**1. Explanatory Data Analysis**

Training data has 184,903,890 rows and test data has 18,790,469 instances. So, we are dealing with a big data problem. Each row of training data has the following features

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

Test data is like training data except that it excludes variable attributed\_time and uses click\_id variable to index clicks. Table 1 and Table 2 show the first 4 rows of train and test data respectively.

Table 1. The first 4 rows of training data

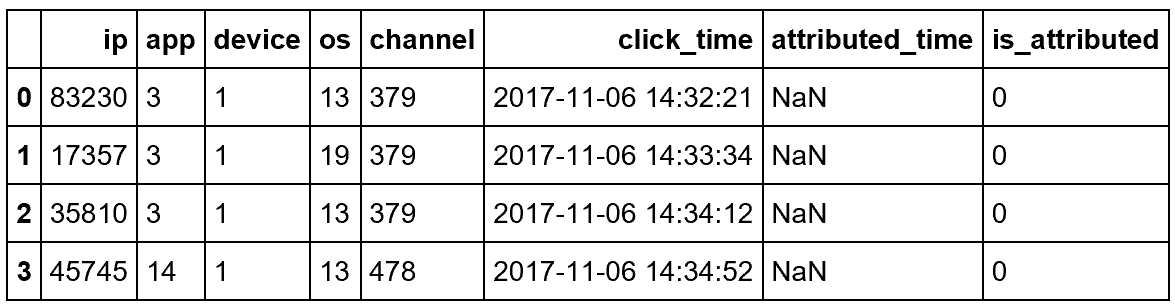
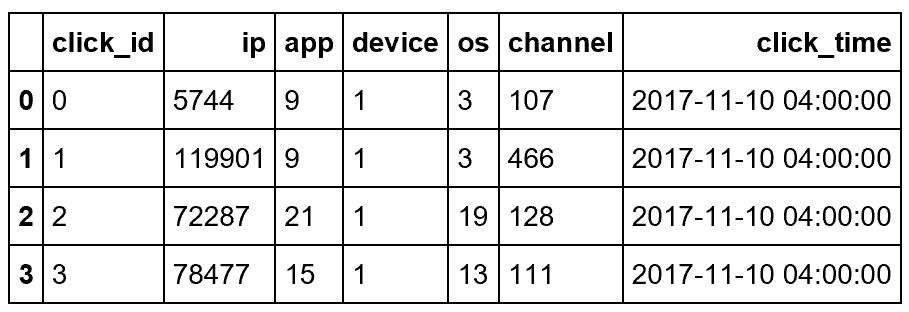


Table 2. The first 4 rows of test data



Our observations from training data are listed as what follows:

1. The data set is clean and has no missing values.
2. Target variable, is\_attributed, is extremely imbalanced with 0.25% instances labeled as 1 (click with download) and 99.75% labeled as 0 (click without download). See Figure 1 in Appendix A.
3. The variable attributed\_time is only available in training data, so it cannot be directly used in classification. We can either remove this variable from training data or predict its value for test instances based on other variables in the data set.
4. To get useful information from click\_time, we replaced click-time by 4 new columns indicating day, hour, minute, and second corresponding to timestamp of each click. Year and month of click-time are not of interest because their values are the same for all instances in training and test.
5. Categorical variables in this data set have many categories. For example, as shown in Figure 2, variable ip has 277396 categories. Most of Machine learning algorithms cannot handle categorical variables immediately and some machine learning algorithms such as random forest has limitation on the number of categories. These variables first need to be converted to several dummy variables based on the so called One Hot Encoding approach before they are used by machine learning algorithms.

**2. Initial Models and Results**

To fit a classification model and get initial results, we took a sample of 100,000 instances, and fit two models. We removed the feature called attribute\_time since it was not included in the test dataset of the competition. The dataset was split so that we use 1/3 of it for testing purposes. Because of class imbalance, we have used class\_weight to address the issue in each of the classifiers below. For instance in XGBoost, the scale-pos-weight is set to 470. Note that the results below are on a sample of data, we will investigate the whole data later.

**2.1. Random Forest**

The first one is a Random Forest (RF) model with 150 trees (weak learners) and 200 features (to be randomly selected to consider splitting on in each splitting iteration of every tree). Although we only have six features that are used as predictors in our classification models, i.e. ip, app, device, os, channel, and click\_time, we needed to make dummy variables because five of the variables are categorical. This increased the number of predictors to almost 70,000. See the results in Figure 3.

**2.2. XGBoost**

Next, we fit a Gradient Boosting Tree model (XGBoost). For this model, we have omitted the variable “ip” because of the computational and memory considerations as it contains a lot of noise and has a lot of categories, resulting in myriad of dummy variables. The number of estimators used is 160, gamma 0.2, and max\_depth of 7. See the results in Figure 4. Results look a little worse than RF but still good.

**3. Feature Engineering**

There are two types of variables in the raw dataset namely categorical and numeric. The categorical variables are app, device, os, channel and the target variable which is called is\_attributed. The other variables include the click\_time which is considered numeric and the ip which could be considered both categorical or numeric, but we consider it as a continuous numeric variable for two reasons:

* First, there are thousands of levels for this variable and this can create extensive number of dummy variables.
* Second, after careful consideration of the data, we saw a behavior that is not very reasonable but could be because of some transformation or preprocessing that the creator of the data has done before publishing it. The ip values do not follow the conventional format and they are rather integer numbers. Moreover, the higher that number is, the higher the number of clicks associated to that specific ip. This does not make sense in the first sight, but it is true as can be seen in the following plot. The data could be collected in a way that more clicks are associated with lesser numbered ip’s but anyway this characteristic is visible. See Figure 5 in Appndix A.

Hence, we made this assumption that ip variable should be a continuous variable since otherwise we lose the ordinal information that exists among ip values. Note that the existence of the ip variable in the first place was proven to be useful in our initial results presented in Part 2.

Before moving to other feature engineering aspects of our work, we converted the click\_time variable into three new variables for each instance :

(day or night)

(day number)

(click hour)

in which is the time in seconds that the click has happened and is the hour in which the click has happened, indicates whether it happened during day or night and is a categorical variable, is the day number (as the clicks happened during 4 consecutive days) and is an ordinal variable which we also treated as a continuous variable. Finally,  **is a** **cyclical version of time** variable, meaning that it starts at midnight from 0, goes all the way to 12\*3600 at noon and again goes back to 0 at midnight. This way, we kept the distances more realistic between those times that are close but far in the conventional clock time system. For instance, in the conventional system, 11:59 and 00:01 are very different in numbers (especially if you convert them to seconds), but, they are merely 2 minutes different. This last variable is also considered a numeric variable.

The next step in our feature engineering is to scale the numeric variables to be in the same range as our categorical (and dummy) variables i.e. between 0 and 1. This was done through normalizing the ip, , and features.

Finally, the we treated our categorical variables in two different manners each of which has its own pros and cons in terms of performance. The first method is feature hashing for which we used the FeatureHasher function with features which is a lot of features, but we know that because of the efficient use of memory via sparse matrices, we should not worry about it. The other method we used is more innovative and makes use of dummy variables. Although in big data analysis it is not straight forward to use this method as in each chunk of data the dummy variables created can be different and there is no package in python to deal with that, we retrieved all the possible labels for categorical variables at hand, and input it to our get\_dummies function to create “all-zero” columns for those categories that do not happen in some specific iteration of partial fit (if applicable). We observed that using this method could make the algorithms run more efficiently in terms of computational time, on the other hand, it adversely affects the memory complexity as the number of features that are created and stored in a “DataFrame” are high. In our future computations, we will try to store it as a sparse matrix, so that we can have the best of both worlds.

**4. Final Models and Results**

In this section, we investigate procedures to provide a classification model appropriate for our dataset. In order to evaluate our proposed model performance we discuss several metrics. Test accuracy is one of the widely used metrics to evaluate quality of classification solution. As discussed before, we have a highly imbalanced dataset and this property makes test accuracy an inefficient evaluation metric. One could simply classify every data point as the majority class and yet get very high test accuracy.

Because of these limitations, we use other metrics for imbalanced datasets. Several classification methods not only provide class of each data point but also give the probability of that data point belonging to that specific class. Using these probabilities we can consider a cut-off probability level as our model’s hyper-parameter. Usually we set this cut-off level at 0.5 but we may change it and for every cut-off level between 0 and 1 we will obtain a different classification solution. Using this procedure we are able to plot ROC curve.

Since ROC curve considers both false positive and false negative, we may calculate the area under this curve and use it as a good metric for imbalanced dataset. In fact, this metric (auc: Area Under Curve) is the evaluation metric used in Kaggle challenge. Therefore, throughout our model implementations we are using auc as our evaluation metric.

One challenge that arises in modeling big data is the limitation of storage and computational capacity. During implementation of this project one of the main challenges has been the storage and memory issue. Our training data has 200 million data points and we were not able to load this dataset into memory and consequently not able to fit model using classic method. In order to deal with this issue we utilized a very useful and intuitive method. Instead of reading all the data from hard-disk and putting it into RAM memory, we gradually read data from hard-disk and each time we only work with a subset of data. Methods like SVM and logistic regression utilize stochastic gradient decent method in order to work with gradual data input. Gradient gets updated only based on a subset of data. However, weights are initialize using the previously fitted model. Since reading data from hard-disk is not an efficient task in terms of running time, we try to read data in big chunks (as much as our memory allows). Inside each chunk of data we have batches of data relating to data subsets for stochastic gradient decent optimizer.

A similar idea is used to enable incremental learning in other classification methods such as random forest. In random forest each batch of data produces a new tree in our forest and we use that subset of data to create the tree using a greedy impurity based approach.

We implemented incremental learning with random forest for the training data. In each iteration model is fitted on a subset of data and is tested on a separate subset of data. Therefore, in each iteration we obtain a training auc and and a test auc. Then we plot each iteration’s auc versus amount of data that we provided for the dataset.

We considered 5000 data points in each batch of data corresponding to one tree in the forest. We consider dummy variables for categorical data and use the feature engineering methods mentioned before. Figure 6 in Appendix A is the learning curve of this model.

This learning curve suggests a drop in the training auc, which suggests that our model is not able to capture nonlinearity in larger sets of data. This behavior is an indicator of under-fitting and lack of flexibility of the model. It shows that even a complex model such as random forest is a shallow model and can under-fit given a large set of data points. Then, we increase complexity of the model by increase max depth of trees and max features that are used in each tree for node splits. Figure 7 in Appendix A is the learning curve of the updated model.

Finally, Logistic regression was run by a constant learning rate equal to 0.001 and L1 penalty. To deal with the imbalanced target variable (is\_attributed), class weight was set to 407 for class 1 and 1 for class 0 respectively. Data was read by using chunks of size 100,000 and model was fit within 10 iterations using data at each chunk. We evaluated the model is after each chunk (or 10 iterations) using a separate validation data containing 1000 rows.

Figure 8 in Appendix A shows the learning curve from logistic regression model in the first 110 chunks which contain 11,000,000 data. We can see a lot of variation in auc from both training and validation set. This means that the amount of data is not sufficient to obtain a good model with high auc on training and validation set. To improve auc, we can either increase the number of iterations and read more data or to use a more complex classifier.

**Appendix A. Figures**

**Figure 1.** Class Distribution of our data set:

A screenshot of a cell phone

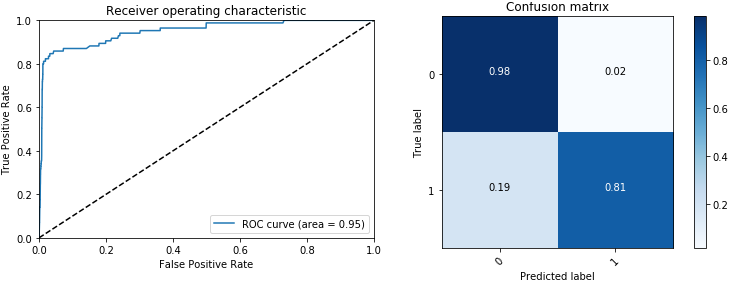
Description generated with high confidence

**Figure 2.** Number of unique values per categorical variable:

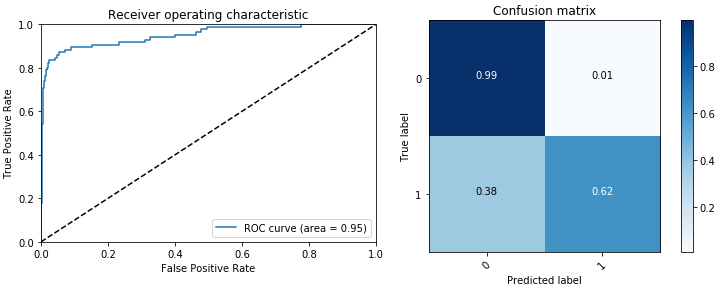
A screenshot of a cell phone

Description generated with high confidence

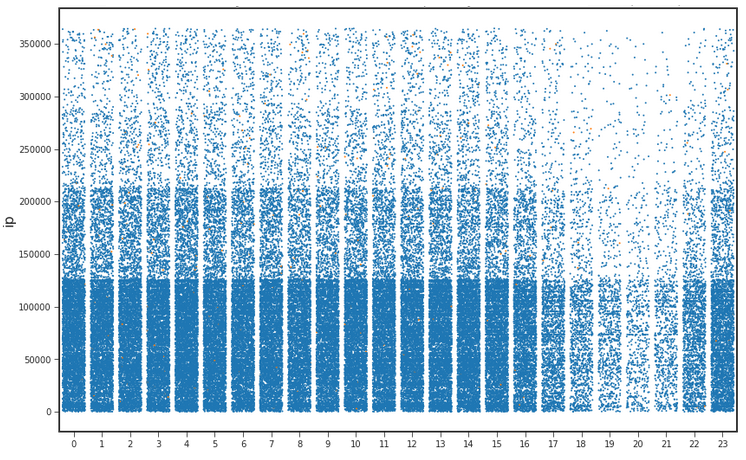
**Figure 3.** Initial test results of fitting a Random Forest classifier trained on a subset of training data (with IP field):



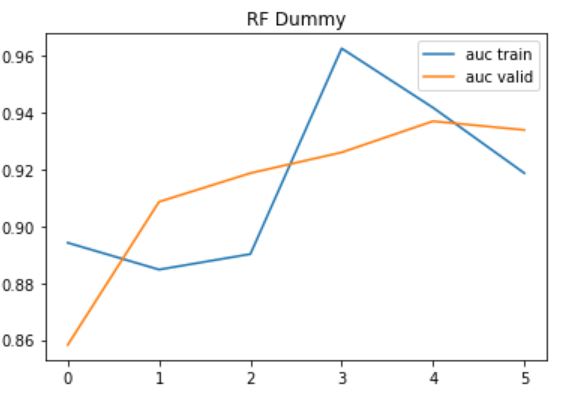
**Figure 4.** Initial test results of fitting an XGBoost classifier trained on a subset of training data (without IP field):



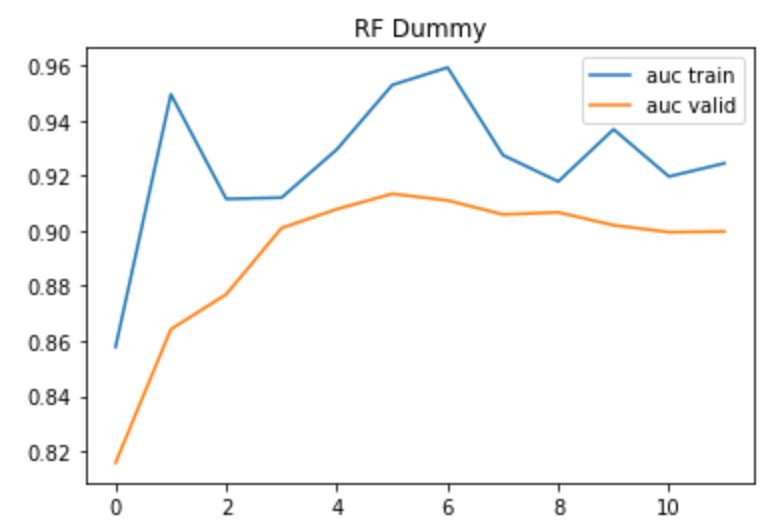
**Figure 5.** Number of Clicks Distribution based on ip versus time of day:



**Figure 6.** Learning curve of Random Forest using dummy variables:



**Figure 7.** Learning curve of the updated Random Forest using dummy variables:



**Figure 8.** Learning curve from logistic regression model with dummy variables:

**A screenshot of a cell phone

Description generated with very high confidence**

**6. References**

[1] James, Gareth, et al. An introduction to statistical learning. Vol. 112. New York: springer, 2013.

[2] “The Home of Data Science & Machine Learning.” Kaggle: Your Home for Data Science, www.kaggle.com/.