**IE 583 Project**

**April 27th 2018**

**Group - P7**

**Aditya-Rao Madahavaram**

**Deepak-George Thomas**

**Nishant Kumar**

**Table of Contents:**

* **Introduction**
* **Data Analysis**
* **Output Analysis**
* **Future Study**
* **Challenges**
* **Conclusion**
* **Documents Attached**

**Introduction:**

This project is about Fraud Analysis as the advertising companies need to control the fraud risk. Being in marketing world, it would be challenging to avoid the consequences of the fraud related loss. We got our data from Kaggle.com. Technology advancement also bring lots of challenges such as online malware, phishing sites and fraud advertising of mobile apps.

China possesses the world’s largest mobile market thereby making it a potential target for fraudsters who use ad-channels to continuously click on certain ads. These organizations causes huge losses to app organizations due to their fraudulent activities as they simply drive up ad click numbers without leading to app downloads or the desired result of the ad.

China’s largest independent big data service platform, TalkingData handles 3 billion clicks per day, of which 90% are potentially fraudulent. By measuring the journey of user’s click process across their portfolio, the firm flags the IP addresses which produce lots of clicks, but never end up installing. They have developed a blacklist for IP address and device, based on this approach. With more than 200 million clicks generated over 4 days, the competition in kaggle wants it’s users to develop a predictive algorithm on the downloadability of the app after clicking the app advertisement.

**Data Analysis:**

We first downloaded the data from Kaggle site and due to extremely large size of the data files we had to split the data into chunks so that our computer could handle it. We ran into memory issues very frequently while running the code so we changed the R code into three parts. We also ran our code in AWS( Amazon Web Services) server.

**Part 1.** Ran the R code (**Kaggle\_Rcode1.R**, **Kaggle\_Rcode1\_function**) to load the data and read the train.csv file in chunks(1 million in one chunk) and create a model and fit the test data to create the output file with click\_Id and Is\_attributed. Output files are named as chunk1.txt, chunk2.txt,..chunk20.txt. We created 20 output files with 18790469 rows in each chunk. Due to computing limitation, we trained the model reading the 20 million records from train.csv file.

**Part 2.** Ran the R code (**Kaggle\_Rcode2.R**) to combine all 20 chunk.txt files in one file. After combining the 20 files, every click id has 20 values in a row. We average out the records selection. If the is\_attributed has more than 10 ones then it is set to 1, otherwise 0. Created a new file to and saved the file to local hard drive to reduce the stress on the RAM.

**Part 3.** This R code (**Kaggle\_Rcode3\_knit.Rmd**) created to predict the values and make sense out of the results.

Also, in order to improve the speed while understanding different aspects of the data, the dataset was run on different computers using different parts of the data.Therefore another data division technique employed on a separate computer was as follows -

The entire data was divided into 16 chunks, with each consisting of 12 million instances. This was done to better process the data and overcome the computational difficulties involved in drawing the entire dataset together into R studio. From each chunk, 5.5 million majority instances were extracted. This helped maintain the uniformity of the dataset as opposed to taking data from a few specific chunks and ignoring the rest. The minority instances from all the chunks were extracted, since the number of minority instances are extremely small as compared to majority instances. This gave a total of 88 million majority instances and 22,000 minority instances. The R code details is indicated in the documents attached section.

**Classification Techniques**

Random Forest and Naive Bayes were the techniques used for classification. The rationale behind using Random Forest was its fast computation time and its better prediction performance. For large datasets, with the absence of powerful computational devices, cross validation is highly time consuming in R. Random Forest gives us the ability to obtain quick results and the package ‘randomForest’ in R uses out of bag error as performance measure to obtain the true error. Also, we used Naive Bayes as another classification technique to evaluate the model performance. We found that the attributes in the dataset were independent of each other. This fits perfectly with the assumptions of Naive Bayes. Although, after we performed feature construction, the independence between attributes would have reduced. However, this classification technique was still used as it saved time due to the absence of need of significant tuning.Furthermore, we didn’t use other conventional techniques like knn or decision trees. This was because the former doesn’t work very well with categorical variables (many of the variables in the dataset were converted into factor variables) and the latter was already incorporated while building random forest.

**Feature Engineering**

Features were derived from the training and the test data in order to improve the performance of the model. The features that were constructed are as follows -

1. Hour
2. Morning - created another feature for any clicks between 6 - 11
3. Afternoon - created another feature for any clicks between 12 - 17.
4. Evening - created another feature for any clicks between 18 - 23.
5. Night - created another feature for any clicks between 0 - 5.
6. App & device - The values of app and device were placed together as a separate instance.
7. App & operating system - The values of app and operating system were placed together as a separate instance.
8. App & channel - The values of app and channel were placed together as a separate instance.
9. Device & operating system - The values of device and operating system were placed together as a separate instance.
10. Device & channel - The values of device and channel were placed together as a separate instance.
11. Operating system & channel - - The values of operating system and channel were placed together as a separate instance

We did tune up of the parameters as well for better accuracy. Then we selected the best model based on the accuracy. We have not considered the overall accuracy as a final decision maker to select the model. This was because overall accuracy in a highly imbalanced dataset can be very deceptive. Moreover, they fail to consider the misclassification costs (*Garcia et. al, 2009).* We looked for the predictor values as well as other performance metrics such as balanced accuracy and kappa.

**Feature Resampling**

In order to reduce the inherent imbalance in the data, we experimented with various resampling techniques including downsampling and SMOTE. Upsampling was not performed since it was not possible to train the number of instances generated with the current computational capability.

* **Non implementation of constructed features and no resampling**

In this case no constructed feature was used and no resampling was performed. 1.8 majority instances and 1.2 minority instances were randomly sorted and the ratio between majority to minority instances was 60:40.

is\_attributed

0 : 180000

1 : 84000

The confusion matrix obtained is as follows -

|  |  |  |
| --- | --- | --- |
|  | Reference |  |
| Prediction | 0 | 1 |
| 0 | 52162 | 4677 |
| 1 | 1838 | 31323 |

**The overall accuracy obtained was 92.76%, kappa = 0.85 and balanced accuracy = 91.8%.**

* **Downsampling**

|  |  |  |
| --- | --- | --- |
|  | Reference |  |
| Prediction | 0 | 1 |
| 0 | 3704272 | 1631 |
| 1 | 195728 | 12074 |

**This method yielded an overall accuracy of 94.96%, kappa of 10.31% and a balanced accuracy of 91.54%.**

* **SMOTE**

We used SMOTE and changed the perc.over, perc.under values and we picked perc.over = 1000, perc.under = 100 because of better accuracy. Below are results for some of the SMOTE tuning we have done.

With- perc.over = 400, perc.under = 100

Naive Bayes Random Forest

Accuracy : 0.6801762 0.9838473

With- perc.over = 800, perc.under = 100

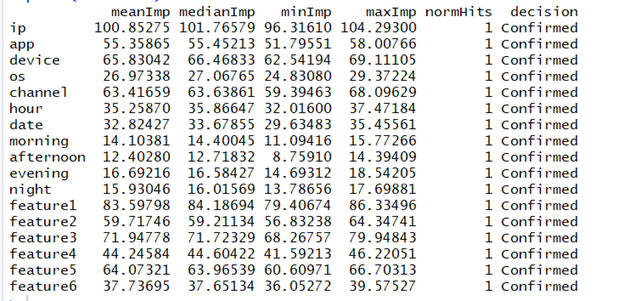
Naive Bayes Random Forest

Accuracy : 0.6229202 0.9869383

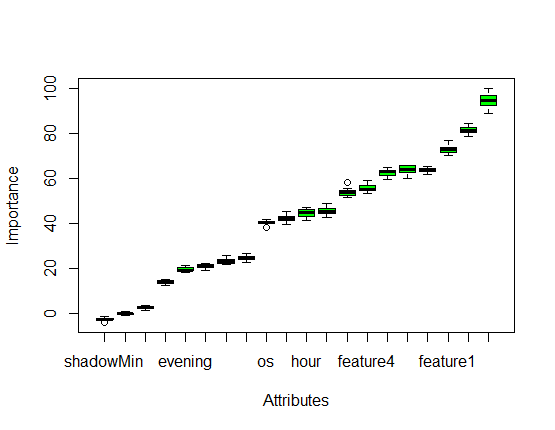
With- perc.over = 1000, perc.under = 100

Naive Bayes Random Forest

Accuracy : 0.6689325 0.9903072





a

Using Boruta we found that all the features are significant.

Once we got the best model, we fit into the test.csv file and got the output files with predicted output values for is\_attributed . Due to the computational limitation we created the predicted output files for 20 chunks. Which is 20 times 18790469 rows.

**Output Analysis:**

|  |
| --- |
| Logistic Regression : The predictions values are 0’s and 1’s, it seems logical to fit a logistic model to the data as shown below.  Call:  ## glm(formula = pred ~ ip + app + device + os + channel + click\_time +  ## feature1 + feature2 + feature3 + feature4 + feature5 + feature6,  ## data = test.pred)  ##  ## Deviance Residuals:  ## Min 1Q Median 3Q Max  ## -4.8709 -0.0385 -0.0178 0.0067 1.5022  ##  ## Coefficients:  ## Estimate Std. Error t value Pr(>|t|)  ## (Intercept) -6.608e-02 1.382e-04 -478.294 < 2e-16 \*\*\*  ## ip -1.364e-10 8.787e-10 -0.155 0.877  ## app 1.030e-02 1.701e-05 605.951 < 2e-16 \*\*\*  ## device 1.038e-03 9.187e-05 11.297 < 2e-16 \*\*\*  ## os -6.088e-05 1.398e-05 -4.356 1.33e-05 \*\*\*  ## channel 6.960e-05 2.655e-07 262.169 < 2e-16 \*\*\*  ## click\_time 9.877e-04 8.643e-06 114.281 < 2e-16 \*\*\*  ## feature1 -1.071e-06 8.943e-09 -119.784 < 2e-16 \*\*\*  ## feature2 -2.281e-06 4.093e-08 -55.720 < 2e-16 \*\*\*  ## feature3 -5.916e-06 1.675e-08 -353.113 < 2e-16 \*\*\*  ## feature4 -7.167e-07 9.641e-09 -74.336 < 2e-16 \*\*\*  ## feature5 5.920e-08 9.178e-08 0.645 0.519  ## feature6 6.001e-07 1.396e-08 42.972 < 2e-16 \*\*\*  ## ---  ## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  ##  ## (Dispersion parameter for gaussian family taken to be 0.01973606)  ##  ## Null deviance: 434334 on 18790468 degrees of freedom  ## Residual deviance: 370850 on 18790456 degrees of freedom  ## AIC: -20433315 |

We clearly see 97% of the predicted variables has ‘0’ and 3 % has ‘1’. Also from the results we noticed IP and feature 5 are not significant, since the probability value obtained is more than 0.1. We focused on prediction values equal to 1 with goal of finding interesting interaction between app, device, os and channel. There are few app, device, operating system and channel they have highest number of clicks compare to others. We created the features and observed it’s behavior.

|  |  |
| --- | --- |
| Feature # | Observations |
| feature1 (combination of app and device) | the app 29 and device 1 has the highest frequency of clicks |
| feature2(combination of app and os) | the app 19 and os 24 has the highest frequency of clicks |
| feature3(combination of app and channel) | the app 19 and channel 347 has the highest frequency of clicks |
| feature4(combination of device and os) | the device 0 and os 24 has the highest frequency of clicks |
| feature5(combination of device and channel) | the device 0 and channel 347 has the highest frequency of clicks |
| feature6(combination of os and channel) | the os 33 and os 347 has the highest frequency of clicks |

In the above results, we only considered the two labels with the highest frequency for each of the features.

Following are the combination of the features that gives the interesting results

|  |
| --- |
| interesting0 = subset(test.pred.one, app == '19' & device == '0' & (os == '24' | os == '33') & channel == '347')  The app ‘19’, device ‘0’, os = “24” or “33” and channel = 347 has total **44332** clicks  which is approximately 10% of the total predicted clicks.  interesting1 = subset(test.pred.one, (app == '19' | app == '29') & (device == '0' | device == '1') & (os == '0' | os == '24') & (channel == '213' | channel == '347'))  The app ‘19’ or ‘29’, device ‘0’ or ‘1’, os = ‘0’ or “24” and channel = ‘213’ or ‘347’ has total **115143** clicks. Which is approximately 25 % of the total predicted clicks. |

**Future Study:**

Note that the results of this analysis is based on analyzing the first 20 million observations of the training data set. A follow-up would be to consider all the observations in the training data set. The main reason why we could not include all the observations in the training data set was due to computing limitations.

**Challenges:** Large data file, Computing limitation, insufficient time, lack of understanding for the data points,

**Conclusion:**

Due to the huge dataset and lack of computing power, so we took a different approach, we divided data into chunks and each chunk contains 1 Million observations. We then created features like hour, operating system, combination of channels and OS, device and we created 6 features.

Once we created the features then next step is to do SMOTE because the data set is unbalanced. Then we created training dataset from SMOTE and test data set then we did variable selection by using BORUTA algorithm. We then took the results from BORUTA and selected the variables with most important ones in terms of predicting the variables of interest one, here it is is\_attributed. Then we proceed to fit the model, we used Naïve Bayes and Random Forest Classifications to select our best models. For tuning we considered number of trees and the number of variables. With the combination of these we selected features with performance metrics other than accuracy and compared for both classifiers and we selected data with the most accuracy. This process was then repeated for each chunk and then we took the average of the predictor to get to the results.

**Documents attached:**

1. Kaggle\_Rcode1.R (part 1)
2. Kaggle\_Rcode1\_Function.R (part-1)
3. Kaggle\_Rcode2.R (part-2)
4. Kaggle\_Rcode3\_knit.Rmd (part-3)
5. Kaggle\_Rcode3\_knit.docx (code with output results in it)
6. GroupP7\_Project\_Report.docx. (Final Report)
7. GroupP7\_Presentation.pptx (Presentation)

Note : We have been trying several different methods to work on this project due to very large data set file. Finally we had two approach to get the output from the part 1. Here is the R code used for the second approach.

Supplementary R Code:

6. Supp\_feature.R

7 Supp\_training.R

8. Supp\_tuning.R