Harvard University Professional Certificate in Data Science Capstone Project: Ad Tracking Fraud Detection

Nitya Rudraraju

January 10th 2020

Executive Summary

The Dataset

China is the largest mobile market in the world. With over 1 billion active smart mobile devices, China has a concern with the large volumes of fraudulent ad click traffic. Click fraud can occur at a very significant volume, resulting in misleading click data, which therefore influences the price from ad channels.

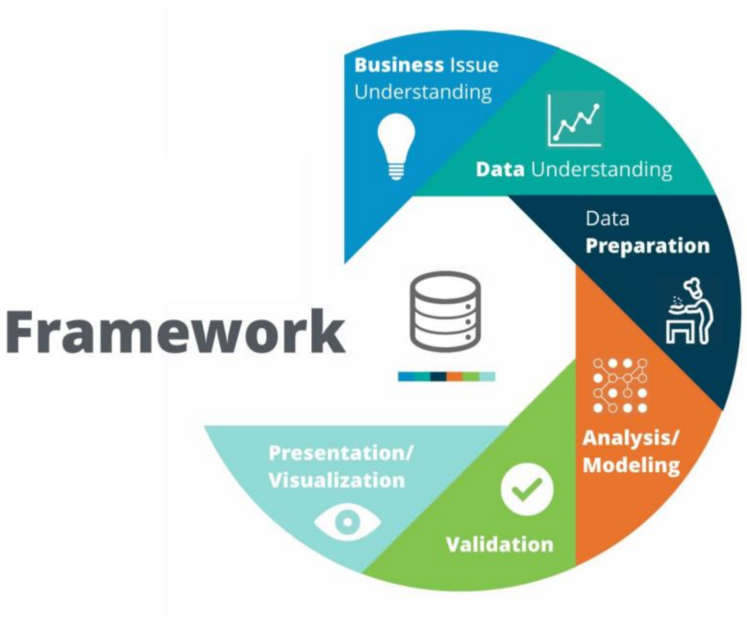
The data used in this capstone is from the "TalkingData AdTracking Fraud Detection Challange" launched on Kaggle about 10 months ago. The data was provided by China's largest independent big data service platform (covers over 70% of active mobile devices in the country), TalkingData.

The Assignment

This project was inspired by the "TalkingData AdTracking Fraud Detection Challenge" on Kaggle, where the goal is to predict whether a user will download an app after clicking a mobile app ad. TalkingData's current approach for detecting fraud is to measure the journey of a user's click across their portfolio and flag IP addresses who produce various clicks and yet never downloads an app. This is an assumed indication that the ad clicking is fraudulent, rather than legitimate. With this information, they maintain an IP and device blacklist, which ultimately prevents wasted funds invested in ad channels.

Methods & Process

My approach and process follows the "Cross-Industry Process for Data Mining (CRISP-DM)" methodology very closely with some minor alterations as to honor the purpose of the course capstone. This process includes the following steps:



We have already briefly gathered an understanding of the business issue, which was originally presented by TalkingData as a Kaggle competition. The next steps will include understanding the data, cleaning the data, undergoing an exporatory data analysis (or EDA), followed by modeling the data and validating the results. The algorithms used to predict the binary outcome of "download" or "no download" are: **decision tree, random forest, linear support vector machine and radial kernal support vector machine**.

The final step is this final product (an R script, Rmd, and PDF file). Below details the exact steps taken in this capstone project.

Preliminary Look at the Data

Now that the data is loaded, we can begin taking a preliminary peak at the data to determine its components and structure.

```
#Explore
str(train)

## 'data.frame':    100000 obs. of  8 variables:
##  $ ip             : int  87540 105560 101424 94584 68413 93663 17059 12150
5 192967 143636 ...
##  $ app            : int  12 25 12 13 12 3 1 9 2 3 ...
##  $ device         : int  1 1 1 1 1 1 1 1 2 1 ...
##  $ os             : int  13 17 19 13 1 17 17 25 22 19 ...
##  $ channel        : int  497 259 212 477 178 115 135 442 364 135 ...
##  $ click_time     : chr  "2017-11-07 09:30:38" "2017-11-07 13:40:27" "2017
-11-07 18:05:24" "2017-11-07 04:58:08" ...
##  $ attributed_time: chr  "" "" "" "" ...
##  $ is_attributed  : int  0 0 0 0 0 0 0 0 0 0 ...

str(test_valid)

## '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 1493
2 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" "2017-11-1
0 04:00:00" "2017-11-10 04:00:00" ...

head(train)

##       ip app device os channel          click_time attributed_time
## 1  87540  12      1 13     497 2017-11-07 09:30:38
## 2 105560  25      1 17     259 2017-11-07 13:40:27
## 3 101424  12      1 19     212 2017-11-07 18:05:24
## 4  94584  13      1 13     477 2017-11-07 04:58:08
## 5  68413  12      1  1     178 2017-11-09 09:00:09
## 6  93663   3      1 17     115 2017-11-09 01:22:13
##   is_attributed
## 1             0
## 2             0
## 3             0
## 4             0
## 5             0
## 6             0

table(train$is_attributed)
```

```
##
##     0     1
## 99773   227
```

```
#Missing Data?
colSums(is.na(train)) #none
```

```
##              ip             app          device              os
##               0               0               0               0
##         channel      click_time attributed_time   is_attributed
##               0               0               0               0
```

Luckily, there is no missing data in our dataset. Additionally, we can see there is a difference between the data provided in the `train` and `test_valid` datasets. We will take care of this later. For now, we will work primarily with the train set, which will be partitioned into a train set and test set to see if our models generalize to the full `train` dataframe effectively. We will also reformat our `click_time` feature by extracting specific time information such as year and month.

Data Cleaning & Feature Engineering

First, we will remove the `attributed_time` feature since it isn't present in the `test_valid` dataset.

```
train$attributed_time=NULL
```

Next, I will engineer the time feature into multiple features.

```
#Reformat click_time feature
train$click_time<-as.POSIXct(train$click_time,
                             format = "%Y-%m-%d %H:%M",tz = "America/New_York
")

train$year <- year(train$click_time) #year
train$month <- month(train$click_time) #month
train$days <- weekdays(train$click_time) #weekdays
train$hour <- hour(train$click_time) #hour

#Remove original feature now that needed information is extracted into
#new features
train$click_time=NULL
```

Now, let's take a look at how many unique variables we have in each column.

```
#Determine number of unique observations per column
apply(train,2, function(x) length(unique(x)))
```

```
##            ip           app        device            os       channel
##         34857           161           100           130           161
## is_attributed          year         month          days          hour
##             2             1             1             4            24
```

As we can see above, the data only reflects information that was collected over a single month within a single year. Because these fields don't provide helpful information, we will remove the month and year fields.

```
train$month=NULL #only 1 month present: feature no longer needed
train$year=NULL #only 1 year present: feature no longer needed
```

Lastly, we will factorize our binary response variable (`is_attributed`) and days of the week.

```
#Format appropriate columns as factors
train$is_attributed=as.factor(train$is_attributed)
train$days=as.factor(train$days)
```

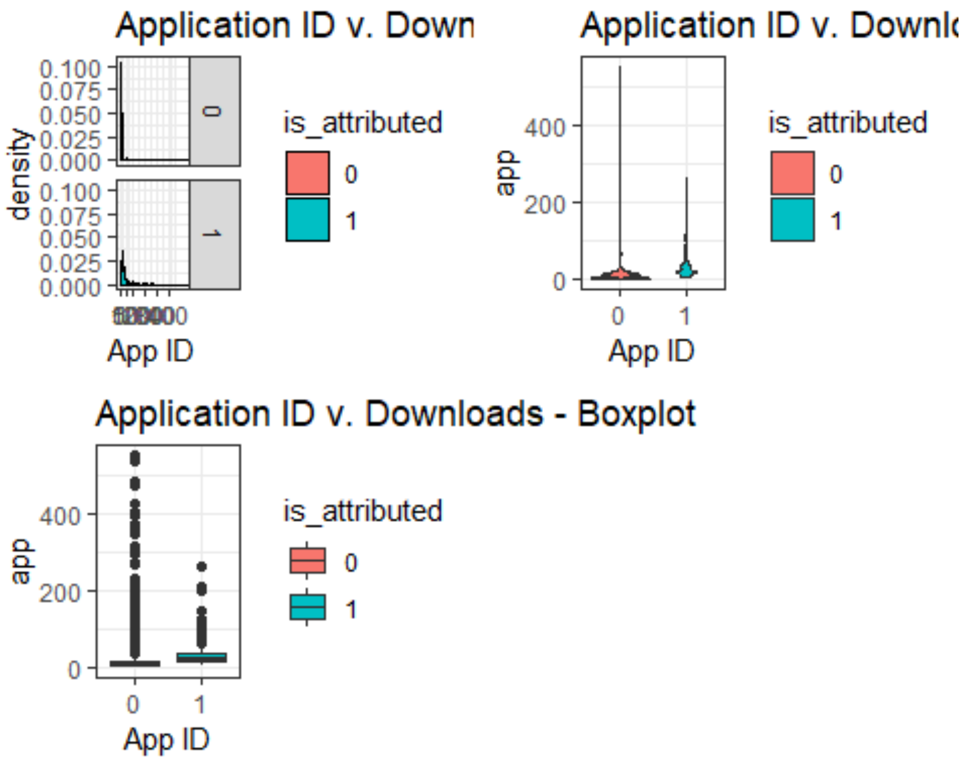
Exploratory Data Analysis (EDA) & Visualization

Now that our data is prepared for analysis, the EDA process begins. The purpose of the exploratory data analysis is to discover insights provided by the data. More specifically, I am interested in discovering which features will make good predictors of app downloads. That is, *"...which fields are significantly related to the response variable?"*.

First, we will explore the relationship between app downloads (`is_attributed`) and app ID. We will mainly use visualization techniques that best show feature distributions: density plot, violin plot and boxplot.

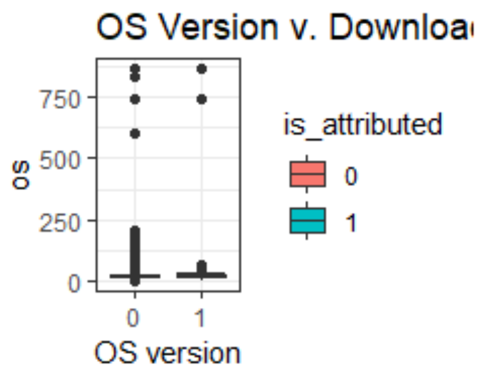
```
p1 <- ggplot(train,aes(x=app,fill=is_attributed)) +
  geom_density()+
  facet_grid(is_attributed~.) +
  scale_x_continuous(breaks = c(0,50,100,200,300,400)) +
  ggtitle("Application ID v. Downloads - Density plot") +
  xlab("App ID") +
  labs(fill = "is_attributed") +
  theme_bw()


p2 <- ggplot(train,aes(x=is_attributed,y=app,fill=is_attributed) )+
  geom_violin() +
  ggtitle("Application ID v. Downloads - Violin plot") +
  xlab("App ID") +
  labs(fill = "is_attributed") +
  theme_bw()

p3 <- ggplot(train,aes(x=is_attributed,y=app,fill=is_attributed)) +
  geom_boxplot() +
  ggtitle("Application ID v. Downloads - Boxplot") +
  xlab("App ID") +
  labs(fill = "is_attributed") +
  theme_bw()
```

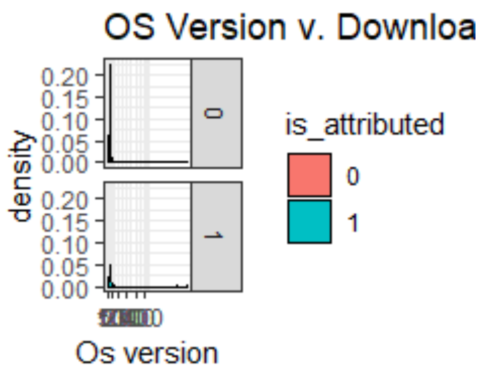
```r
grid.arrange(p1,p2,p3, nrow = 2, ncol = 2)
```

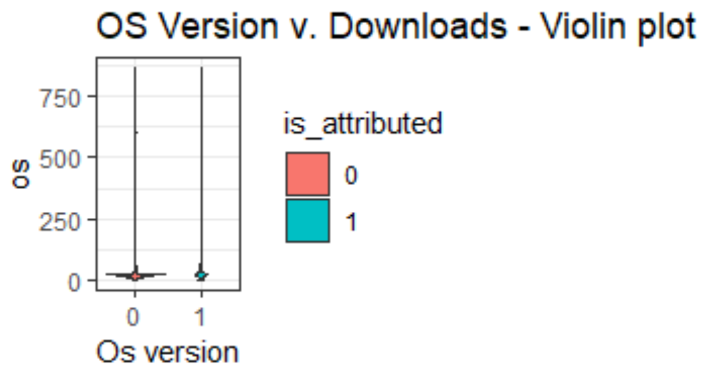


This will be a helpful feature to determine whether a user downloaded an app or not.

I create a similar graph grid for the response variable vs. the OS version (os):

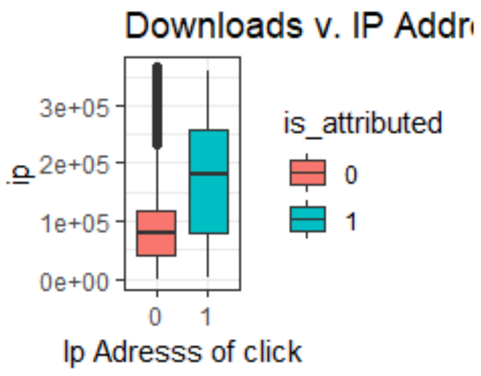


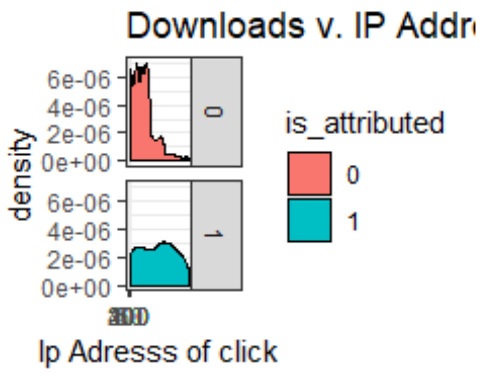


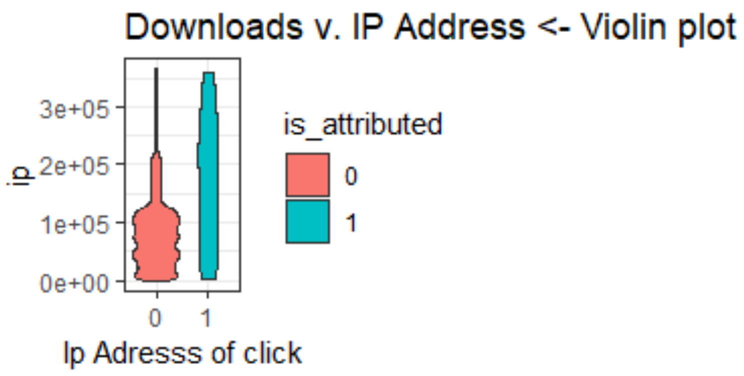


There doesn't appear to be a very strong relationship between the two. We will remove this feature.

Next, we look at IP address (`ip`):

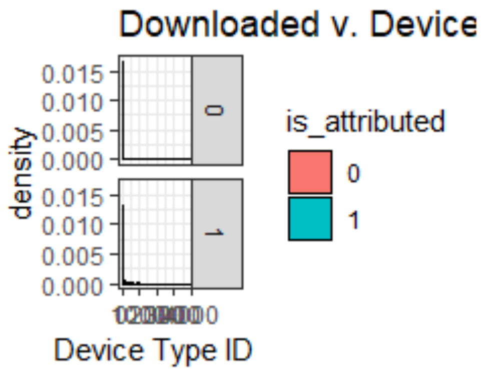


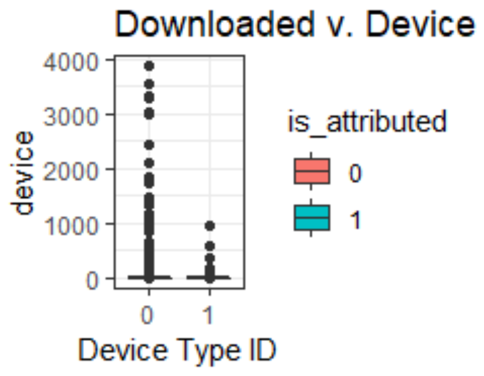


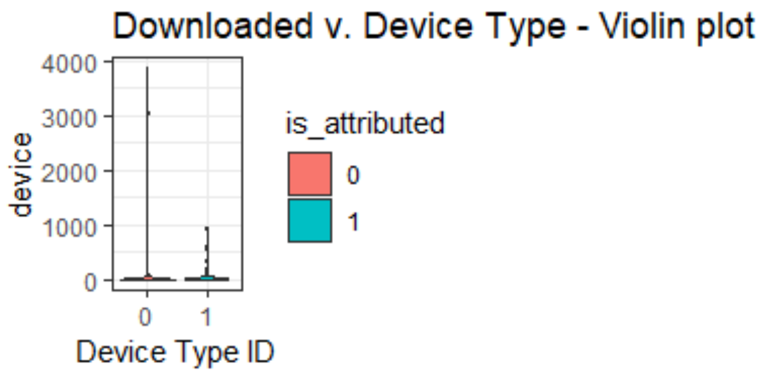


We can clearly see a very strong relationship between the distributions of IP address and downloads in all 3 graphs. I will retain this feature.

Next, we pair our response variable with device type (`device`):

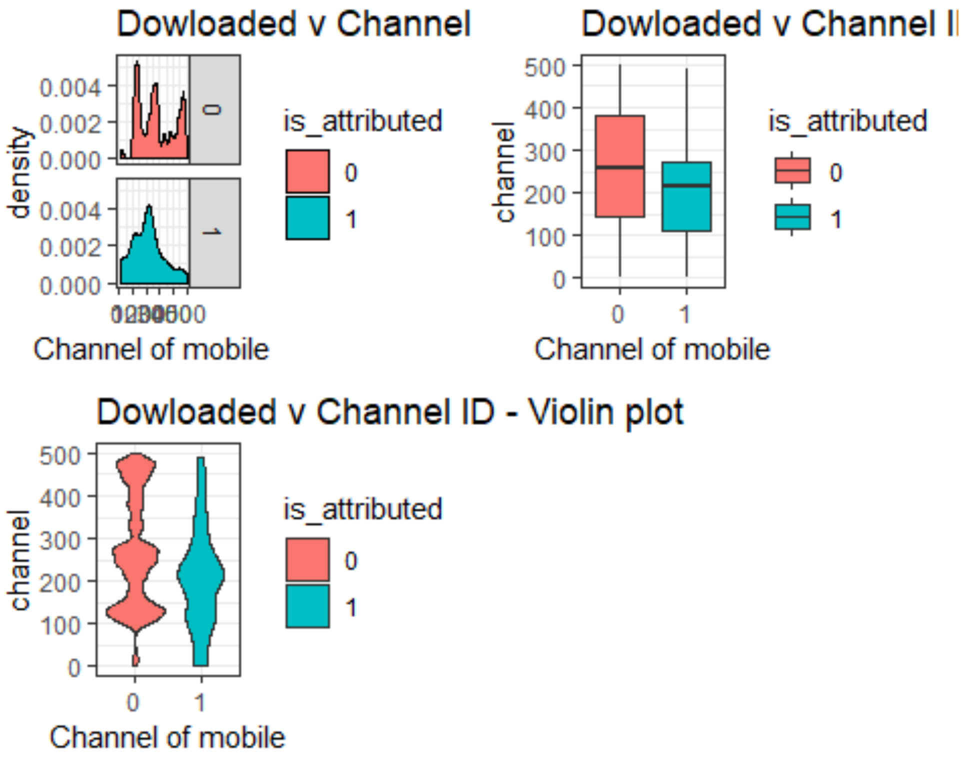






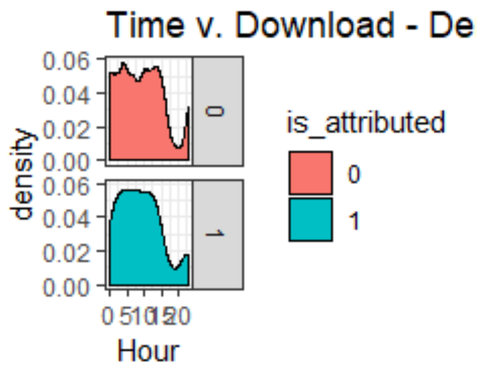
We do not see a strong indication of differentiation in any of the above charts. This feature will be removed.

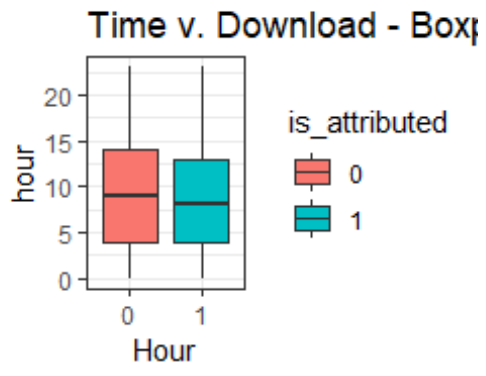
Next, `channel`, or channel ID:

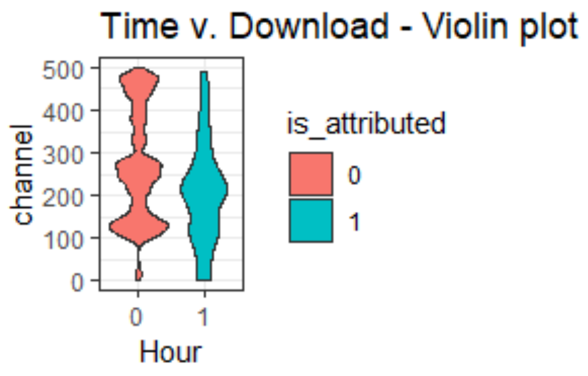


Much like the IP address distributions, we can clearly see a strong differentiation in the distributions of channel ID and our response variable. Given its predictive power, we will retain the `channel` feature.

Next, let's see if time is relevant. First, we'll explore the hour of the day:

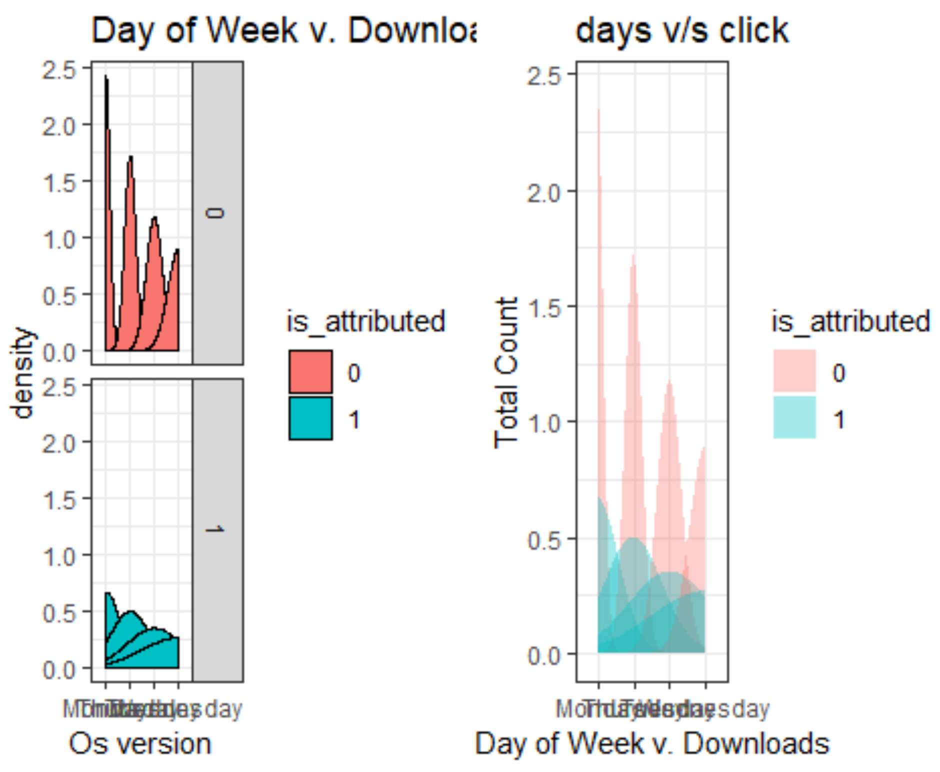






There is a small observable difference in the average number of downloads for both "download" and "no download" groups. It's not much (as we will quantify later using the AUC method), but we will keep this feature.

Lastly, let's see how the days of the week, `days`, vary in comparison with our response variable:



No strong differentiation.

Cross Validation & Modeling

Let's begin some modeling. First (for the sake of comparison), I will model on *all features* as opposed to the ones we've selected earlier in the EDA section.

I begin to design my multi-folded, cross validation:

```
#Cross Validation
set.seed(1)
cv.10 <- createMultiFolds(train$is_attributed, k = 10, times = 10)
myControl <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                          index = cv.10)
```

The first algorithm we will explore is the **decision tree**.

```
set.seed(1)
dt <- caret::train(x = train[,-6], y = train[,6], method = "rpart", tuneLengt
h = 30,
                   trControl = myControl)
```

Next, I will create a confusion matrix and store its results in a results table called `model.results`. We will review these results later in the evaluation phase.

```
pred <- predict(dt$finalModel,train, type = "class")
confusionMatrix(pred,train$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction      0      1
##           0 99763    176
##           1    10     51
##
##                Accuracy : 0.9981
##                  95% CI : (0.9979, 0.9984)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.002836
##
##                   Kappa : 0.3535
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9999
##             Specificity : 0.2247
##          Pos Pred Value : 0.9982
##          Neg Pred Value : 0.8361
##              Prevalence : 0.9977
##          Detection Rate : 0.9976
##    Detection Prevalence : 0.9994
##       Balanced Accuracy : 0.6123
##
##        'Positive' Class : 0
##

#Decision Tree #1 Performance
dt.misclass <- mean(train$is_attributed != pred)
dt.accuracy <- 1 - dt.misclass
dt.f1 <- F_meas(data = pred, reference = factor(train$is_attributed))

model.results <- data.frame(method = "Decision Tree: All Feat.",
                           misclass = dt.misclass,
                           accuracy = dt.accuracy,
                           f1_Score = dt.f1) #stores results
```

In the table, I include the misclassification rate, accuracy and F1 score. We will compare these results with other algorithms in the “Results Evaluation” section later in the report.

Next, I create the same decision tree model, only this time on our select featues:

```
#Remove undesired features
train$days=NULL
train$os=NULL
train$device=NULL

#Model decision tree on select features
set.seed(1)
dt.2 <- caret::train(x = train[,-4], y = train[,4], method = "rpart", tuneLen
gth = 30,
                    trControl = myControl)
```

Now, we calculate our key metrics of performance and assign it to the `model.results` object.

```
pred.2 <- predict(dt.2$finalModel,data = train,type="class")
confusionMatrix(pred.2,train$is_attributed) #better specificity

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 99754   167
##          1    19    60
##
##                Accuracy : 0.9981
##                  95% CI : (0.9979, 0.9984)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 0.002836
##
##                   Kappa : 0.3914
##  Mcnemar's Test P-Value : < 2.2e-16
##
##             Sensitivity : 0.9998
##             Specificity : 0.2643
##          Pos Pred Value : 0.9983
##          Neg Pred Value : 0.7595
##              Prevalence : 0.9977
##          Detection Rate : 0.9975
##    Detection Prevalence : 0.9992
##       Balanced Accuracy : 0.6321
##
##        'Positive' Class : 0
##

#Decision Tree 2 Performance Accessment
dt.2.misclass <- mean(train$is_attributed != pred.2)
dt.2.accuracy <- 1 - dt.2.misclass
dt.2.f1 <- F_meas(data = pred.2, reference = factor(train$is_attributed))

model.results <- bind_rows(model.results,
                           data.frame(method = "Decision Tree: Select Feature
s",
                                      misclass = dt.2.misclass,
                                      accuracy = dt.2.accuracy,
                                      f1_Score = dt.2.f1))#add results to tab
le

## Warning in bind_rows_(x, .id): Unequal factor levels: coercing to characte
r
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

If I print the object, you'll notice that the performance of the 2 decision trees are nearly identical…

```
print(model.results)

##                           method misclass accuracy  f1_Score
## 1       Decision Tree: All Feat.  0.00186  0.99814 0.9990687
## 2 Decision Tree: Select Features  0.00186  0.99814 0.9990686
```

However, you will note that the specificity is better in the second decision tree.

Specificity.Tree1	Specificity.Tree2
0.2246696	0.2643172

Now that we've identified a relatively reliable method (decision trees), let's move forward with generalizing our models to a test set (unseen data).

First, I partition the data into a train and test set:

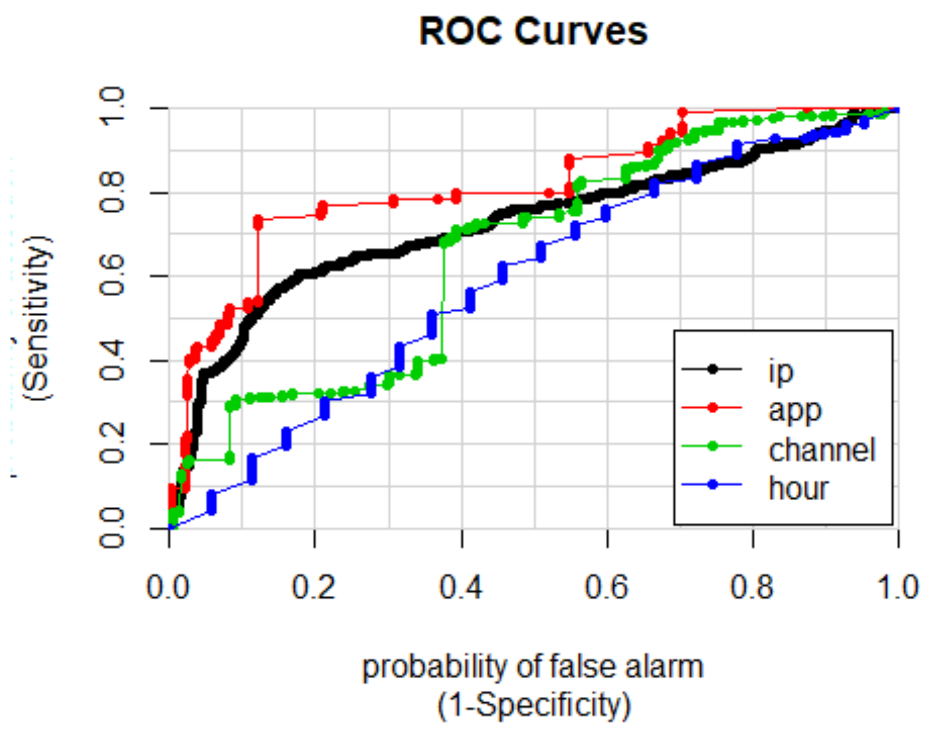
```
set.seed(1)
train_index <- createDataPartition(train$is_attributed,times=1,p=0.7,list=FAL
SE)
train <- train[train_index,]
test <- train[-train_index,]
```

Next, I need to address how unbalanced the data is. That is, there are far more non-downloads than downloads. To resolve this matter, I want to rebalance my data. I do this, using the SMOTE ("Synthetic Minority Oversampling") techique. This is a statistical technique for increasing the number of cases in your dataset in a balanced way.

```
#Rebalance the data since there is a low prevelance of downloads
set.seed(1)
smote.train = SMOTE(is_attributed ~ ., data  = train)
table(smote.train$is_attributed)

## 
##   0   1 
## 636 477
```

As you can see, the data is better balanced now. Because we have our selected features and rebalanced data, now would be agreat time to take a look at the resulting ROC Curve to review the TP and FP tradeoff for each feature:

```
colAUC(smote.train[,-4],smote.train[,4], plotROC = TRUE)
```



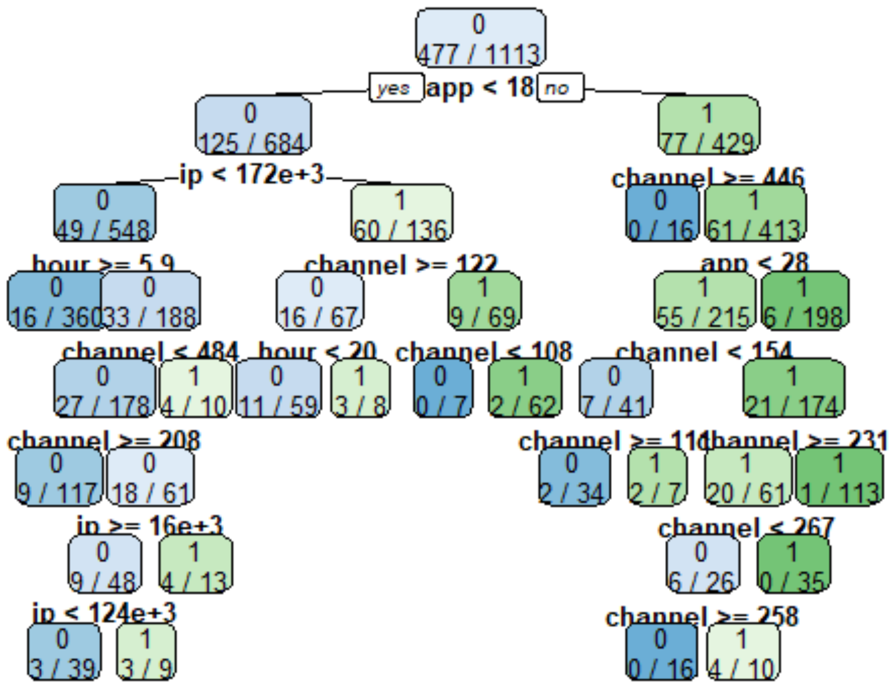
```
##                   ip       app   channel      hour
## 0 vs. 1 0.7252466 0.8160839 0.6563856 0.5892633
```

Based on the visualization and the AUC outputs, the app ID is the best discriminant. Just as expected from our EDA section, the hour feature performs the worst with an AUC of 0.58.

Now, I re-attempt the decision tree on select features, only this time with the rebalanced data

```
set.seed(1)
cv.10 <- createMultiFolds(smote.train$is_attributed, k = 10, times = 10)
myControl <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                          index = cv.10)

set.seed(1)
dt.3 <- caret::train(x = smote.train[,-4], y = smote.train[,4], method = "rpa
rt", tuneLength = 30,
                      trControl = myControl)
```

You can see a plot of the results here:



Now, we add the results to our table the same as I did in the previous examples. Note that the predictions were made on the test dataset this time to generalize the model. This will be the case moving forward:

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 19610     3
##          1  1403    45
##
##                Accuracy : 0.9332
##                  95% CI : (0.9298, 0.9366)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.056
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.93323
##             Specificity : 0.93750
##          Pos Pred Value : 0.99985
##          Neg Pred Value : 0.03108
##              Prevalence : 0.99772
##          Detection Rate : 0.93110
##    Detection Prevalence : 0.93125
##       Balanced Accuracy : 0.93537
```

```
##
##        'Positive' Class : 0
##
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

```
##                           method   misclass  accuracy  f1_Score
## 1       Decision Tree: All Feat. 0.00186000 0.9981400 0.9990687
## 2 Decision Tree: Select Features 0.00186000 0.9981400 0.9990686
## 3    Decision Tree: + Rebalanced 0.06675846 0.9332415 0.9653916
```

As a natural progression, I will try a **random forest** model on the reblanced, select features:

```
#Random Forest Model, select features + rebalanced data
set.seed(1)
cv.10 <- createMultiFolds(smote.train$is_attributed, k = 10, times = 10)
myControl <- trainControl(method = "repeatedcv", number = 10, repeats = 10,
                          index = cv.10)

set.seed(1)
set.seed(1)
rf<- caret::train(x = smote.train[,-4], y = smote.train[,4], method = "rf", t
uneLength = 3,
                  ntree = 100, trControl = myControl)
```

Then, the results are stored.

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 20193     0
##          1   820    48
##
##                Accuracy : 0.9611
##                  95% CI : (0.9584, 0.9636)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.1009
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.9610
##             Specificity : 1.0000
##          Pos Pred Value : 1.0000
##          Neg Pred Value : 0.0553
##              Prevalence : 0.9977
##          Detection Rate : 0.9588
##    Detection Prevalence : 0.9588
##       Balanced Accuracy : 0.9805
##
##        'Positive' Class : 0
##
```

```
## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

```
##                             method   misclass  accuracy  f1_Score
## 1       Decision Tree: All Feat. 0.00186000 0.9981400 0.9990687
## 2 Decision Tree: Select Features 0.00186000 0.9981400 0.9990686
## 3    Decision Tree: + Rebalanced 0.06675846 0.9332415 0.9653916
## 4                  Random Forest 0.03893452 0.9610655 0.9801000
```

Let’s try another model. This time, I will use a **linear support vector machine**.

```
set.seed(1)
svm.model <- tune.svm(is_attributed~.,data=smote.train, kernel="linear", cost
=c(0.1,0.5,1,5,10,50))

best.linear.svm <- svm.model$best.model
pred.svm.lin <- predict(best.linear.svm,newdata=test,type="class")

#Performance Evaluation
confusionMatrix(pred.svm.lin,test$is_attributed)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 18147    14
##          1  2866    34
##
##                Accuracy : 0.8633
##                  95% CI : (0.8585, 0.8679)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0187
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.86361
##             Specificity : 0.70833
##          Pos Pred Value : 0.99923
##          Neg Pred Value : 0.01172
##              Prevalence : 0.99772
##          Detection Rate : 0.86164
##    Detection Prevalence : 0.86230
##       Balanced Accuracy : 0.78597
##
```

```
##        'Positive' Class : 0
##

dt.5.misclass <- mean(test$is_attributed != pred.svm.lin)
dt.5.accuracy <- 1 - dt.5.misclass
dt.5.f1 <- F_meas(data = pred.svm.lin, reference = factor(test$is_attributed)
)

model.results <- bind_rows(model.results,
                           data.frame(method = "Linear Support Vector Machine
",
                                      misclass = dt.5.misclass,
                                      accuracy = dt.5.accuracy,
                                      f1_Score = dt.5.f1))#add results to tab
le

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

Lastly, I will use and evaluate a **radial kernal support vector machine**

```
#Radial Kernal Support Vector Machine (SVM)
set.seed(1)
svm.model.2 <- tune.svm(is_attributed~.,data=smote.train,kernel="radial",gamm
a=seq(0.1,5))
summary(svm.model.2)

##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma
##    4.1
##
## - best performance: 0.1069659
##
## - Detailed performance results:
##   gamma     error dispersion
## 1   0.1 0.1734878 0.04039412
## 2   1.1 0.1258526 0.04743286
## 3   2.1 0.1168678 0.04585481
## 4   3.1 0.1105534 0.04420488
## 5   4.1 0.1069659 0.04304597

best.radial.svm <- svm.model.2$best.model

pred.svm.rad <- predict(best.radial.svm,newdata = test)

#Performance Evaluation
```

```
confusionMatrix(pred.svm.rad,test$is_attributed)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 19408     5
##          1  1605    43
##
##                Accuracy : 0.9236
##                  95% CI : (0.9199, 0.9271)
##     No Information Rate : 0.9977
##     P-Value [Acc > NIR] : 1
##
##                   Kappa : 0.0465
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.92362
##             Specificity : 0.89583
##          Pos Pred Value : 0.99974
##          Neg Pred Value : 0.02609
##              Prevalence : 0.99772
##          Detection Rate : 0.92151
##    Detection Prevalence : 0.92175
##       Balanced Accuracy : 0.90973
##
##        'Positive' Class : 0
##

dt.6.misclass <- mean(test$is_attributed != pred.svm.rad)
dt.6.accuracy <- 1 - dt.6.misclass
dt.6.f1 <- F_meas(data = pred.svm.rad, reference = factor(test$is_attributed)
)

model.results <- bind_rows(model.results,
                           data.frame(method = "Radial Kernal Support Vector
Machine",
                                      misclass = dt.6.misclass,
                                      accuracy = dt.6.accuracy,
                                      f1_Score = dt.6.f1))#add results to tab
le

## Warning in bind_rows_(x, .id): binding character and factor vector,
## coercing into character vector
```

Now that we have all of our models and their performances saved to `model.results`, let's evaluate them:

Results Evaluation

```
kable(model.results) #First 2 models were trained and tested on same (train)
data.
```

method	misclass	accuracy	f1_Score
Decision Tree: All Feat.	0.0018600	0.9981400	0.9990687
Decision Tree: Select Features	0.0018600	0.9981400	0.9990686
Decision Tree: + Rebalanced	0.0667585	0.9332415	0.9653916
Random Forest	0.0389345	0.9610655	0.9801000
Linear Support Vector Machine	0.1367456	0.8632544	0.9264819
Radial Kernal Support Vector Machine	0.0764446	0.9235554	0.9601741

The first two decision tree models performed well as expected, because they were not predicted on unseen data. Thus, while the results are favorable, it is likely that these models are over fitted and will not generalize to the test set well.

Thus, the third decision tree model is tested on "unseen" data (the test set). We may also note that the third decision tree was tested on select features. Also note that the data was rebalanced to account for the low prevalence of app downloads (which is a very rare occurrence). As expected, it didn't perform as well as the over fitted decision tree models, but it still performed fairly well (ie: 96% F1 score with 93% accuracy).

The **random forest** was a natural progression from the third decision tree model and an enhanced performer with an **accuracy of 96%** and **F1 score of 98%**.

While the linear and radial kernal support vector machines were respectable attempts (the latter even moreso), they did not outperform the random forest model.

Thus, the random forest model is the superior model.

Conclusion

As a conclusion, the random forest was the best model with a superior F1 score, accuracy and misclassification rate once the data is rebalanced and our features were selected.

For future analysis, logistic regression should be considered as well, however since the goal of this capstone was to go beyond regression methods, I omitted it from this exercise.

Additionally, because the hour feature had the worst performance from all predictors, it may be worth removing the feature for future modeling purposes.

I would also recommend to TalkingData to consider additional features, which may be better predictors of fraudulent click activity (ie: type of app, ad channel, ad popularity, etc.).

The final script has results for the predictions on unseen, unclassified `test_valid` data, which should also be evaluated once / if their actual values are collected.