TalkingData: A Data Science Analysis

Andrew Liu, Andrew Milne, Arthur Cockfield, Sonia Tang, Zac Burns

COMM 461

Professor Mikhail Nediak

April 26th, 2018

Problem Description

Fraud risk is a problem that plagues most companies which advertise on mobile phones. With an increasing volume of misleading clicks in the mobile space, companies are wasting millions of dollars preventing fraudulent ad clicks at a large scale. The biggest market for this specific issues is China, with over 1 billion smart mobile devices active every month. With China being the largest mobile market in the world, it suffers from increasing volumes of fraudulent traffic.

TalkingData is China's premier independent big data platform, covering over 70% of mobile devices across the country. While they handle over 3 billion clicks per day, they estimate that over 90% of their clicks are potentially fraudulent. They are currently using a number of strategies to track and flag these fraudulent issues, from tracking user clicks to flagging and blacklisting suspicious IP addresses. However, TalkingData would like a effective strategy to detect further fraud. TalkingData has created a Kaggle data science competition, which our team is entering.

Our team has decided to attempt to develop a classification model for detecting fraudulent clicks, by attempting to identify key variables which help prevent and identify potentially malicious users on this advertising platform. For this project, malicious users are those who produce a large amount of clicks, but never end up installing apps. The goal will be to block and blacklist IP addresses and devices which are classified as fraudulent.

For the problem, TalkingData has released training data with defined target variable of whether the user is fraudulent or not. Our job is to train a model based on this training data, and then test and tune this model so it can be generalizable for their larger data sets. This model will directly improve TalkingData's bottom line, as they will generate cost savings by blacklisting fraudulent traffic.

Business Implications

The problem illustrated by TalkingData, possible solutions, and models we have created highlight a number of business implications. As a team, we used our in-depth analysis to draw three key business implications focused on Fraud Detection, Marketing, and Consumer Insights.

1) Fraud Detection

The problem and solution proposed by our team has highlighted fraud detection applications not only to the mobile advertising space, but to a variety of other industries as well. A few examples of other industries that our models can be applied to could be healthcare, financial institutions, and insurance. With the cost of insurance fraud in these industries estimated to be 40 billion dollars a year, our solution could be used to find fraudulent insurance claims, and therefore catch them before they cost the company more money. In addition, our model could be applied to credit card companies, as these companies lose approximately 7 cents for every hundred dollars in transactions from not paying back their credit card bills. Identify those most at risk to default or not pay their full bill could save financial institutions a ton of money.

2) Marketing

In addition to fraud detection, our proposed solutions can also be applied to various marketing strategies and applications. Companies such as TalkingData can use our insights to gain further analysis into customer intelligence and profitability, which allows them to decrease margins and increase revenue. Furthermore, TalkingData and other companies will be able to apply our proposed models to predict and determine the effectiveness of promotional campaigns, loyalty programs, and how to cross and up-sell.

3) Consumer Insights

Finally, our proposed solution, insights, and analysis can help TalkingData and other companies gain greater insights on their customers. Understanding their customers, and where they are clicking and diverting their attention to regarding mobile advertisements is the key to future success. Extensions of our models can provide further insights and valuable predictions into where users are clicking and what type of mobile advertisements they are responding well to.

<u>Data Collection</u>

All of our data was collected from Talking Data's dataset that was provided for the kaggle competition. The sample set provided included 100000 rows of randomly selected training data with each row containing a click record of the following features; IP address, app, device, os, channel, click time, attributed time and is attributed. Descriptions of each feature are provided in the chart below.

Diving deeper into the data set we used for this project, the data we collected and used can be divided into the following categories:

<u>Data Collected</u>	**<u>Description</u>**
Features	Includes: *IP Addresses, App ID: what app was being used, Device Type: what phone was being used, Operating System Version, Channel ID of mobile ad publisher, Timestamp(time user clicked), Time Attributed(time of app download)*
Target Labels	Predicting whether an app was downloaded or not

Note: For an example of what the data looks like, please refer to Appendix K.

<u>Pre-processing & Data Cleaning</u>

All models were initially built using a training sample provided by the competition. The training sample represented roughly 100 000 rows. All models built to classify fraudulent data had accuracy rate above 99%. It was found that less than 1% of the dataset contained the fraudulent clicks, so by guessing that the data would be not fraudulent every time by default would create an accuracy of about 99%. This meant that a model with a high accuracy may not generalize well over other sets. Additionally, different metrics such as ROC curves showed that a few of our models were roughly as good as guessing (with an AUC of about 0.50)

Because of this problem, it was important to balance the proportion of the classes so that the model would need to deal with both fraudulent and non-fraudulent data in a more equal distribution. Although changing parameters for the models improved performance on the ROC curve (such as setting the scale_pos_weight parameter for the XGBClassifier to 0.5), it did not sufficiently improve the performance even close to what top Kaggle competitors were scoring at (generally over 0.90 on their AUC).

The team then determined that inserting cases where fraudulent clicks were found would be an excellent way to balance out the proportion of the classes. A python script was built to load 30 000 000 rows from the full training set (7.3 GB), and extract any fraudulent clicks. The code for this process can be found on Appendix A at the end of this document. Over 70 000 fraudulent clicks were pulled from this data and written to an excel file. Then, the tables of the training sample and these new fraudulent rows were merged as a new training set.

Features Chosen

The team excluded the time stamp and time attributed features in our modeling after it was discovered they greatly hurt performance for all models on the core metrics of AUC and accuracy. All other features were used in modelling (IP Addresses, App ID, Device Type, OS Version, Channel ID).

Models

For the purpose of the project, we decided to implement a variety of different data science and machine learning techniques to achieve the desired outcomes highlighted by TalkingData and our group. Outlining our techniques below, we decided to focus on implementing the following techniques:

Data Science Technique	**Description**
Aggregate Statistics	Averages, Histograms, Base Rate
Classification	XGBoost, Random Forest, Logistic Regression
Clustering	K-Means, K-Medoids
Time Series	Base rate for clicks, Forecasting of live volume stats

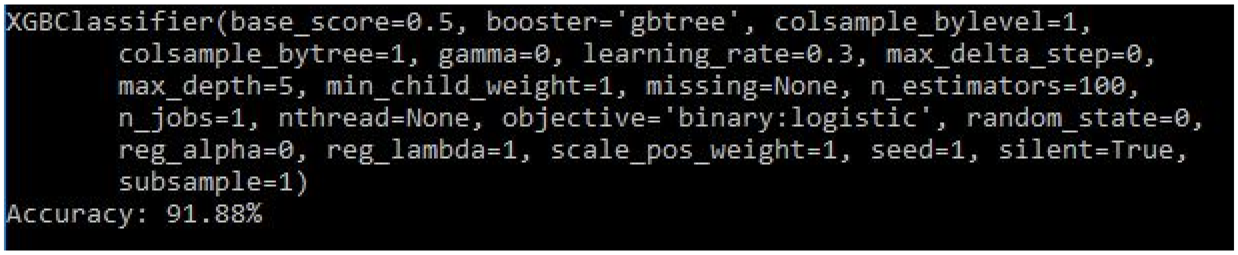
Classification

i) XGBoost

XGBoost was an extremely useful and powerful tool for this project, as it allowed us to implement the extreme gradient boosting classification algorithms using python. Using XGBoost provided provided many benefits for our team to perform further analysis and draw key insights on the data set provided.

Primarily, XGBoost provided significant benefits in execution speed and model performance. While comparing XGBoost to other gradient boosting algorithms, XGBoost vastly outperformed other options. In addition, its model performance with structured & tabular datasets on classification and predictive modelling problems was excellent.

Parameter Tuning



```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.3, max_delta_step=0,
       max_depth=5, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=1, nthread=None, objective='binary:logistic', random_state=0,
       reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=1, silent=True,
       subsample=1)
Accuracy: 91.88%
```

With an initial test of AUC of 0.89 and accuracy of ~90%, the team decided to tune the parameters of the XGBoosting Algorithm in order to improve performance. The learning rate was experimented with until 0.3 was found to improve performance marginally (AUC 0.90 and an increase of accuracy by about ~1%). Increasing the learning rate past 0.6 decreased performance. The subsample and colsample_bylevel were shown to only decrease performance if they were not at their default number of 1. Changing the max_depth to 5 and min_child_weight improved its performance, bringing the accuracy up to 91.88% and an AUC of 0.91. The ROC curve for this model can be found at Appendix C.

Results:

Accuracy = 91.88%

AUC = 0.91

ii) *Random Forest*

The next method we used was implementing a Random Forest classifier to classify instances and to gain a greater understanding of the relationships between the features. This model performed similarly well to the XGBooster. This is likely due to the fact that the Random Forest Classifier does an excellent job of re-sampling the data and understanding which features create the most information gain.

Results:

Accuracy = 90.35%

AUC = 0.90

ii) Logistic Regression

We attempted to build a logistic regression so that we could classify instances using class probability estimation. The logistic regression struggles to predict all instances correctly as seen by the 75% accuracy metric but does have some predictive power. Analyzing the ROC curve, this is evident through comparing the tradeoffs between the *True Positive* rate and *False Positive* rate. The model is not as powerful as the XGBoost or Random Forest model, however it adds some interpretability to the insights drawn in other models.

Overall, logistic regression provided valuable analysis and measurements for further detection and prediction of fraudulent behaviour but does not provide the same predictive power as some of the other classification models listed above.

Results:
Accuracy: 75.17%
AUC = 0.74

Clustering
For the purpose of this report, our team decided to use Clustering as a tool to help measure, detect, and predict the count of potentially fraudulent clicks. Through clustering by total IP Count in the data set, our team attempted to detect a subset of customers that are outliers. It was not informative to add other features to this clustering model as they did not help further separate the potentially fraudulent traffic from organic traffic. The model utilized the K-Means method to organize the data into 3 clusters. This detection helped us address whether the outliers were based from a certain server, and whether the fraudulent clicks were possibly automated & centralized from a certain server. This model should be leveraged over time to flag IP addresses that fall outside of the normal activity range in the model for further inspection and possibly blacklisting for Talking Data services.

Through clustering, we were able to identify that the IP addresses that fall outside of the first cluster (See Appendix D, the blue cluster in the graphic illustration) have a disproportionate amount of unique visits. This discovery provides a starting point to further investigate suspicious fraudulent behaviour. Due to the nature of the encoding of the IP addresses in the dataset provided we can not be sure where these clicks originated from, thus further inspection of the decoded IP addresses would add to the depth of this analysis. We would not want to flag large public hotspots as being fraudulent as they may represent a large number of unique users sharing the same IP address.

Time Series

We primarily used this data science technique for further pattern detection. Pattern detection proved to be an extremely useful insight as it helped our team to identify & highlight fraudulent traffic, as spikes in traffic during non-peak hours can be further investigated. By leveraging past data for any given day of the week and hour of that day we can make a prediction as to the amount of clicks we should expect to see and the conversion ratio associated with these clicks. (See Appendix D) Any major deviations from this average should be investigated as a possible influx fraudulent activity. This model struggles to separate organic spikes in traffic, from a new advertising campaign or organic growth of Talking Data's consumer base, from fraudulent activity but provides a strong base rate for traffic and conversion ratio.

Our time series analysis was divided in Hourly Click Frequency and Hourly Conversion Ratio. These metrics allowed us to monitor the cyclical nature of the traffic that Talking Data experiences and highlight any periods of time where the Hourly Conversion Ratio was lagging behind its average. (See Appendix E) These periods of time should be examined further to determine the origin of the traffic as they represent an unexpected drop in the conversion ratio which may indicate fraudulent activities.

Results

Model & Technique Used	**Result**

XGBoost	Accuracy = 91.88%
	AUC = 0.91

Random Forest	Accuracy = 90.35%
	AUC = 0.90

Logistic Regression	Accuracy = 75.17%
	AUC = 0.74

Clustering	Identification of subset of outlier customers Identification of outlier IP addresses
Time Series	Pattern detection for fraudulent traffic during non-peak hours
Other	Aggregate statistics to do surface level analysis (histograms, averages, etc.)

Observing and analyzing our results at a surface level, it can be seen that we achieved adequate results through accuracy and AUC. However, these results do not take into account various factors that we believe could be improved to further fine tune results for even greater and deeper insights.

Challenges

As a team, we faced various challenges when completing this project. Some of these challenges are highlighted below:

Computing Power

The data set provided by the Kaggle competition proved to be a hurdle for our team. The desired training set was over 7.3 GB of data. This exceeded the RAM capabilities for the computers used by this team. Additionally, when trying to load the datasets into pandas, it would often get "stuck" reading the data from the hard drive and then raise a memoryerror when the program was eventually cancelled. We hypothesized this might have been due to the artificial cap Windows puts on virtual memory. As a way to deal with this problem working with the training sample provided and adding another 70 000 fraudulent clicks created models that would perform well in the actual competition.

Model Selection

Selecting the right classifier was difficult for us, as we had limited knowledge of best practices for these types of algorithms. We ended up combining our in class knowledge with looking at some Kaggle kernels. We learned in class how powerful the random forest classifier could be, and learned about XGBoost from other Kaggle submissions. Ultimately, the XGBoost was a much more efficient and better performing model for our submission.

Class Imbalance

The class predictions were incredibly imbalanced, which made building a model where accuracy would be meaningful difficult. The team dealt with this problem by taking a large sample of fraudulent clicks from the large training set provided by TalkingData.

Technical Appendices

Appendix A: Loading Kaggle Training Set Where is_attributed = 1 (fraudulent click)

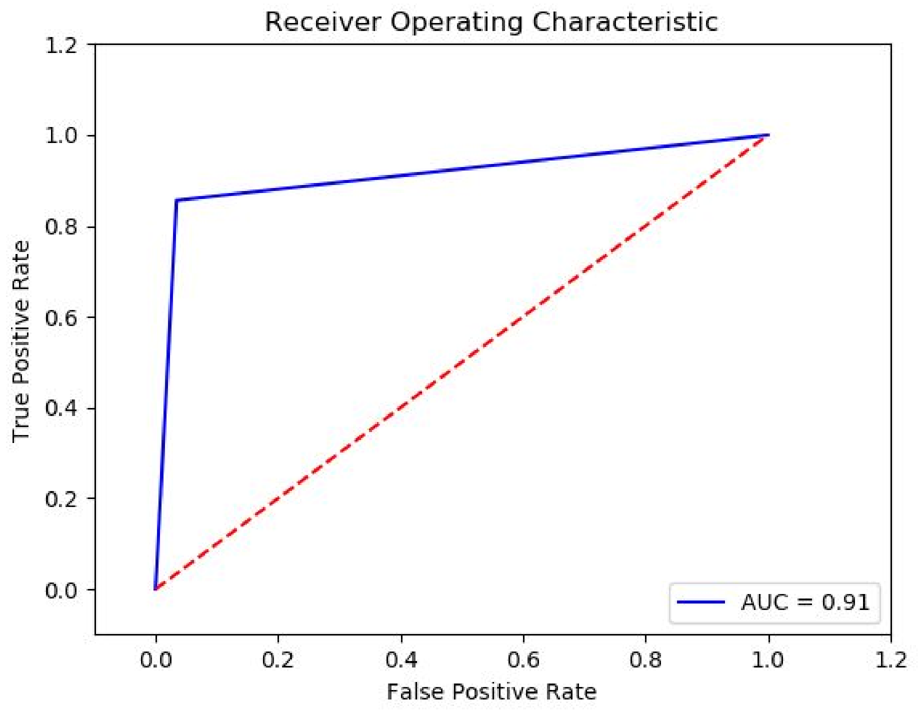
```
import pandas as pd
import numpy as np

# some parameters
#
names=['ip','app','device','os','channel','channel','click_time','attributed_ti
me','is_attributed']
# skiprows = 1,
# dtype={'ip':np.complex128,'app':np.complex128, 'device':np.complex128, 'os':
np.complex128, \ 'attributed_time':np.complex128, 'is_attributed':np.int16}
csv_chunks = pd.read_csv('Train.csv',chunksize = 100000, nrows = 30000000)

# concatenate chunks back into one dataframe
dataset = pd.concat(chunk for chunk in csv_chunks)
data_predicted = dataset.loc[dataset['is_attributed'] == 1]

# Save to excel
writer = pd.ExcelWriter('output.xlsx')
data_predicted.to_excel(writer,'Sheet1')
writer.save()
```

Appendix B: Running the XGBClassifier Model

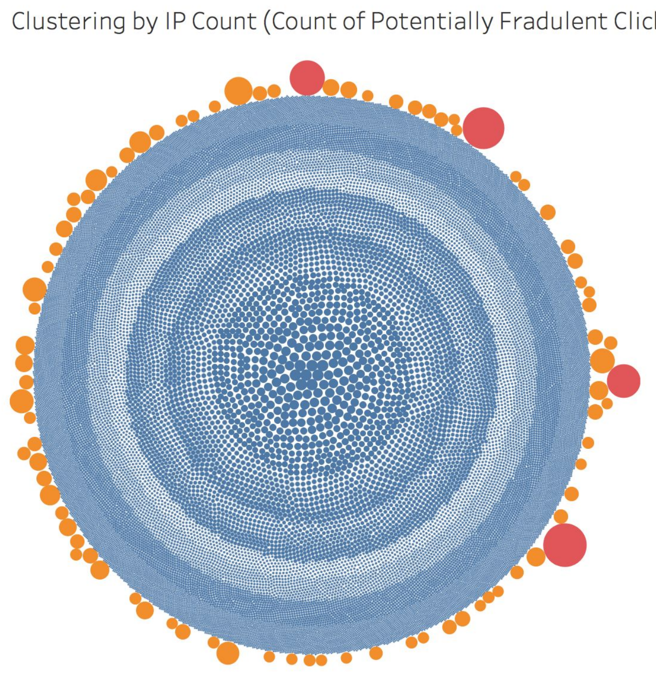
```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc
import matplotlib.pyplot as plt

dataset = loadtxt('train_samplev2.csv', delimiter=",",  usecols=range(6))

# Open columns
X = dataset[:,0:5]
Y = dataset[:,5]
```

```
seed = 7
test_size = 0.80

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)

# fit model no training data
model = XGBClassifier(max_depth = 5, min_child_weight=1, gamma= 0, subsample =
1, colsample_bytree = 1, learning_rate= 0.3, seed = 1 )
model.fit(X_train, y_train)
print(model)

# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]

size_actual = int(174723*test_size)

actual = dataset[:size_actual,5]

accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

#plot ROC
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test,
y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

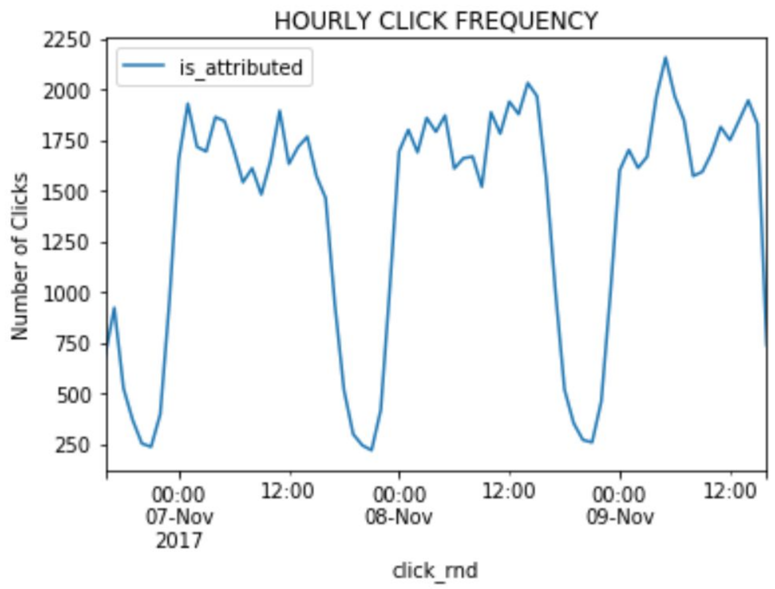
Appendix C: ROC Curve for XGBClassifier



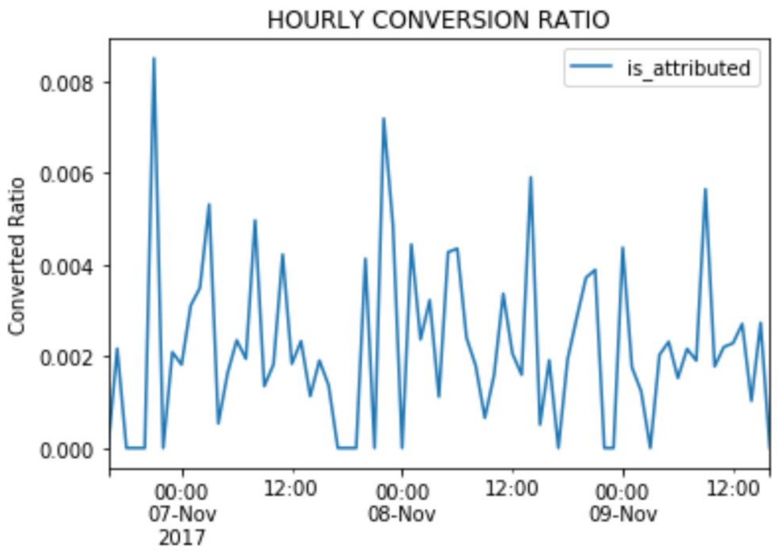
Appendix D: Clustering by Visit Frequency and IP



Appendix E: Hourly Click Frequency

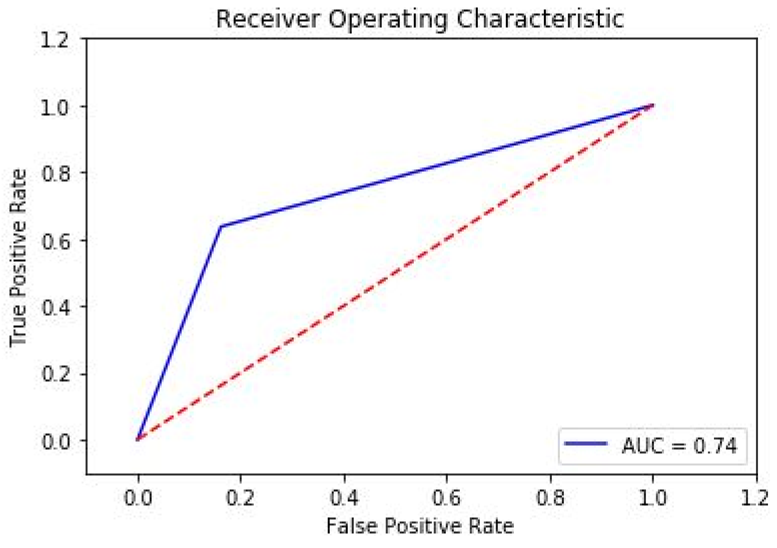


Appendix F: Hourly Conversion Ratio



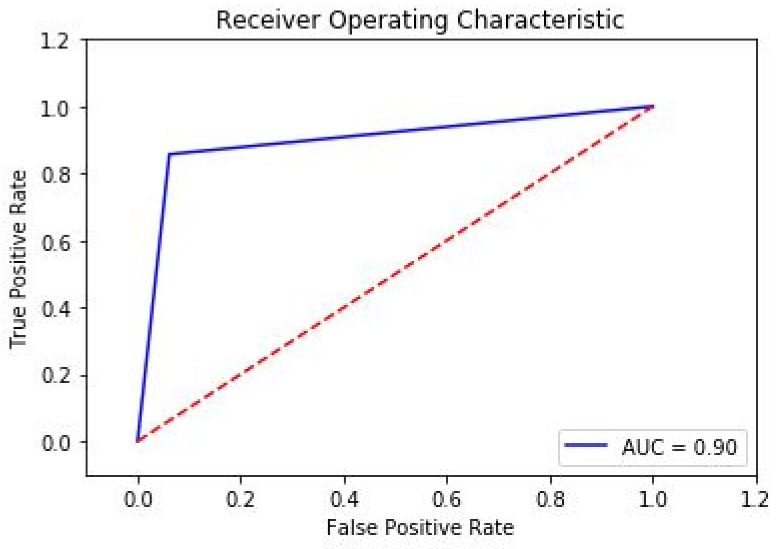
Appendix G: Logistic Regression

```
Accuracy: 75.17%
```



Appendix H: Random Forest Classifier

```
Accuracy: 90.35%
```



Appendix I: Code for Random Forest Classifier

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
%matplotlib inline
```

```python
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_curve, auc

dataset = loadtxt('train_samplev2.csv', delimiter=",",  usecols=range(6))


# Open columns
X = dataset[:,0:5]
Y = dataset[:,5]


seed = 7
test_size = 0.80

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)

# fit model no training data
model = RandomForestClassifier()
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)
predictions = [round(value) for value in y_pred]

size_actual = int(174723*test_size)

actual = dataset[:size_actual,5]

accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

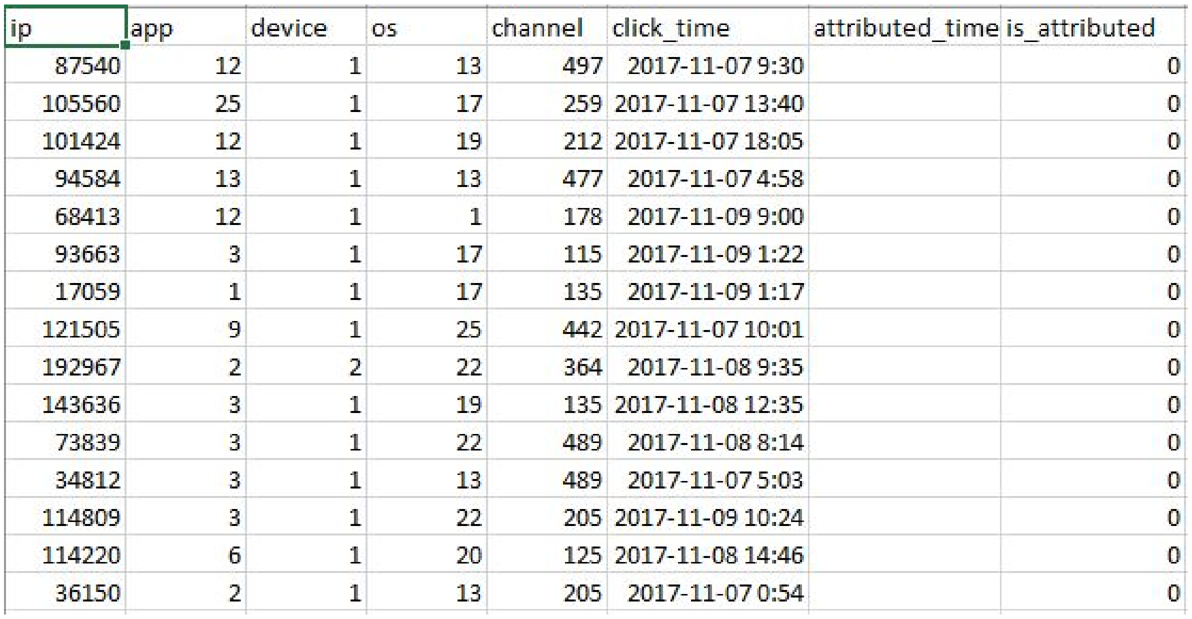
```
#plot ROC
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

Appendix J: Code for Logistic Regression

```

dataset = loadtxt('train_samplev2.csv', delimiter=",",  usecols=range(6))


# Open columns
X = dataset[:,0:5]
Y = dataset[:,5]


seed = 7
test_size = 0.80

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_size,
random_state=seed)

# fit model no training data
model = LogisiticRegression()
model.fit(X_train, y_train)

# make predictions for test data
y_pred = model.predict(X_test)
```

```
predictions = [round(value) for value in y_pred]

size_actual = int(174723*test_size)

actual = dataset[:size_actual,5]

accuracy = accuracy_score(y_test, predictions)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

#plot ROC
false_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(false_positive_rate, true_positive_rate)
plt.title('Receiver Operating Characteristic')
plt.plot(false_positive_rate, true_positive_rate, 'b',
label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.plot([0,1],[0,1],'r--')
plt.xlim([-0.1,1.2])
plt.ylim([-0.1,1.2])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

Appendix K: Data Labels



ip	app	device	os	channel	click_time	attributed_time	is_attributed
87540	12	1	13	497	2017-11-07 9:30		0
105560	25	1	17	259	2017-11-07 13:40		0
101424	12	1	19	212	2017-11-07 18:05		0
94584	13	1	13	477	2017-11-07 4:58		0
68413	12	1	1	178	2017-11-09 9:00		0
93663	3	1	17	115	2017-11-09 1:22		0
17059	1	1	17	135	2017-11-09 1:17		0
121505	9	1	25	442	2017-11-07 10:01		0
192967	2	2	22	364	2017-11-08 9:35		0
143636	3	1	19	135	2017-11-08 12:35		0
73839	3	1	22	489	2017-11-08 8:14		0
34812	3	1	13	489	2017-11-07 5:03		0
114809	3	1	22	205	2017-11-09 10:24		0
114220	6	1	20	125	2017-11-08 14:46		0
36150	2	1	13	205	2017-11-07 0:54		0