CS 6375 – MACHINE LEARNING SEMESTER PROJECT REPORT

OUTBRAIN CLICK PREDICTION

CONTRIBUTERS:

1	Adit Desai	aud160030
2	Hiteshi Dehal	hxd151630
3	Gaurav Dhingra	gxd150230
4	Sagar Kansara	suk150130
5	Sunny Anand	sxa151231

Contents

Introduction

Outbrain is the web's leading content discovery platform, pairs relevant content with curious readers in about 250 billion personalized recommendations every month across many thousands of sites. In one of the challenges on Kaggle, Outbrain asks to predict which pieces of content its global base of users are likely to click on. We picked their dataset as our project.

Dataset Details:

The dataset contains a sample of users' page views and clicks, as observed on multiple publisher sites in the United States between 14-June-2016 and 28-June-2016. The dataset contains numerous sets of content recommendations served to a specific user in a specific context. Each context (i.e. a set of recommendations) is given a display_id. One display_id can have a set of ad_id. In each such set, the user has clicked on at least one recommendation. Each user in the dataset is represented by a unique id (uuid). A person can view a document (document_id), which is simply a web page with content (e.g. a news article). On each document, a set of ads (ad_id) are displayed. Each ad belongs to a campaign (campaign_id) run by an advertiser (advertiser_id). We are provided metadata about the document, such as which entities are mentioned, a taxonomy of categories, the topics mentioned, and the publisher.

FileDescriptions

Dataset has following 9 csv files:

1. **page_views.csv** is a the log of users visiting documents containing following fields:
    - uuid
    - document_id
    - timestamp (ms since 1970-01-01 - 1465876799998)
    - platform (desktop = 1, mobile = 2, tablet =3)
    - geo_location (country>state>DMA)
    - traffic_source (internal = 1, search = 2, social = 3)

2. **clicks_train.csv** is the training set, showing which of a set of ads was clicked. Each display_id has only one clicked ad. It has following fields:
    - display_id
    - ad_id
    - clicked (1 if clicked, 0 otherwise)

3. **clicks_test.csv** is the same as clicks_train.csv, except it does not have the clicked ad. This will be used for prediction.

4. **events.csv** provides information on the display_id context. It covers both the train and test set with the following fields:
   - display_id
   - uuid
   - document_id
   - timestamp
   - platform
   - geo_location

5. **promoted_content.csv** provides details on the ads with the following fields:
   - ad_id
   - document_id
   - campaign_id
   - advertiser_id

6. **documents_meta.csv** provides details on the documents with the following fields:
   - document_id
   - source_id (the part of the site on which the document is displayed, e.g. edition.cnn.com)
   - publisher_id
   - publish_time

7. **documents_topics.csv** gives the confidence that the given topic was referred to in the document.
8. **documents_entities.csv** gives the confidence that the given entity was referred to in the document.
9. **documents_categories.csv** gives the confidence that the given category was referred to in the document.

7^{th}, 8^{th} and 9th file provides information about the content in a document, as well as Outbrain's confidence that it was referred, in each respective relationship.

Below are the details of all the files with respective fields:

clicks_train

display_id	ad_id	clicked

To predict

clicks_test

display_id	ad_id

documents_meta

document_id	source_id	publisher_id	publish_time

documents_categories

document_id	category_id	confidence level

document_entities

document_id	entity_id	confidence level

document_topics

document_id	topic_id	confidence level

promoted_content (Set)

ad_id	document_id	campaign_id	advertiser_id

events

display_id	uuid	document_id	timestamp	platform	geo_location

page_views

uuid	document_id	timestamp	platform	geo_location	trafffic_source

Reading & Loading the data into R workspace

The Outbrain dataset at Kaggle has large amount of data in few of the tables such as page view, so as per their recommendation, we can make use of the page_view_sample dataset. To make the data overall consistent as per page_views_sample data we analyzed the data to understand the relationship of data sitting in various tables and fetch correspondingly significant data as per page_views_sample dataset.

All the tables in outbrain dataset are with header lines and comma separated. We use the in-built read.table() function to initially read the data and load into the R workspace.

Pre-processing the data

Before we started with model creation we wanted to make sure the data is suitable for applying any machine learning classification technique. For this we analyzed each of the subset dataset we had loaded into the R workspace for the following:

- Check for NA (Not Available Values)
- Check for Inf (Infinite Numbers)
- Check for NaN(Not a Number)
- Check for NULL(not NULL values)

We identified none of the tables suffered from either of these issues.

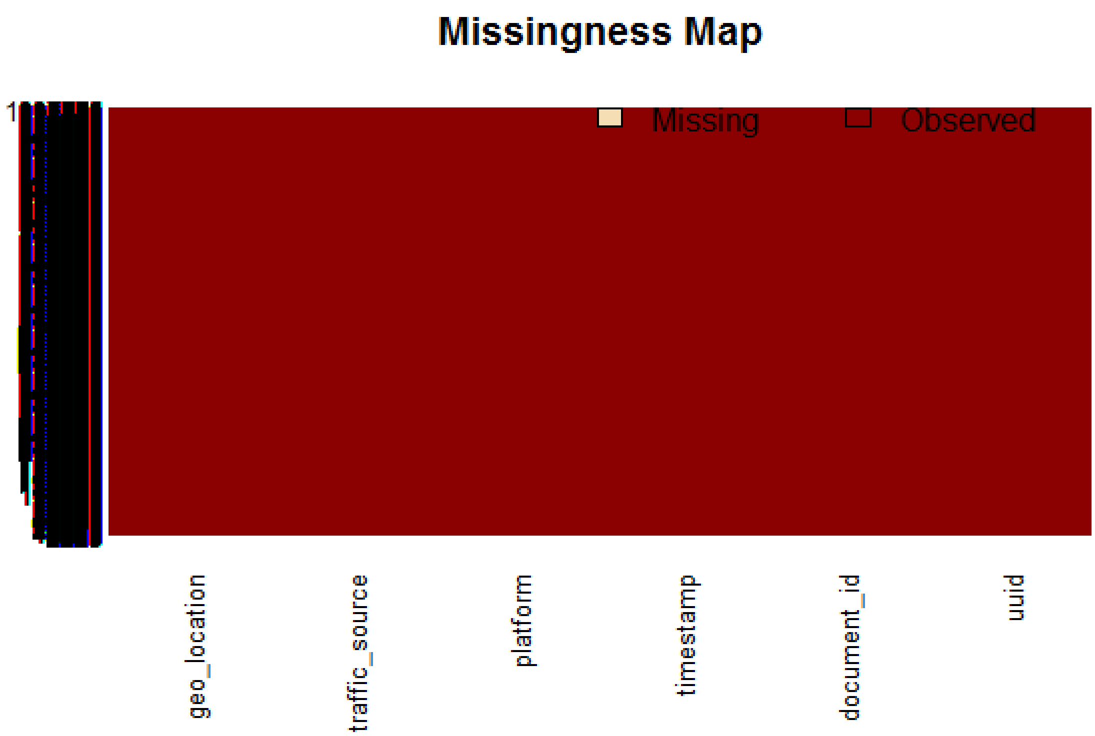
We then started to check for outliers in each table and their attributes. We did this using the boxplot graph and checked if there are any values outside the 1st or 3rd quartile to be labeled as outliers. During this analysis we realized that there are variables which are **categorical** and have to be transformed. The result of boxplot analysis for each attribute of each table is logged into the log file below which carries complete analysis of outliers. The data with respect to attributes is not skewed in favor of any single attribute as per the analysis.

Boxplot.stats log:

		$stats	$n	$conf	$out
1. clicks_test					
i.	display_id	16874594 16923615 16971915 17019839 17066994	1000000	16971763 16972067	integer(0)
ii.	ad_id	1 94994 159665 226253 368055	1000000	159457.6 159872.4	integer(0)
2. events					
i.	display_id	1.0 250000.5 500000.5 750000.5 1000000.0	1000000	499210.5 500790.5	integer(0)
ii.	uuid	Boxplot not applicable to factors			
iii.	document_id	Document_id are independent			
iv.	timestamp	61 25559393 38634947 50183344 61443773	1000000	38596041 38673853	integer(0)
v.	platform	Boxplot not applicable to factors			
vi.	geo_location	Boxplot not applicable to factors			
3. clicks_train					
i.	display_id	1 48824 97771 146800 195916	1000000	97616.2 97925.8	integer(0)
ii.	ad_id	35 103756 171554 235798 353469	1000000	171345.4 171762.6	integer(0)
iii.	clicked	Boxplot not applicable to factors			
4. promoted_content					
i.	ad_id	1.0 140624.5 281770.0 422190.5 573098.0	559583	281175.3 282364.7	integer(0)
ii.	document_id	Here, document_id corresponds to secondary level of linkage between ads displayed on page and the webpage that gets opened after clicking. These outliers are not significant for our analysis.			
iii.	campaign_id	1 7243 17101 27502 35554	559583	17058.21 17143.79	integer(0)
iv.	advertiser_id	2 602 1635 2556 4532	559583	1630.873 1639.127	integer(0)
5. page_views_sample					
i.	uuid	Boxplot not applicable to factors and non-numeric values			
ii.	document_id	120 95029 732651 1759011 1832150	1000000	730021.9 735280.1	integer(0)
iii.	timestamp	54 28712384	100000	43882026	integer(0

		43935089 62296579 86399931	0	43988152	)
iv.	platform	Boxplot not applicable to factors			
v.	geolocation	Boxplot not applicable to factors			
vi.	traffic_source	Boxplot not applicable to factors			
6. document_entities					
i.	document_id	2 758902 1330801 1778416 2999318	1000000	1329190 1332412	integer(0)
ii.	entity_id	Boxplot not applicable to factors and non-numeric values			
iii.	confidence_level	0.001104595 0.266984704 0.351651133 0.597482965 0.996569486	1000000	0.3511289 0.3521733	numeric(0)
7. document_topics					
i.	document_id	2 429991 972093 1486003 2999284	1000000	970424.5 973761.5	integer(0)
ii.	topic_id	0 74 143 231 299	1000000	142.7519 143.2481	integer(0)
iii.	confidence_level	0.00800000 0.01365688 0.02737284 0.05440597 0.11552782	1000000	0.02730845 0.02743722	Few outliers
8. document_categories					
i.	document_id	2 621364 1240200 2022606 2999318	1000000	1237986 1242414	integer(0)
ii.	category_id	1000 1406 1702 1902 2100	1000000	1701.216 1702.784	integer(0)
iii.	confidence_level	0.00100000 0.06575676 0.29463699 0.92000000 1.00000000	1000000	0.2932873 0.2959867	numeric(0)

To cross-verify this whole analysis for NA values we also performed **missingness map test** using the Amelia Package.



The result of this test verified our above approach with respect to data consistency.

Next, we wanted to ensure that the dataset has feasible datatypes so that we can produce correct output when we perform joins on the datasets to get the final table as final table will be used to make the prediction. To accomplish this we had to manually analyze each attribute of the table and ensure that all of them have similar datatypes as these tables share common attributes. We analyzed the datatype of each attribute and the result of this analysis can be found in the log file below.

Datatype log:

1. clicks_test		
i.	> typeof(clicks_test$display_id)	"integer"
ii.	> typeof(clicks_test$ad_id)	"integer"
2. events		
i.	> typeof(events$display_id)	"integer"
ii.	> typeof(events$uuid)	"integer"
iii.	> typeof(events$document_id)	"integer"
iv.	> typeof(events$timestamp)	"integer"
v.	> typeof(events$platform)	"integer"
vi.	> typeof(events$geo_location)	"integer"
3. clicks_train		
i.	> typeof(clicks_train$display_id)	"integer"
ii.	> typeof(clicks_train$ad_id)	"integer"
iii.	> typeof(clicks_train$clicked)	"integer"
4. promoted_content		
i.	> typeof(promoted_content$ad_id)	"integer"
ii.	>typeof(promoted_content$document_id)	"integer"
iii.	> typeof(promoted_content$campaign_id)	"integer"
iv.	> typeof(promoted_content$advertiser_id)	"integer"
5. page_views_sample		
i.	> typeof(page_views_sample$uuid)	"integer"
ii.	> typeof(page_views_sample$document_id)	"integer"
iii.	> typeof(page_views_sample$timestamp)	"integer"
iv.	> typeof(page_views_sample$platform)	"integer"
v.	> typeof(page_views_sample$geo_location)	"integer"
vi.	> typeof(page_views_sample$traffic_source)	"integer"
6. document_entities		
i.	> typeof(document_entities$document_id)	"integer"
ii.	> typeof(document_entities$entity_id)	"integer"
iii.	> typeof(document_entities$confidence_level)	"double"
7. document_topics		
i.	> typeof(document_topics$document_id)	"integer"
ii.	> typeof(document_topics$topic_id)	"integer"
iii.	> typeof(document_topics$confidence_level)	"double"
8. document_categories		
i.	> typeof(document_categories$document_id)	"integer"
ii.	> typeof(document_categories$category_id)	"integer"
iii.	> typeof(document_categories$confidence_level)	"double"

Once the datatypes were verified to be actually correct we focused on the categorical variables which were there in these tables. To handle the categorical variables, we transformed each of them into different categories and used the Caret packages model.matrix() method to convert it into the dataframe. The categorical attributes in page_views_sample dataset were traffic_source and platform. We created dummy attributes and divided these categorical attributes into separate variable with each category has one separate variable. Attribute 'traffic_source' was divided into 3 attributes: 'traffic_source1', 'traffic_source2' and 'traffic_source3'. Attribute 'platform' was divided into 3 attributes: 'platform1', 'platform2' and 'platform3'.

Joining the data

After reading each dataset and creating dummy variables in the page_views_sample dataset the next task is to combine all data into one dataset. We used 'sqldf' and 'tcltk' packages of R to combine different dataset based on the unique key concept. We did this processing to ensure all the relevant data is available to us. We joined the following tables.

- Events
- Pageviews
- Clickstrain
- Promotedcontent

There were several challenges while joining these tables as the amount of data getting generated due to joins led to huge matrices getting generated. To overcome this challenge we broke the relational joins in single joins and projection of each join on the following tables in the following order:

- events_pageviews_Clickstrain_promotedcontent & documents_topics
- documents_entities
- documents_categories

Dimensions of our final table were 18 rows X 56260 columns. Columns included in the final table are:

1. display_id
2. uuid
3. document_id
4. platform1
5. platform2
6. platform3
7. geo_location
8. traffic_source1
9. traffic_source2
10. traffic_source3

11. ad_id
12. topic_id
13. topic_conf_level
14. entity_id
15. entity_confidence
16. category_id
17. category_confidence
18. clicked

The final table was then analyzed for the process of feature selection.

Feature Selection

The final table thus obtained after joining various tables was then analyzed for the process of feature selection. The final table and the data which we had after pre-processing was very large and we needed to find a way to reduce the dimensionality of this data. To begin with we started with a linear regression model to find the **p-value** characteristic of the different attributes in the final table with respect to the response variable. We ran a **linear regression model** with all attributes and analyzed the summary of this linear model.

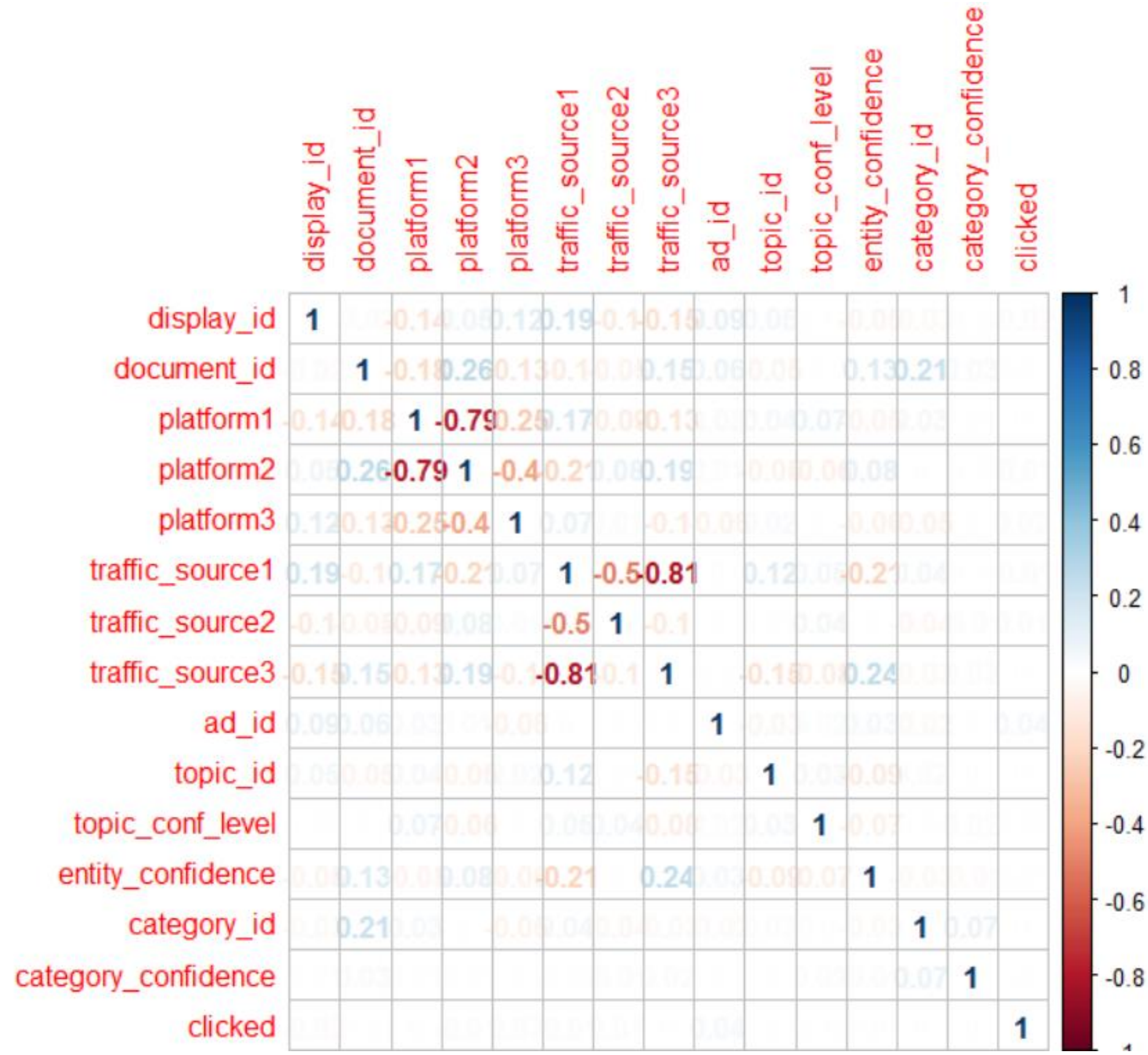
```
Coefficients:          Estimate Std. Error t value Pr(>|t|)
(Intercept)           2.355e-01  1.867e-02  12.612  < 2e-16 ***
display_id           -1.415e-07  3.081e-08  -4.594 4.37e-06 ***
document_id          -1.209e-08  5.288e-09  -2.286   0.0223 *
ad_id                 2.057e-07  2.105e-08   9.770  < 2e-16 ***
topic_id             -5.773e-06  2.007e-05  -0.288   0.7737
category_id           1.034e-05  9.582e-06   1.079   0.2807
platform1            -3.031e-02  6.068e-03  -4.995 5.91e-07 ***
platform2            -2.965e-02  5.811e-03  -5.101 3.38e-07 ***
platform3                    NA         NA      NA       NA
traffic_source1      -1.043e-02  5.389e-03  -1.936   0.0529 .
traffic_source2       3.824e-03  8.638e-03   0.443   0.6580
traffic_source3              NA         NA      NA       NA
topic_conf_level      3.996e-02  2.900e-02   1.378   0.1683
entity_confidence    -1.248e-02  7.039e-03  -1.773   0.0762 .
category_confidence -1.619e-06  4.210e-03   0.000   0.9997
```

The variables that were found to be significant as per p-values are display_id, document_id, ad_id, platform1 and platform2.

To verify our findings we applied 2 basic approaches of validation for feature selection as below:

Correlation pairs:

This gave us the top 5 attributes which were very close to the calculated p-value in the linear regression model.

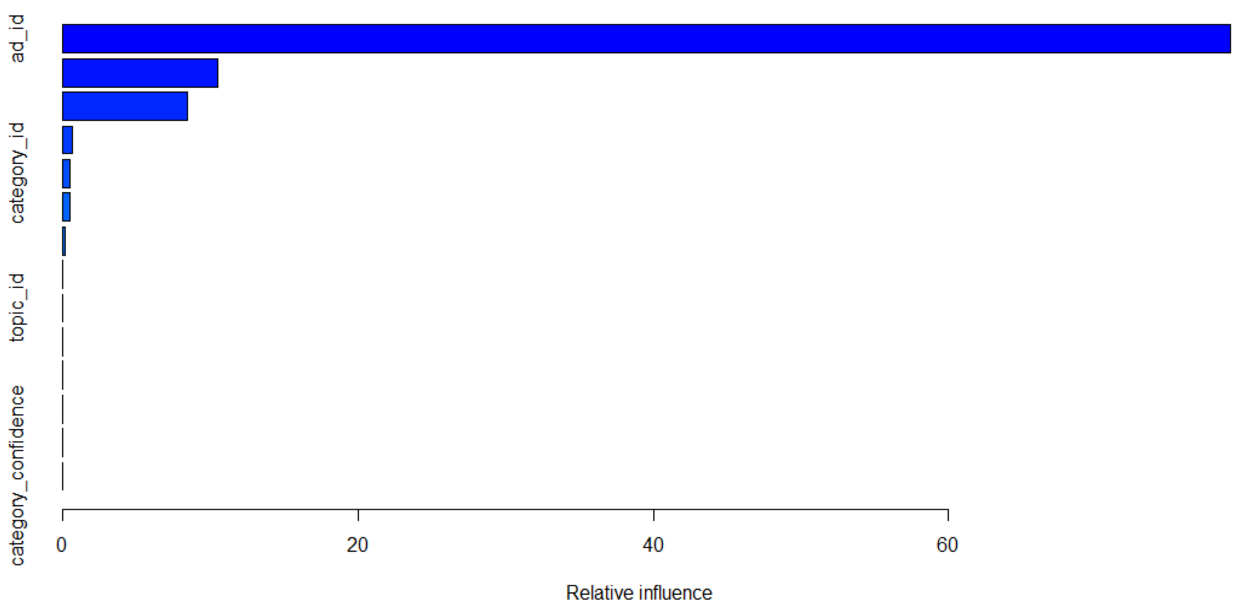


Rank Features by Importance:

Using the gradient boosting method we analyzed that the features of most importance are exactly the ones identified by p-value.

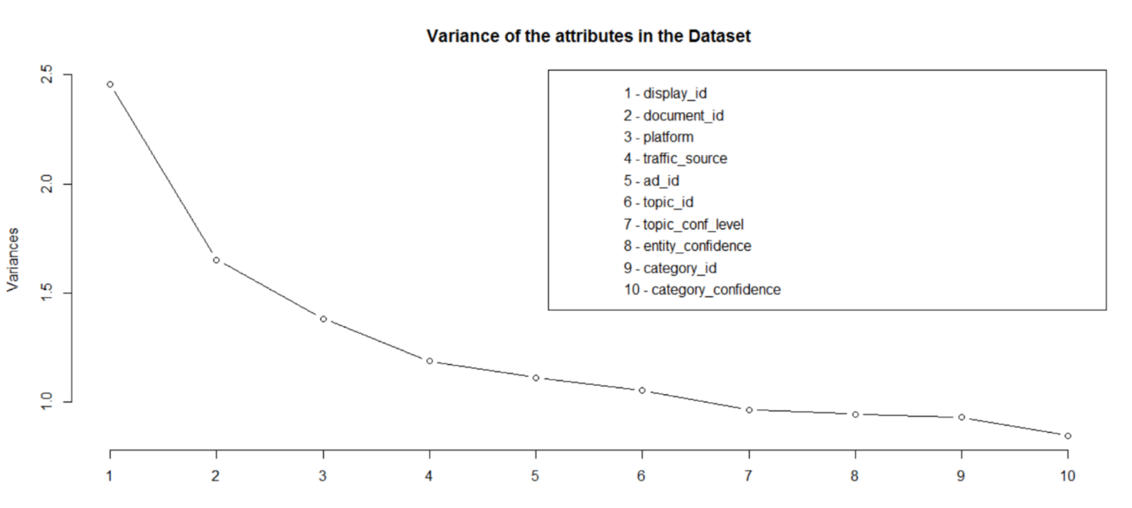
Following is the summary and plot obtained from the model.

```
                                     var      rel.inf
ad_id                              ad_id 77.37606011
display_id                    display_id 10.84519696
document_id                  document_id  8.86397514
traffic_source2          traffic_source2  1.82369988
category_id                  category_id  0.75235386
platform2                      platform2  0.26720532
platform3                      platform3  0.04101629
entity_confidence      entity_confidence  0.03049243
topic_id                        topic_id  0.00000000
platform1                      platform1  0.00000000
traffic_source1          traffic_source1  0.00000000
traffic_source3          traffic_source3  0.00000000
topic_conf_level        topic_conf_level  0.00000000
category_confidence category_confidence  0.00000000
```



This model shows that important attributes are: ad_id, display_id, document_id, traffic_source2, category_id, platform2 and platform3.

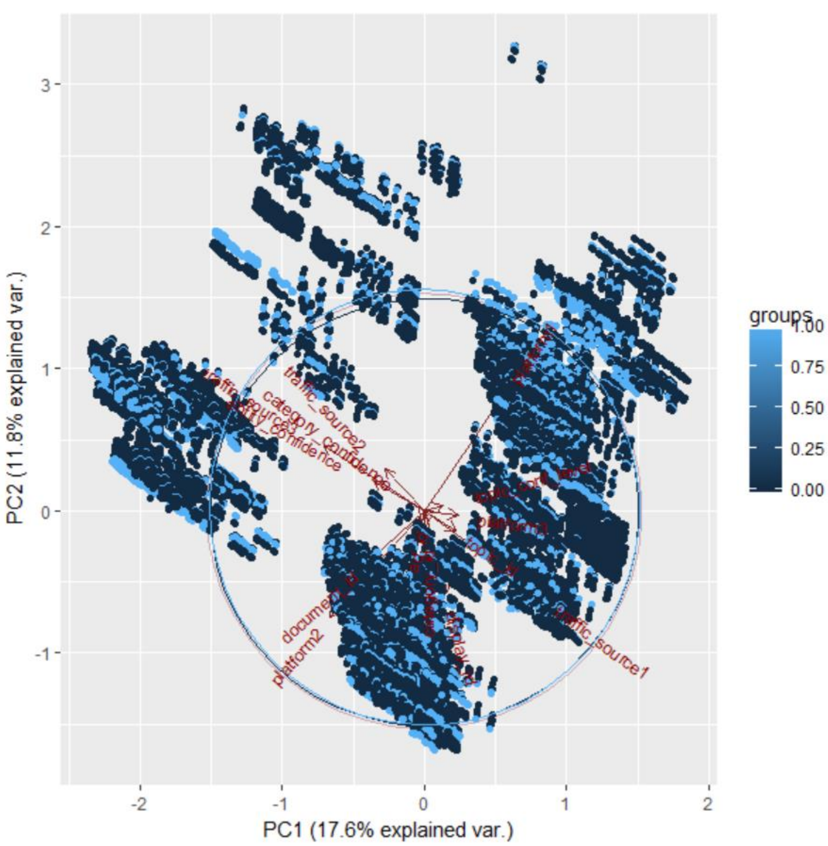
Variance of the Dataset



The graph clearly shows that the first 6 attributes have variance greater than 1. This in turn shows that infact these attributes are of major importance while determining whether the advertisement would get clicked or not.

Also this graph verifies the finding done above for variable importance.

The below shown biplot further verifies it.



Linearity Test—SVM

Support vector machines are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories (class labels either 0 or 1), an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a ***non-probabilistic binary linear classifier***. An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Because of their this property they are used to check whether a model is linear or not. Here as shown below the accuracy of the model using different combinations of attributes comes out to be maximum 93% but not a complete 100%. Thus, the given data set is not linearly separable.

Model 1:

Code :

svm.model <- svm(clicked~ad_id+display_id, train, cost = 100, gamma = 1);

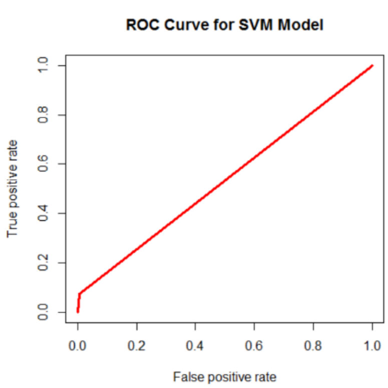
Accuracy : 79.77%

svm.model is the model created

svm is the function used for creating svm model

train is the dataframe used for training the model

cost is taken as 100

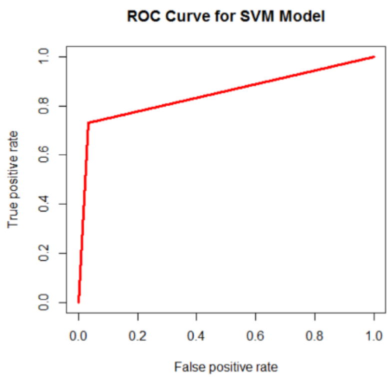


Model 2:

Code :

```
svm.model <- svm(clicked~ad_id+display_id+document_id+category_id+platform2+entity_confidence, train, cost = 100, gamma = 1);
```

Accuracy : 91.53%

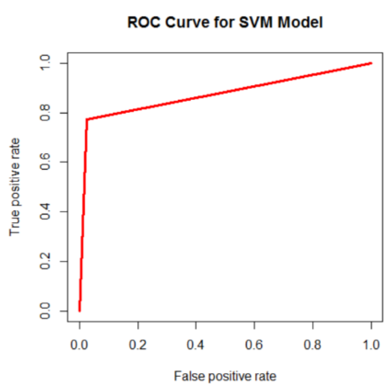


Model 3:

Code :

```
svm.model <- svm(clicked~ad_id+display_id+document_id+category_id+platform2+entity_confidence+traffic_source2+traffic_source3, train, cost = 100, gamma = 1);
```

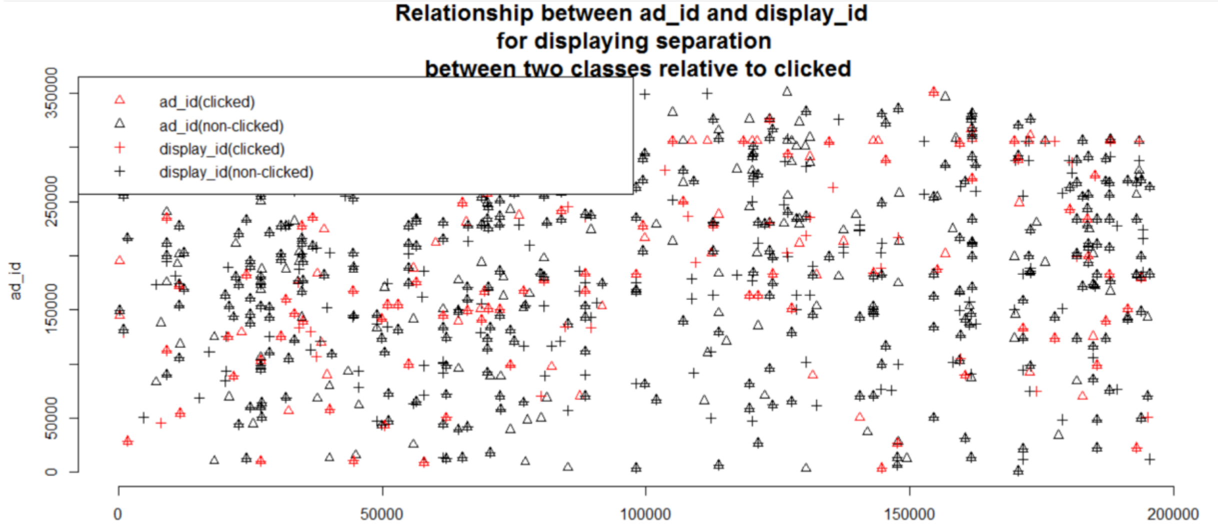
Accuracy : 93.17%



Model_1	Model_2	Model_3
ad_id+ display_id	ad_id+display_id+document_id+category_id+platform2+entity_confidence	ad_id+display_id+document_id+category_id+platform2+entity_confidence+traffic_source2+traffic_source3
79.77%	91.53%	93.17%

Graph for Linearity Test (Variable Separation Test)

Also, the following graph shows the same. As seen from the graph the data for the clicked and non-clicked overlap each other and cannot be separated by a clear straight line. Hence it is not linearly separable.



Model Creation:

Once we are done preprocessing the data we will move ahead and create corresponding machine learning models.

The process of training an ML model involves providing an ML algorithm (that is, the *learning algorithm*) with training data to learn from

The training data must contain the correct answer, which is known as a *target* or *target attribute*. The learning algorithm finds patterns in the training data that map the input data attributes to the target (the answer that you want to predict), and it outputs an ML model that captures these patterns.

We have created several models which have been mentioned below with a concise interpretation of them.

Evaluation Techniques:

Various evaluation Techniques are employed for obtaining precise accuracy. They are :

ROC Curve:

In statistics, a receiver operating characteristic (ROC), or **ROC curve**, is a graphical plot that illustr ates the performance of a binary classifier system as its discrimination threshold is varied. The cu rve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at vario us threshold settings

An **ROC curve** demonstrates several things:

1. It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity).
2. The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test.
3. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Confusion Matrix:

ConfusionMatrix function can be used from the caret package for creating the matrix based on the actual and the predicted value.

Root Mean Square Error:

RMSE or root-mean-square deviation (RMSD is a measure of the differences between values (sample and population values) predicted by a model or an estimator and the values actually observed.

Naïve Bayes

Naïve Bayes is a probabilistic model for data classification. It is based on applying Bayes' theorem with strong independence assumptions between the features.

In the Naïve Bayes we are finding the posterior for every class and then finding the one that gives the maximum value.

Advantages of using Naïve Bayes:

- It is highly scalable.
- It is simple yet powerful.

Model 1:

Code:

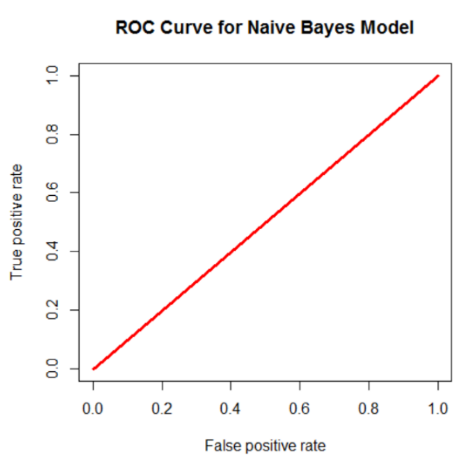
modelNB <- naiveBayes(clicked~ad_id+display_id,data = train)

Accuracy: 78.53%

modelNB is the model.

naiveBayes is the method for creating a probabilistic model.

train is the dataframe.

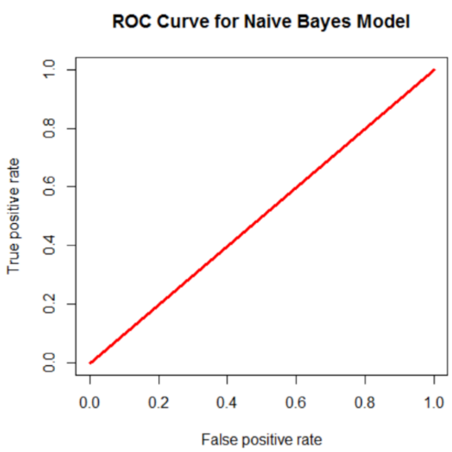


Model 2:

Code:

```
modelNB <- naiveBayes(clicked~ad_id+display_id+document_id+category_id+platform2+entity_confidence,data = train)
```

Accuracy: 78.53%

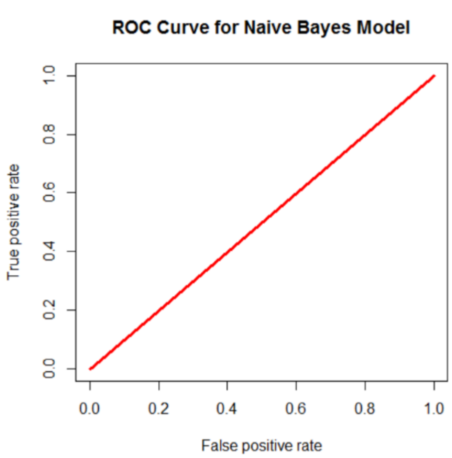


Model 3:

Code:

```
modelNB <- naiveBayes(clicked~ad_id+display_id+document_id+category_id+platform2+entity_confidence+traffic_source2+traffic_source3,data = train)
```

Accuracy: 78.13%



Model_1	Model_2	Model_3
ad_id+ display_id	ad_id+display_id+document_id+ category_id+platform2+entity_confidence	ad_id+display_id+document_id+category_id+platform2+entity_confidence+traffic_source2+traffic_source3
78.53%	78.53%	78.13%

KNN (K –Nearest Neighbors Algorithm)

The **KNN** or **k-nearest neighbors** algorithm is one of the simplest machine learning algorithms and is an example of instance-based learning, where new data are classified based on stored, labeled instances. More specifically, the distance between the stored data and the new instance is calculated by means of some kind of a similarity measure. This similarity measure is typically expressed by a distance measure such as the Euclidean distance, cosine similarity or the Manhattan distance

Determining the value of **k** plays a significant role in determining the **efficiency** of the model.

A **large k value** has benefits which include reducing the variance due to the noisy data.

Here first we created a subset of the attributes to be used in the **KNN** model and have tried and tested various combinations and have come out with the below set of attributes that ensures high accuracy with the low variance.

myvars<- c("document_id","platform1","platform2","platform3","traffic_source1","traffic_source2","traffic_source3","ad_id","topic_id","topic_conf_level","entity_confidence","category_id","category_confidence")

Once the subset is created we have created different **kNN** models with varying values of k

Creating a **KNN** model with different **k** values :

knn.1 <- knn(train.final_table2,test.final_table2,train.def,k=1)

In the model provide the training set along with the k value

train is a matrix or a data frame of training (classification) cases
test is a matrix or a data frame of test case(s) (one or more rows)

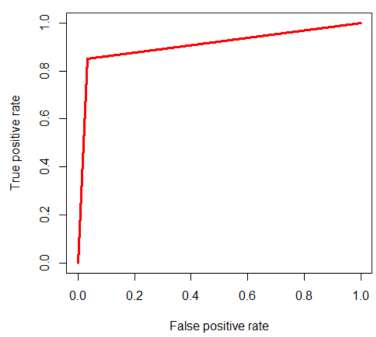
cl is a vector of classification labels (with the number of elements matching the number of classes in the training data set)
k is an integer value of closest cases (the k in the k-Nearest Neighbor Algorithm); normally, it is a small odd integer number

Once the model is created then calculate the **accuracy** :

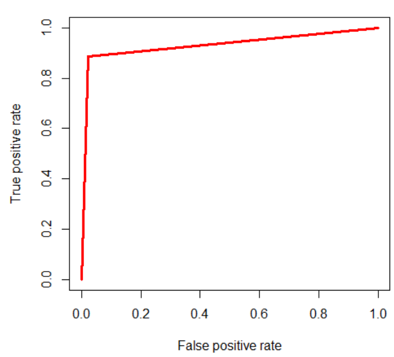
The accuracy can be calculated in different ways :

1) ROC Curve

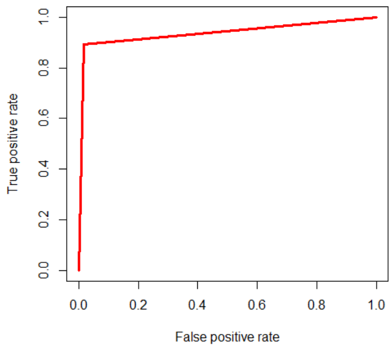
ROC Curve : KNN 1



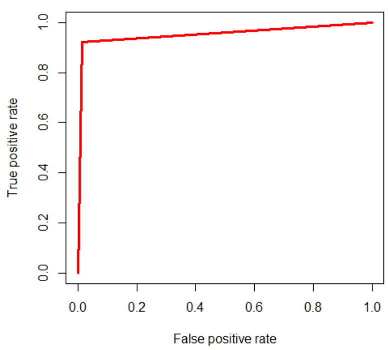
ROC Curve : KNN 5



ROC Curve KNN 10



ROC Curve : KNN 20



Here we can clearly see that the graph is more aligned towards left border and the top border for **KNN** with the **k** value as 20

Hence this model has the highest accuracy among the models mentioned above.

K Value	Accuracy
K= 1	72.41
K =5	73.28
K = 10	74.78
K = 20	77.47

As we have increased that with the increasing value of **k** the accuracy is increasing

Also we have to choose k large enough that the noise in the data in minimized and small enough so the samples of the other classes are not included

The value of **k** is determined by the dataset too specifically training data

Random Forest

Decision trees can suffer from high variance which makes their outputs weak to the specific training data used.

Building multiple models from samples of your training data, called **bagging**, can reduce this variance, but the trees are highly correlated.

Random Forest is an extension of bagging that in addition to building trees based on multiple samples of your training data, it also constrains the features that can be used to build the trees, forcing trees to be different. This, in turn, can give a lift in performance.

Random Forest is an **ensemble** learning based classification and regression technique. It is one of the commonly used predictive modelling and machine learning technique.

Key advantages of using **Random Forest**

- Reduce chances of over-fitting
- Higher model performance or accuracy

Implementation of Random Forest in R

Random Forest algorithm is built in **randomForest** package and the same function allows us to use that in R

In the given problem the output variable named **clicked** can be a linear combination of different variables

We created the random forest model by changing the number and type of input attributes in the combination and saw different accuracies.

We created three different models in this case:

Model 1:

model.final1 <- randomForest(clicked ~ document_id +platform2+platform3 +traffic_source2+ad_id+topic_id+topic_conf_level+entity_confidence+category_confidence,train.final,ntree=500, importance=T)

Here output variable clicked is a linear combination of different input attributes.

The **accuracy** in the above case came out to be **90.16%**

randomForest is the function name

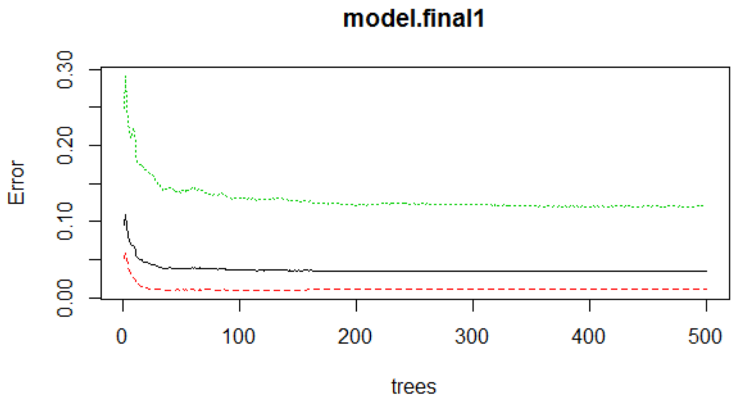
clicked is the output variable which is dependent on the mentioned input variables.

Train.final is the data frame used

Ntree specifies the number of decision trees to be grown.

In all the models we have fixed the number of variables tried at each split to be 3

plot(model.final1)



The **error rate** is plotted against the decision trees .The plot suggests that after a certain number of trees (50 in this case) the error rate showed a deviation. It first increased and then decreased

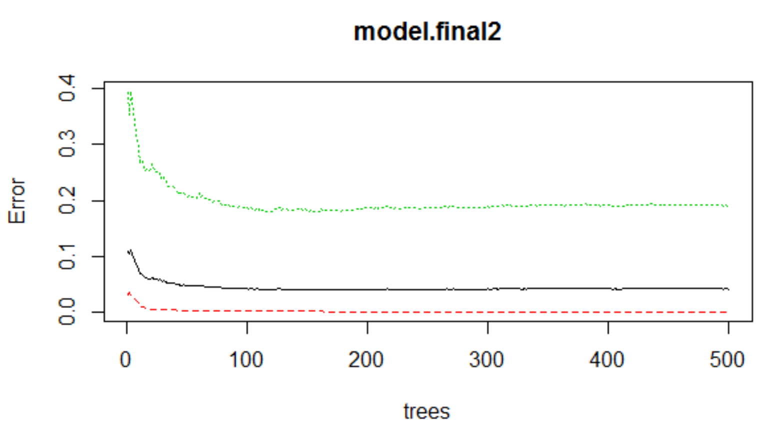
Model 2:

model.final3 <- randomForest(clicked ~ document_id+platform1+platform2+ +traffic_source1+traffic_source2+traffic_source3+ad_id+topic_conf_level+entity_confidence+category_id,train.final,ntree=500, importance=T)

In this model we have chosen a different set of parameters while creating the model

The **accuracy** in this case came out to be **91.1 %**

plot(model.final2)



The **error rate** is plotted against the decision trees .The plot suggests that after a certain number (350 in this case) of the decision trees there is not a reduction in the error rate.

Hence it does not matter after 350 trees how many more tree you create since there will be no reduction in the error rate.

Model 3:

The syntax of the **Random Forest** in R is :

model.final3 <- randomForest(clicked ~ document_id+platform1+platform2+platform3+traffic_source1+traffic_source2+traffic_source3+ad_id+topic_id+topic_conf_level+entity_confidence+category_id+category_confidence,train.final,ntree=500, importance=T)

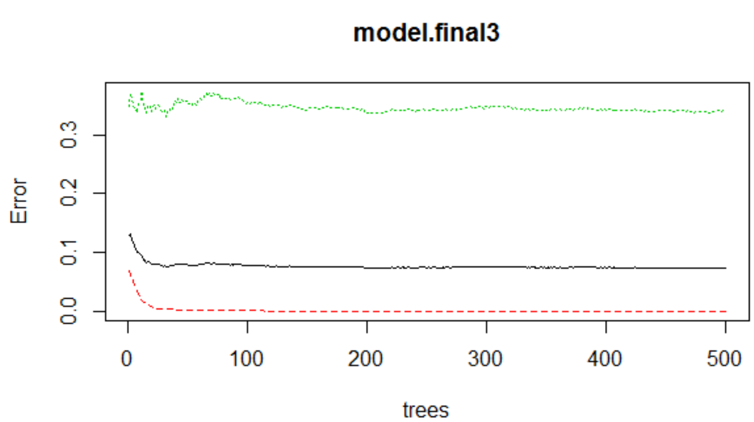
This was the model created with a different set of parameters

The above model gave the **accuracy** as **92.9 %** which is better than the former two cases hence we have chosen this model.

plot(model.final3)

The **error rate** is plotted against the decision trees .The plot suggests that after a certain number (200 in this case) of the decision trees there is not a reduction in the error rate.

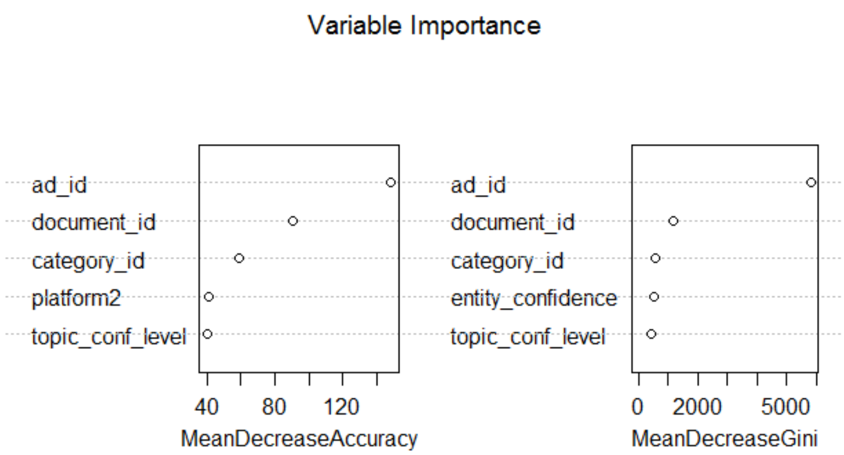
Hence it does not matter after 200 trees how many more tree you create since there will be no reduction in the error rate.



In all the three models the **ntree** value remains the same since we have already seen that there is not much reduction on the error rate and hence the value can remain the same.

While at the same time we have kept the threshold to default which in this case is **0.5**

We have used the **varImPlot** function to plot the variable importance .The variable can be selected for any other predictive modelling techniques.



Calculating the accuracy of the Random Forest

Confusion Matrix

Confusion Matrix : Model 1

Once the model is created we can create the corresponding confusion matrix which shows how many of the predictions were made correct.

OOB estimate of error rate: 9.84%
Confusion matrix:

	0	1	class.error
0	32345	391	0.01526648
1	1174	7689	0.45029391

Confusion Matrix : Model 2

OOB estimate of error rate: 08.90%
Confusion matrix:
0 1 class.error
0 34789 401 0.01612658
1 1280 7784 0.39015254

Confusion Matrix : Model 3

OOB estimate of error rate: 7.10%
Confusion matrix:
0 1 class.error
0 36589 580 0.01434571
1 1450 8124 0.299245849

As we can see the overall error rate in the confusion matrix for the **model 3** is less hen ce this is the preferred model.

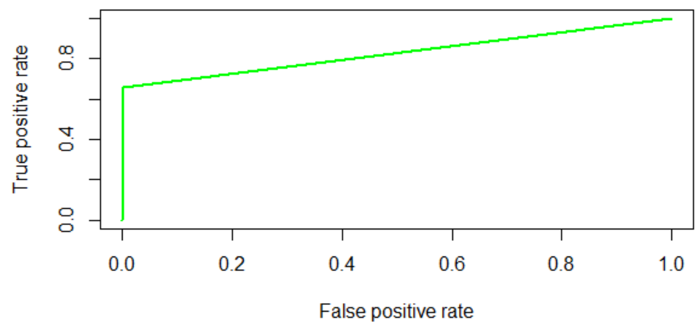
Table

In this accuracy is calculated by adding the **diagonal** elements and hence dividing them by the total number of elements in the matrix.

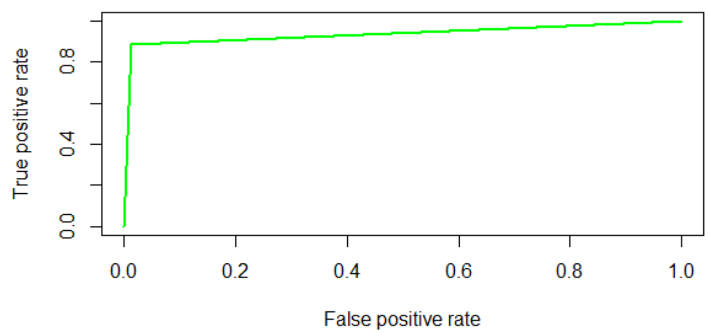
Model 1	Model 2	Model 3
91.66 %	92.1 %	92.9 %

In our case we have achieved the accuracy of **92.9%** which is not bad

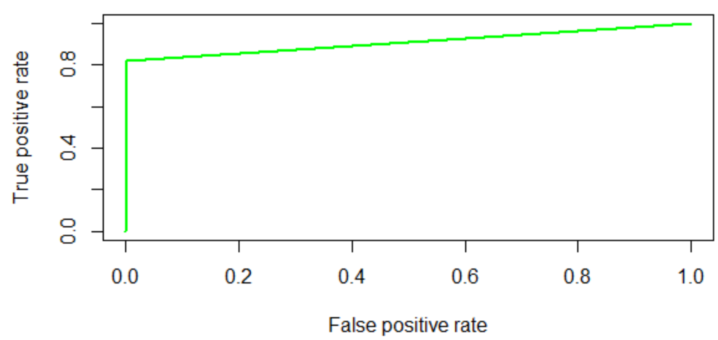
ROC Curve : Model 1



ROC Curve : Model 2



ROC Curve : Model 3



Here we can clearly see that the ROC Curve for the model 3 is closest to the left border and the t op border hence this model is considered to be the best among the three models mentioned ab ove.

Boosting

Boosting is a machine learning ensemble meta-algorithm for reducing bias primarily and also variance in supervised learning, and a family of machine learning algorithms which convert weak learners to strong ones.

Boosting algorithms consist of iteratively learning weak classifiers with respect to a distribution and adding them to a final strong classifier. When they are added, they are typically weighted in some way that is usually related to the weak learners' accuracy. After a weak learner is added, the data are reweighted: examples that are misclassified gain weight and examples that are classified correctly lose weight.

Key advantages of using Boosting:

- Higher model performance accuracy

Model 1:

Code: boosting <- train(clicked~ad_id+display_id, method = "gbm", data = train, verbose = F, trControl = trainControl(method = "cv", number = 10))

Accuracy : 84.89%

Confusion Matrix:

	true	
pred	0	1
0	2203	415
1	6	205

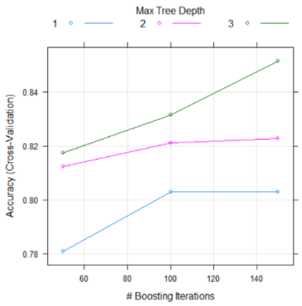
boosting is the model created

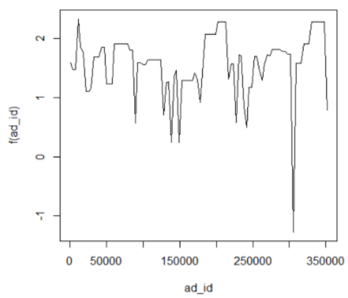
train is the method for making boosting model

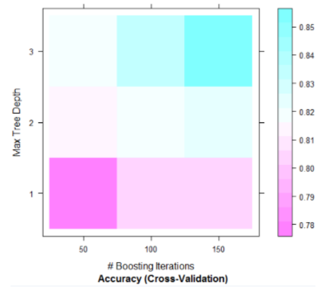
data=train is the dataframe used

trainControl is the method used for making n cross validation where here number = 10 defines it.

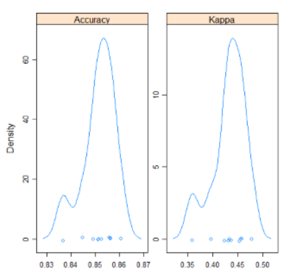
Boosting Graph | Error of the Boosting Model | Max Tree Depth Graph

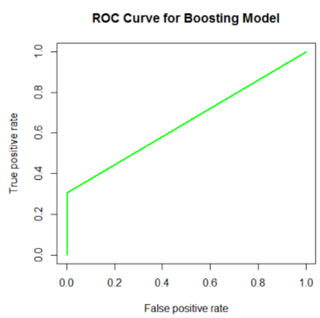






Root Mean Square Error Graph | ROC Curve





Model 2:

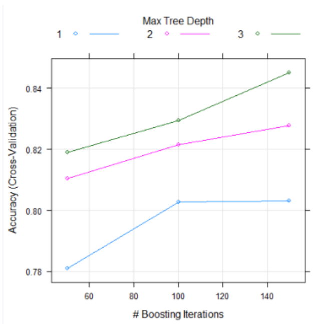
Code: **boosting <- train(clicked~ad_id+display_id+document_id+category_id+platform2+entity_confidence, method = "gbm", data = train, verbose = F, trControl = trainControl(method = "cv", number = 10))**

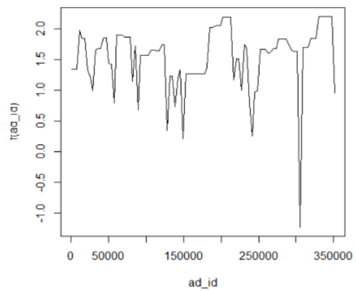
Accuracy : 85.03%

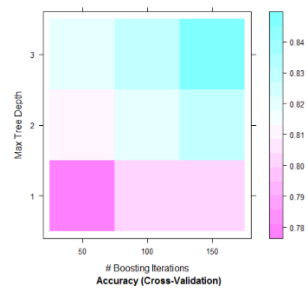
Confusion Matrix:

	true	
pred	0	1
0	2203	415
1	6	189

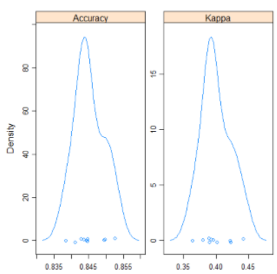
Boosting Graph Error of the Boosting Model Max Tree Depth Graph

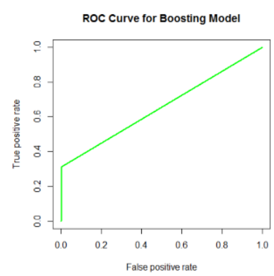






Root Mean Square Error Graph ROC Curve





Model 3:

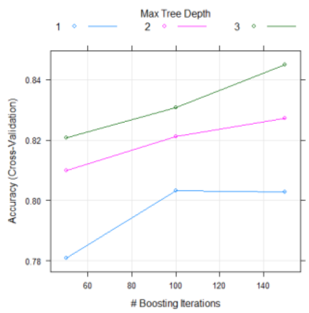
Code: boosting <- train(clicked~ad_id+display_id+document_id+category_id+platform2+entity_confidence+traffic_source2+traffic_source3, method = "gbm", data = train, verbose = F, trControl = trainControl(method = "cv", number = 10))

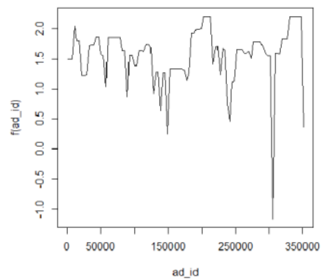
Accuracy : 84.61%

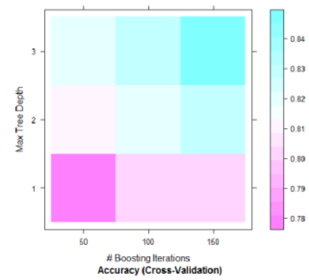
Confusion Matrix :

	true	
pred	0	1
0	2203	427
1	6	177

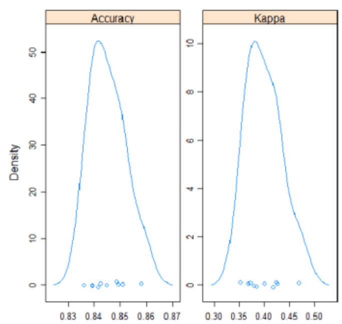
Boosting Graph | Error of the Boosting Model | Max Tree Depth Graph

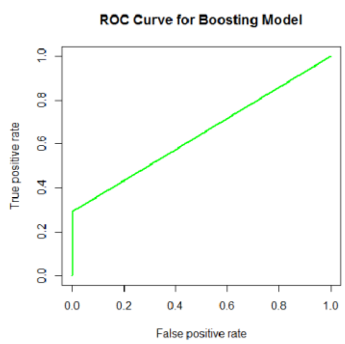






Root Mean Square Error Graph | ROC Curve





Model_1	Model_2	Model_3
ad_id+ display_id	ad_id+display_id+document_id+category_id+platform2+entity_confidence	ad_id+display_id+document_id+category_id+platform2+entity_confidence+traffic_source2+traffic_source3
84.89%	85.03%	84.61%

Conclusion

The Outbrain Click Prediction project takes into account the various features which play a vital role in determining the inherent human behavior. The target of performing this complex computation on huge dataset is to determine what important features actually contribute to users clicking an ad. As we performed various analysis it became clear that there were relevant indicators which were crucial for determining this:

1. Platform_2(mobile)
2. entity_confidence (documents_entities.csv gives the confidence that the given entity was referred to in the document)
3. traffic_source2(search)
4. traffic_source3(social)

Further by using classifiers such as Naïve Bayes, KNN, Random Forest, Boosting and SVM helped us to train models based on these features. These classifiers allowed us to show that as the classifier became more advanced with regard to better handling complex data the accuracy increased for this multi-dimensional dataset. This can be very well seen in the results of the various classifiers which showed how the classifiers like Random Forest and Boosting with SVM were able to achieve high accuracy with this data. So we conclude that this high dimensional dataset continues to improve with more data and better classifiers like Random Forest and Boosting.

Contribution of Team Members

Each Team Member equally contributed at each step of this project. Every Team member performed all the activities of pre-processing considering the large number of datasets we had. Once we analyzed the data after feature selection each team member ran one classifier model and performed the analysis for their classifier. This report is a work of collaboration of multiple individuals.

References

- CTR Prediction for Contextual Advertising: Learning-to-Rank Approach by Yukihiro Tagami, Shingo Ono, Koji Yamamoto, Koji Tsukamoto and Akira Tajima
- https://www.kaggle.com/c/outbrain-click-prediction
- https://www.dpfoc.com/blog/what-is-outbrain
- https://www.r-bloggers.com/
- http://machinelearningmastery.com/
- http://blog.revolutionanalytics.com/
- http://topepo.github.io/caret/