Below is the review of research articles based on different areas of click fraud, important features for consideration, and classification algorithms is summarized below to give a comprehensive approach to the research question for this capstone project.

**Click Fraud – Todays Mobile Ad Economy**

<https://techbeacon.com/app-dev-testing/click-fraud-plagues-todays-mobile-ad-economy>

Costed ad companies over 11 billion dollars back in 2014 and IAB estimates up-to 50 percent publisher traffic is bot activity. A big portion of ad’s published or even viewed are from bots and not human interaction, this takes away the purpose of the advertisers, creating havoc in the ad industry.

**Spotting and Preventing Click Fraud**

<https://liftoff.io/blog/spotting-and-preventing-mobile-ad-fraud/>

3 types of ad fraud – Click Spam, Click Injection, Fake Installs.

Click Spam; Execute clicks on behave of user, creating numerous fake clicks without their knowledge. This is in hopes a user may “accidentally” click an ad that pops up and installs or visits, this is what creates the payout. The clicks can happen at anytime a website or app is running and occurs in the background. Click spamming is spotted by noticing a low conversion, or more clicks than purchases.

Click Injection; package added broadcast is when a fraudster clicks right before the app is open. This allows them to “steal” the attribution of the click. Another is the content provider exploit which notifies the fraudster moments after the install button is clicked. Spotted when above average post click metrics are recognized.

Finally, we have fake installs; fraudster emulate fabricate users to fake installs generating revenues from the apps. Fraudsters rely on server virtualization and softwares to fake install, fake engage, and repeat. Spotted by notice an uptick in installs, followed by large drop offs.

**Custom Ad Fraud Detection System**

<https://theappsolutions.com/blog/development/case-study-custom-ad-fraud-detection-system/>

1 out 5 ad visited websites are by bots. 20% of pay per click conversions considered fraudulent back in 2017. In 2018, Facebook disabled approximately 1.3 billion fake accounts. Prevention methods: Digital footprint (predefined patterns to detect), Anomaly based (uses statistical and historical data/analysis) finally credential (a web crawler to determine)

**Case Study: CPG cut ad fraud**

<https://insider.integralads.com/case-study-cpg-programmatic-fraud/>

CPG marketers were pioneers of digital ads, in hopes of increasing price efficiencies and consumer demographic. They were also early adopters of programmatic buying which helped with pricing efficiencies through automated buying and selling. In Programmatic, 15% of display and 75% of video impressions are delivered via bots. Roughly daily cost of fraudulent activities for these customers are upto $1.28 million dollars. CPG industry has a spend of $4.2 billion a year in digital ad, also responsible for 47% of mobile ad buys.

**Ad Fraud and Mobile Economy**

<https://www.protected.media/special-report-sophisticated-ad-fraud-permeates-the-mobile-app-ecosystem/>

In 2018, mobile in –app advertising fraud surged over 800% relative to the prior year. In-app advertising is worth billions of dollars, which is a very attractive industry for fraudsters. In 2015, app called snaptube was discovered to have had the behaviours of; install fraud (downloading additional ad app payloads and executing on devices), invisible ad views, app making, and affiliate marketing. In a study “ **keepmobiCPA** (where CPA refers to “cost per acquisition,” or paying for people who install apps) in the traffic, and captured JavaScript code, reveal that Snaptube uses JavaScript to perform fake in-app clicks that simulate user interaction.” Over the years, there have been 10’s of millions of downloads on many apps (some as popular as the keyboard app Klika for example) which when studied, have scary amount of background fraudulent ad activity.

Outside of the in-app advertising fraud, web surfing pop up ads are another issue. One where you get popups while web surfing, and at times those popups being extremely aggressive starting with the attractive phrase of “congratulations, click here to claim your prize” This activates the background fraudulent ad generations. These ads at times are not able to even be closed, unless done so forcefully.

**Dataset**

I used the dataset from a Kaggle competition “TalkingData AdTracking Fraud Detection Challenge” found at this link: “*https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection”.*

Training and testing datasets are provided by TalkingData for this competition. In total, there are over 200 million clicks that were captured over a 4 days span. The training set includes 184903890 data points with 8 attributes and the testing set includes 18790469 data points with 7 attributes.

Attributes of these datasets includes:  
1) ip

2) app

3) device

4) os

5) channel

6) click\_time

7) attributed\_time

8) is\_attributed (target attribute)

**Description of Attributes:**

**ip**: ip address of click.

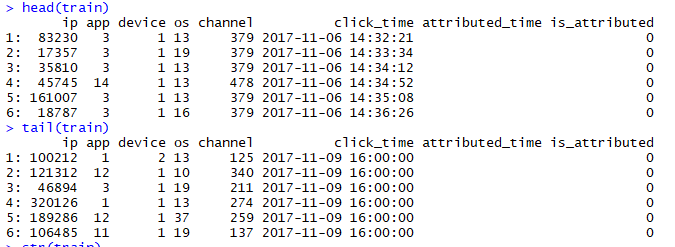
* **app**: app id for marketing.
* **device**: device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
* **os**: os version id of user mobile phone
* **channel**: channel id of mobile ad publisher
* **click\_time**: timestamp of click (UTC)
* **attributed\_time**: if user download the app for after clicking an ad, this is the time of the app download
* **is\_attributed**: the target that is to be predicted, indicating the app was downloaded

Description of the attributes can also be found at below link:

[*https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data*](https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data)

**Summary:**

I used “data.load” library to use “Fread” function to load the dataset to R.



> str(train)

Classes ‘data.table’ and 'data.frame': 18790469 obs. of 7 variables:

$ click\_id : int 0 1 2 3 4 5 6 7 9 8 ...

$ ip : int 5744 119901 72287 78477 123080 110769 12540 88637 14932 123701 ...

$ app : int 9 9 21 15 12 18 3 27 18 12 ...

$ device : int 1 1 1 1 1 1 1 1 1 1 ...

$ os : int 3 3 19 13 13 13 1 19 10 53 ...

$ channel : int 107 466 128 111 328 107 137 153 107 424 ...

$ click\_time: chr "2017-11-10 04:00:00" "2017-11-10 04:00:00" "2017-11-10 04:00:00" "2017-11-10 04:00:00" ...

- attr(\*, ".internal.selfref")=<externalptr>

**Approach:**

\*\* Data Cleaning

\*\* Exploratory

\*\* Feature

\*\* Classification

\*\* Evaluation of models

**Step 1: Data cleaning**

* Checked the dataset for any missing values and found the potential outliers if any. It will have to be examined if those values can be imputed or removed without affecting the quality of the datasets.
* Looking at the click\_time attribute and separated that we can have it in a date and time format. Specifically, with the date component, since the data was collected over 4 days, likely the year and month can be removed from it.

**Step 2: Exploratory analysis**

* Explore patterns within the data and answer questions set in the abstract to get a better appreciation of the data
* At this stage, correlations between different attributes will be done.
* Checking if the dataset is balanced and any necessary adjustment needs to be made to order to balance the dataset
* Graphs to get more visual representation

**Step 3: Feature selection**

* Examining which attributes should be included for the classification algorithms for optimizing the performance of the models. This can be looked at by techniques such as Feature Importance by Gain or Principle Component Analysis as indicated.

**Step 4: Classification algorithms**

* Random Forest as classification algorithms is selected and others as necessary will be done to create models. The target variable “is\_attributed” is used to determine results in the models.
* Ensure that validation sets are considered

**Step 5: Evaluation of models**

* Confusion matrix and other evaluative measures are done to test the performance of the models

Attributes of these datasets includes:  
1) ip

2) app

3) device

4) os

5) channel

6) click\_time

7) attributed\_time

8) is\_attributed (target attribute)

**Description of Attributes:**

**ip**: ip address of click.

* **app**: app id for marketing.
* **device**: device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.)
* **os**: os version id of user mobile phone
* **channel**: channel id of mobile ad publisher
* **click\_time**: timestamp of click (UTC)
* **attributed\_time**: if user download the app for after clicking an ad, this is the time of the app download
* **is\_attributed**: the target that is to be predicted, indicating the app was downloaded

**Initial Coding as below:**

Due to big data sets, I have used below libraries. I installed all these packages first.

library(data.table)

library(plyr)

* corrplot

library(corrplot)

* data visualization

library(ggplot2)

* Cross validation and feature selection

library(caret)

library(MASS)

library(leaps)

* Class imbalance

install.packages("ROSE")

library(ROSE)

* classifier models

library(caret)

install.packages("randomForest")

library(randomForest)

* loading up datasets

test <- fread("Ryerson/test.csv", showProgress = T)

train <- fread("Ryerson/train.csv ", showProgress = T, nrows=18790469)

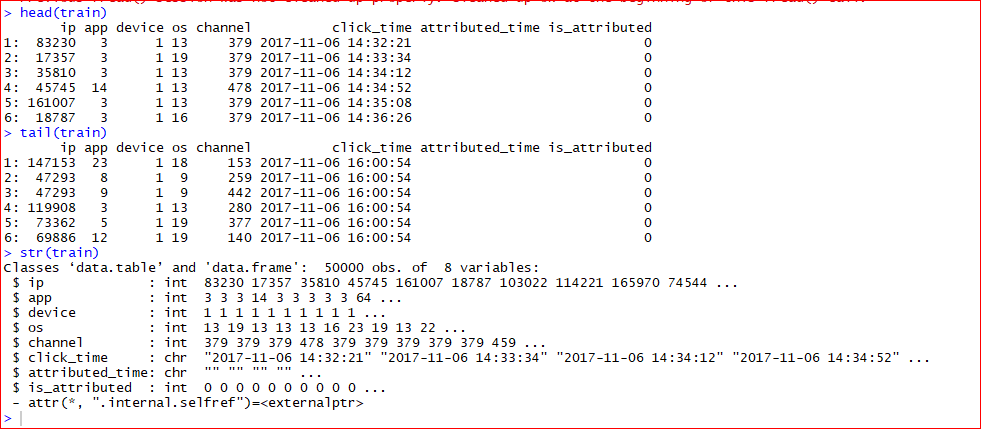
Because Train dataset was huge, I have only loaded 18790469 rows in this dataset to match with test dataset

* quick look at the data

head(train)

tail(train)

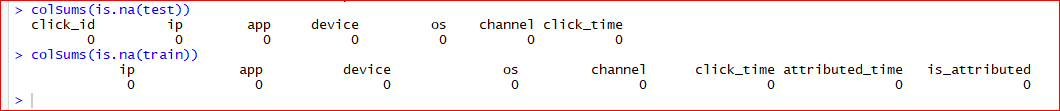
str(train)



* checking for missing values broken down by variables

colSums(is.na(test))

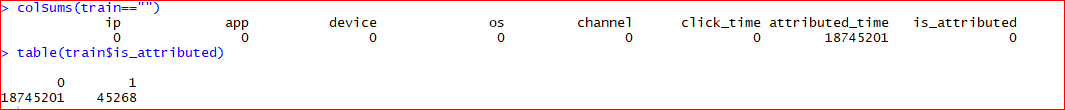
colSums(is.na(train))



Note attribute\_time with blank entries shows they did not download the app (target variable). Proven below where the number matches. Looked at the blank values:

colSums(train=="")

table(train$is\_attributed)



Since the dataset is very huge, I have reduced the dataset to 25000 observations for faster computations and to prevent any memory issues. I will add this to the limitation section at the end of the report for future consideration. A seed was also set to make sure randomization can be replicated again in the future for this reduction.

* Looking at the dataset of target variable, it is skewed (0.24% shows target attribute)

table(train$is\_attributed)

* Control randomization

set.seed(575)

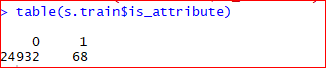
* For easier computation and due to computer limitation, I’ve sampled the dataset. usually it’s a 70/30 split, however, original percentage differences between test and train is 90/10 split

s.train <- train[sample(nrow(train), 100000), ]

s.test <- test[sample(nrow(test), 10000), ]

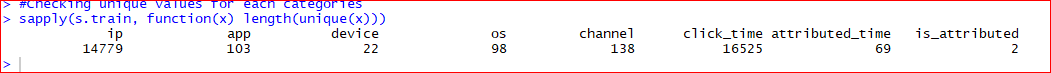
#Ensuring reduced set has similar distribution to original dataset for target variable

table(s.train$is\_attribute)



#Checking unique values for each categories

sapply(s.train, function(x) length(unique(x)))



* splitting click\_time into different columns for better analysis by removing click\_time and year and month since they are the same for all and added seconds:

TrainData.set$click\_time<-as.POSIXct(TrainData.set$click\_time, format = "%Y-%m-%d %H:%M")

TrainData.set$year=year(TrainData.set$click\_time)

TrainData.set$month=month(TrainData.set$click\_time)

TrainData.set$days=weekdays(TrainData.set$click\_time)

TrainData.set$hour=hour(TrainData.set$click\_time)

table(TrainData.set$year)

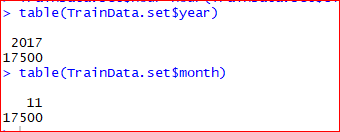
table(TrainData.set$month)

TrainData.set$click\_time=NULL

TrainData.set$year=NULL

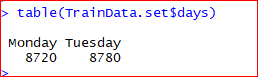
TrainData.set$month=NULL

A closer look at the “Click\_time” attribute was accounted for and it was determined that this attribute can be further split into a date time format. Since the data was only collected over a 4 days span, the assumption was made that the “year” and the “Month” can be removed. For example all 25000 observation of the reduced dataset had the same value for month and year



For further processing, the newly transformed “days” attribute were changed to an integer data type so that each day corresponded to a number (ie. Monday -> 1, Tuesday -> 2, Wednesday -> 3, and Thursday -> 4 etc ). Only 2 days were considered as the data collected by TalkingData only spanned over this amount. This transformation is to help with further computations for this capstone project.

Before changing to Int value



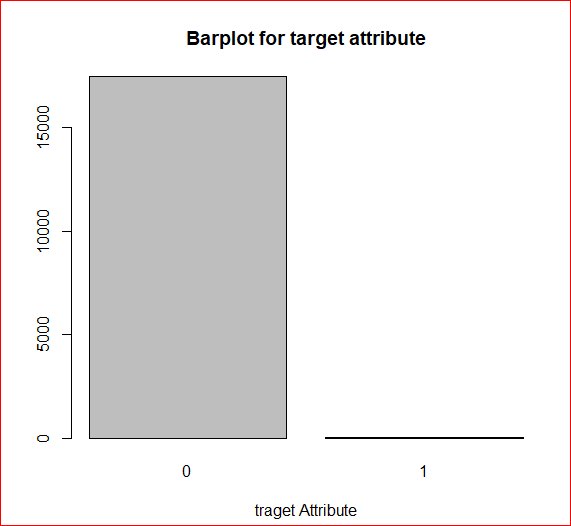
After Changing to Int Value



In addition, for the sake of the challenge which is to determine users who will download an application after clicking on an ad, “attributed\_time” was removed as it provided redundant information. Meaning that “attributed\_time” only existed if there is a target variable in “is\_attributed”. This was seen earlier in this report when looking at blank values where the there are blank values when the target variable, “is\_attributed” value was at 0. More information regarding this “attributed\_time” for future consideration can be found in the limitation sections at the end of this report.

Step 2: Exploratory analysis

Exploring and visualization would provide more information into the data and to reveal important patterns. A barplot was done to look at the distribution of the target attribute, “is\_attributed” in the reduced dataset. It is shown that there is a major class imbalance for this attribute as seen in the barplot below. The exact distribution was shown earlier where there are only 60 observations which are the target variable that we are looking for and the rest of the 24940 observations not being the target variable we are looking for. Consideration for balancing the data for modelling was made at this time due to this major class imbalance.



Next, the reduced dataset was split into two different datasets for the purpose of visualization where one of the dataset contains, “is\_attributed” being equal to 1 and another where “is\_attributed” is equal to 0. Barplots and top frequencies were done with all the attributes to look at how the data differs from between the target and the non-target variable. The blue graphs would represent data where “is\_attributed” is equal to 0 and the red graphs would be for data where “is\_attributed” is equal to 1, which is our target variable that we are looking for.

For the “ip” attribute, differences were found between the blue and the red graphs which shows that “ip” should be an important variable for consideration in our analysis. For the “app” attribute, only the value 9 overlaps between the two graphs indicating that “app” could also be an important attribute as well.

#Split by target attribute

split0 <- TrainData.set[which(TrainData.set$is\_attributed == "0"),]

split1 <- TrainData.set[which(TrainData.set$is\_attributed == "1"),]

#Top 10 frequency for ips (not target/target)

count.ip0 <- sort(table(split0$ip), decreasing = TRUE)[1:10]

count.ip1 <- sort(table(split1$ip), decreasing = TRUE)[1:10]

#Top 10 frequency for apps (not target/target)

count.app0 <- sort(table(split0$app), decreasing = TRUE)[1:10]

count.app1 <- sort(table(split1$app), decreasing = TRUE)[1:10]

#Top 10 frequency for devices (not target/target)

count.device0 <- sort(table(split0$device), decreasing = TRUE)[1:10]

count.device1 <- sort(table(split1$device), decreasing = TRUE)[1:10]

#Top 10 frequency for os's (not target/target)

count.os0 <- sort(table(split0$os), decreasing = TRUE)[1:10]

count.os1 <- sort(table(split1$os), decreasing = TRUE)[1:10]

#Top 10 frequency for channels (not target/target)

count.channel0 <- sort(table(split0$channel), decreasing = TRUE)[1:10]

count.channel1 <- sort(table(split1$channel), decreasing = TRUE)[1:10]

#Top frequency for days (not target/target)

count.day0 <- sort(table(split0$days), decreasing = TRUE)[1:4]

count.day1 <- sort(table(split1$days), decreasing = TRUE)[1:4]

#Top frequency for hours (not target/target)

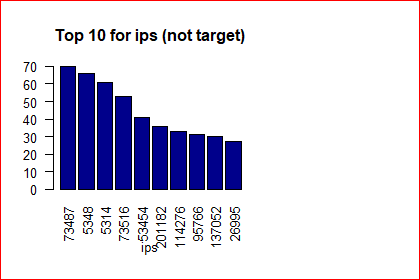
count.hour0 <- sort(table(split0$hour), decreasing = TRUE)[1:24]

count.hour1 <- sort(table(split1$hour), decreasing = TRUE)[1:24]

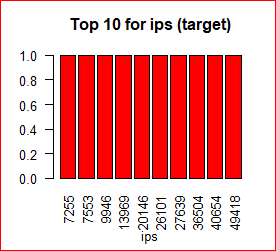
#Barplots

par(mfrow=c(2,2))

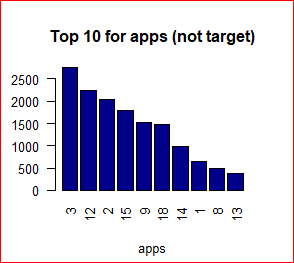
barplot(count.ip0, main="Top 10 for ips (not target)", xlab="ips", col="darkblue", las=2)



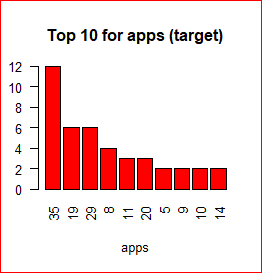
barplot(count.ip1, main="Top 10 for ips (target)", xlab="ips", col="red", las=2)



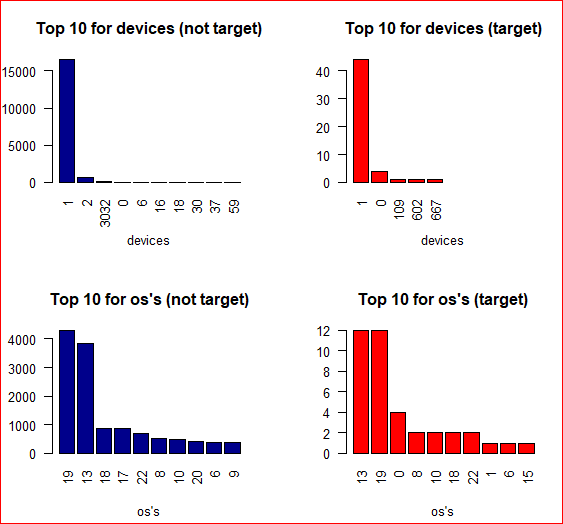
barplot(count.app0, main="Top 10 for apps (not target)", xlab="apps", col="darkblue", las=2)



barplot(count.app1, main="Top 10 for apps (target)", xlab="apps", col="red", las=2)



For the “device” attribute, the majority of devices fall under the category valued 1 between the blue and the red graphs indicating that it may not be an important attribute. For “os” attribute, there were some values (18, 22, 6, and 10) that overlaps between the two graphs suggesting that this attribute may or may not be relevant.



For “channel” attribute, only the value 145 was consistent between the two graphs indicating that this could be another important attribute for our analysis. And for the “days” attribute, the distribution between the blue and the red graphs appears similar suggesting that this may not be as important.

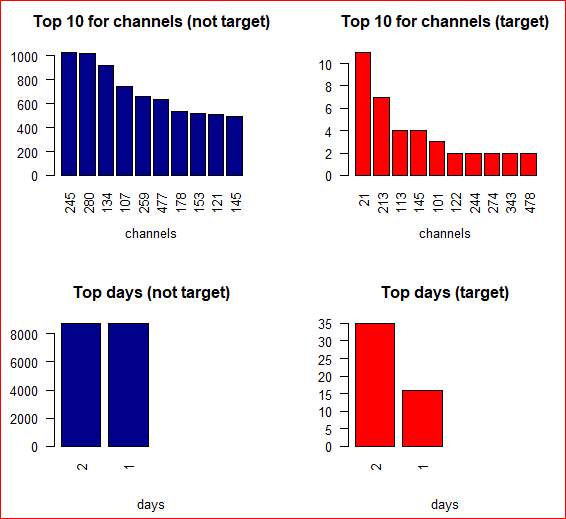
par(mfrow=c(2,2))

barplot(count.channel0, main="Top 10 for channels (not target)", xlab="channels", col="darkblue", las=2)

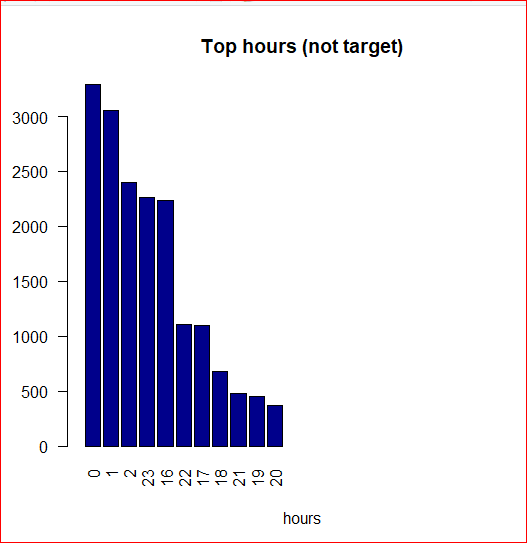
barplot(count.channel1, main="Top 10 for channels (target)", xlab="channels", col="red", las=2)

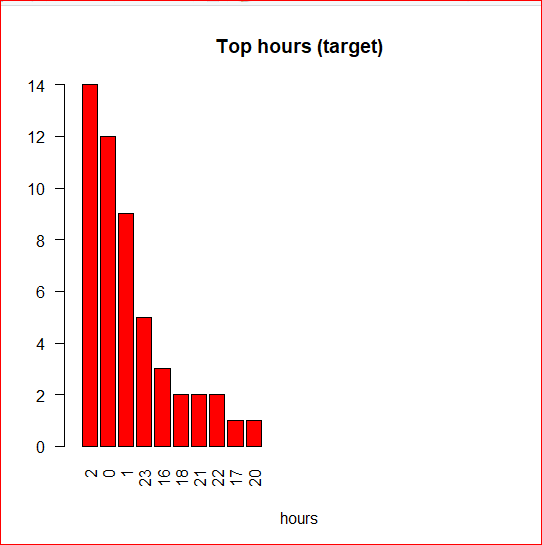
barplot(count.day0, main="Top days (not target)", xlab="days", col="darkblue", las=2)

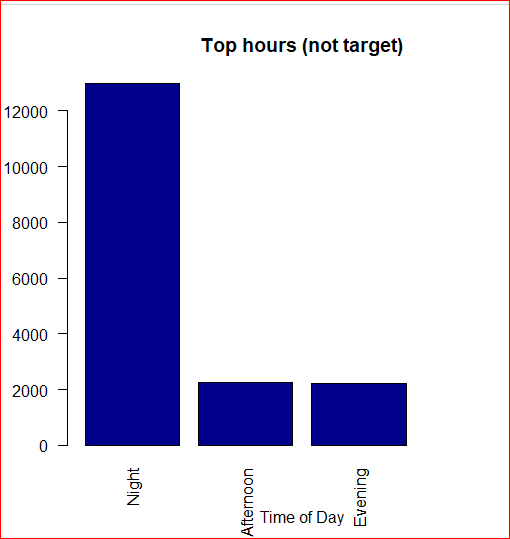
barplot(count.day1, main="Top days (target)", xlab="days", col="red", las=2)

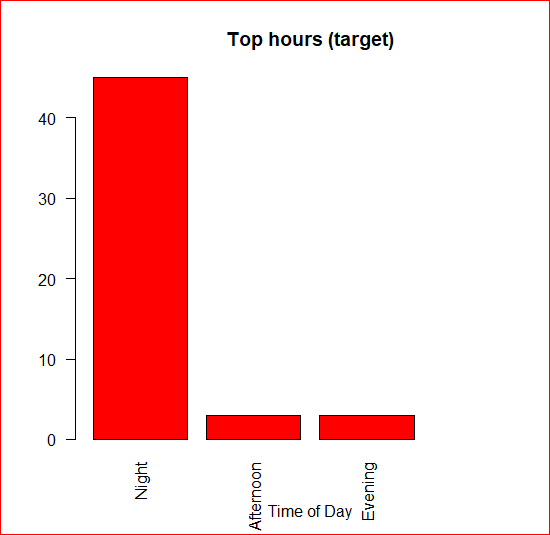


The attribute, “hour” was looked at as well, however, having the data spread out across the 24 hours in the day was hard to examine and interpret. The levels of the hours were then condensed down to morning (5-11), afternoon (12-4), evening (5-7), and night (everything else) for easier interpretation of the data in this attribute. Between the two graphs, it again shows similar distribution between the two groups indicating that this attribute may not be relevant for our analysis.





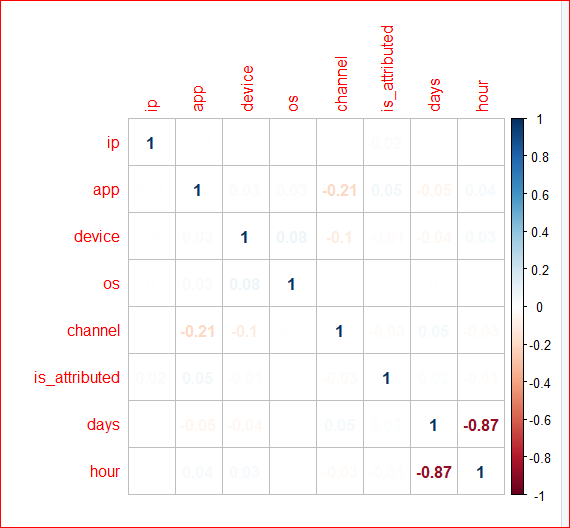




Now that the data is visualized, we can start to answer some of the questions that were set in the abstract.

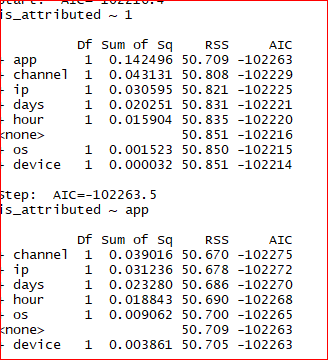
* + Are there particular times in the days where a user will download an application?
    - The red graph shows that night seems to be the time of the day where users would download an application. However, comparing the two graphs from above also shows that there was little differences in terms of times of the day between the target and non-target variable. What is interesting to note is that in both the target and the non-target variable that was fewer clicks and downloads in the evening.
  + Do fraudulent clicks have any relations to the types of mobile devices used?
    - For this question, there is no differences in both the target and the non target-variable as both graphs shows the same distribution. It should be noted that the majority of devices used are in the category valued 2.
  + Are there particular types of an application that a user will be more likely to download?
    - There seems to be differences between the apps downloaded as demonstrated in the graphs for the target and non-target variable. Only the application valued at 9 was consistent between the two graphs.

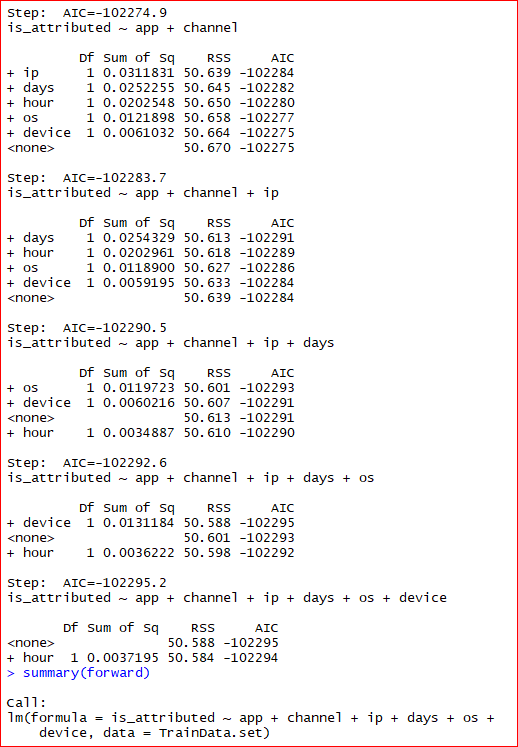
Correlation plot was also done to examine if any attributes were highly correlated to each other. Spearman was used since all the attributes are based on ordinal data and not on interval data.

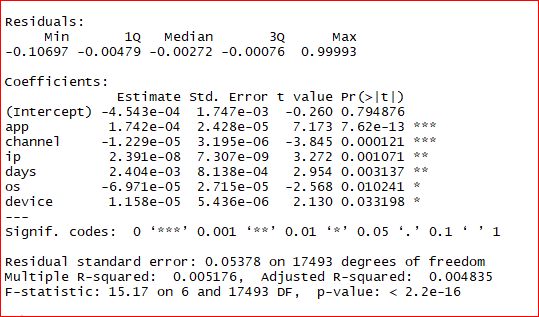


Step 3: Feature selection

Feature selection was done using StepAIC in the forward direction. As seen below, only “ip”, “app”, “channel” and “os” attributes were determined to be significant and should be used as features for the modelling process.







A look into the patterns from visualizing the data into graphs in the exploratory analysis section seems to support the features selected from the StepAIC in the forward direction. The attributes, “device”, “days” and “hour” were determined earlier that they may not be relevant for analysis and modelling.

Step 5: Evaluation of models

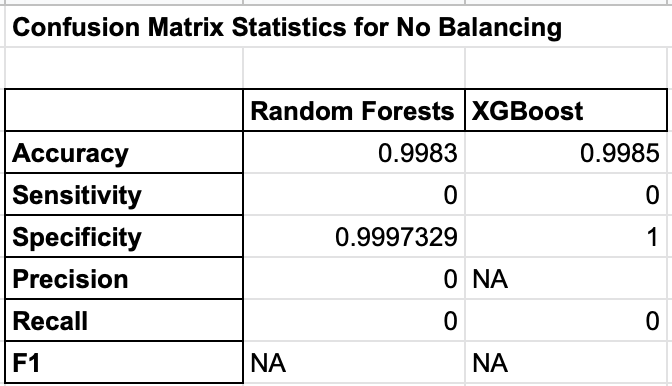
Confusion matrix and Area Under the Curve (AUC) based on the Receiver Operating Characteristic Curve (ROC) curve were used for evaluations of the models. Properties of the confusion matrix such as accuracy, sensitivity, specificity, precision, recall and F1 score are also considered. AUC - ROC curve considers the true positive rate (TPR) against the false positive rate (FPR) through varying thresholds. The greater the AUC - ROC score, it would then indicate better classification performance of the model. The AUC - ROC score would also allow us to compare the different models. In addition, AUC - ROC curve was used over Precision-Recall (PR) curve due to the fact that we will be balancing our datasets using SMOTE and oversampling. Statistical significance was also considered for the models by determining any differences from the resampling processes which was held constant across the different models by using the same seed. The paired t-test is assumed as the resampling distribution is the same between models because of the same seed, thus, allowing us to see if there are any significant differences between the models. The function also assumes the bonferroni correction for p-value adjustment for multiple comparisons. Null hypothesis was already set in that there are no differences between the models while the alternative hypothesis is to reject the null hypothesis in which there are differences between the models.

**Coding for each of the steps can be found in each of the respective areas in the FinalClickFraud.R file from the link: *https://github.com/PHLHY/Capstone-***

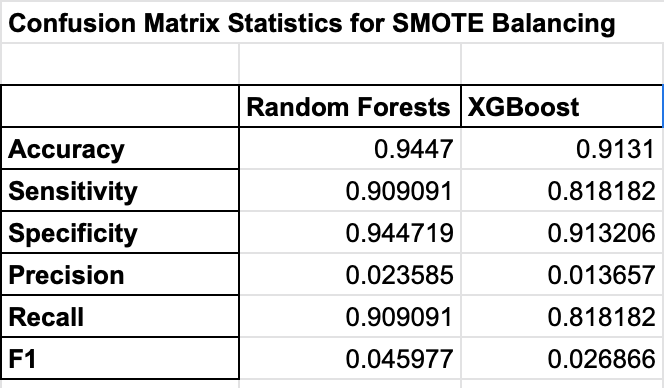
**Results**

Confusion matrix statistics is the first evaluative metric for consideration on the unbalanced dataset, balancing with SMOTE, and finally with oversampling using the RandomForest and XGBoost algorithms.

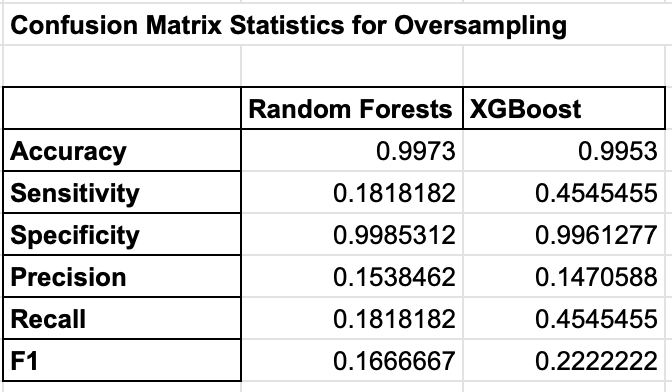
Starting with the no balancing dataset, both algorithms shows a high percentage of accuracy, however, both RandomForest and XGboost does have poor sensitivity affecting precision, recall, and F1 scores. Sensitivity was 0 in both cases as the models did not accurately identified any true target variables. So far, both of these no balancing models are not effective based on the confusion matrix statistics.



The next of modelling was done by balancing the training dataset using SMOTE while using the same two algorithms, RandomForest and XGBoost. Just like above, accuracy is in the 90s for both the RandomForest and XGBoost algorithms. However, while sensitivity and specificity are high compared to the no balancing models, it is noticeable that the precision for both algorithms suffered. The reason for this is because a high proportion that was identified as the target variable in the algorithms were actually not relevant. The confusion matrix of both the RandomForest and XGBoost using SMOTE shows a high amount of false positives (type 1 error) which affected the precision and the F1 scores.



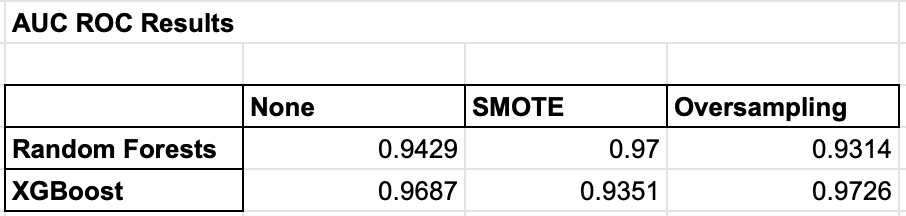
The last approach was done using the oversampling balancing technique, and again both algorithms have high accuracies in the 90s as well. Sensitivity in both of these algorithms are lower than the models that were balanced using SMOTE. However, it does have a higher precision score and resulting F1 scores for both the RandomForest and XGBoost algorithms compared to the other models.



So far, it is safe to assume that accuracy would not be a good indicator for evaluation as all the models are in the 90s. Based on research articles from the literature review section, precision is the key metric that we are aiming for here as we are trying to determine users downloading an application when clicking on a relevant advertisement. Based on this, RandomForest and XBoost with oversampling balancing had the highest precision scores. With consideration of recall, the F1 score would also be an appropriate evaluative measure as it balances out both precision and recall. However, since the overall precision and the F1 scores are low, another evaluative measure should be considered at this time.

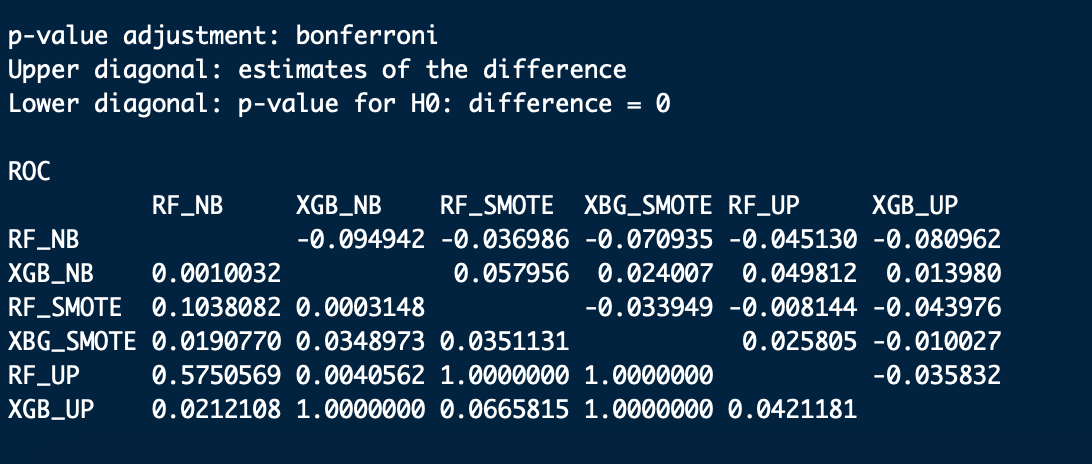
To further analyze the results, another evaluative measure was done based on the Area of the Curve (AUC) of the Receiver Operating Characteristic (ROC) curve.

Based on the AUC - ROC curve, we can see that all algorithms and balancing methods are in the 90s which is usually a good score. As mentioned prior, a high AUC - ROC score is desirable, however, all of our models appears to scored similarly to each other.

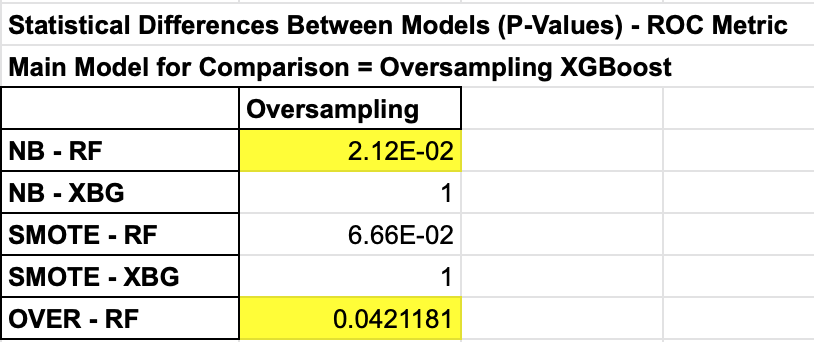


Statistical significance for the models should be looked at as the AUC - ROC scores appears similar to each other and it should be determined whether the differences in scores are significant. The p-value (which is shown below the diagonal in the chart below) is used to distinguish whether a model is significantly different to other models. A p-value of less than 0.05 is assumed to reject the null hypothesis.

Since the AUC is based on the ROC curve, the ROC was used for comparison between the models to see if the differences are significant.



In this case, XGBoost which is balanced by oversampling will be the main model for comparison due to it having the highest AUC - ROC score. The following comparisons are made below with the cell being filled by yellow indicating that there is a significant differences between oversampling XGBoost and to no balancing RandomForest and to oversampling RandomForest. From the AUC - ROC scores seen above, we can see this made sense as they had lower AUC - ROC scores and that the higher AUC - ROC score is in favor of the oversampling XGBoost model. For the rest of the models, we can not reject the null hypothesis.



**Conclusion**

The Kaggle challenge was to build an algorithm for determination of users that will download a mobile application after clicking on a relevant advertisement. A test dataset was provided by TalkingData for this competition to apply the algorithm of choice for submission. Based from the results using the evaluative measures above, the AUC - ROC scores should be the main evaluative metric used as it has scores that are high compared to the statistics that are found in the confusion matrix. Since **oversampling XGBoost model** has the highest AUC - ROC score and is significant different compared to two of the other models in regards to the ROC, **this model will be the choice selection for this Kaggle challenge**. In addition, while the overall precision and F1 scores are low from the confusion matrix, oversampling XGBoost again also has the second highest precision score and the highest F1 score further supporting that it should be the choice model for this competition.

Some limitations of this project are identified below:

Computational limitations

* Dataset was reduced significantly to only 25000 points for training and testing. This could impact on important patterns not seen and potentially affect the training of models
* Some of the attributes were not converted to categorical data type due to the many different levels resulting in memory errors while running models. Attempts were made at reducing the amount of levels, however, a significant amount of information was lost as a result (ie. 15046 unique ips in the reduced set)

Further exploratory analysis and feature engineering

* “Attributed\_time” was not examine closely as it would have provided similar information to “is\_attributed”. However, differences in time between “click\_time” and “attributed\_time” could have resulted in a new attribute which could have been potentially important.
* Potentially all attributes could have been included for modelling as a benchmark for comparison

Modelling

* Preprocessing was not done (ie. standardizing such as scaling and centering) due to all algorithms mostly being tree-based and not on calculating distances. Perhaps preprocessing could have been considered to see how it affected the evaluation of the models.
* Could have considered different types of balancing (ie. ROSE, undersampling etc.)