Machine Learning Engineer Nanodegree

Capstone Project on TalkingData AdTracking Fraud Detection Challenge

Sakshi
September 22, 2019

I. Definition

Project Overview

Fraud risk is everywhere however, with online advertisement, adfraud can happen at an overwhelming volume resulting in misleading click data and wasted money. Online advertising is a growing, multi-billion-dollar market. It is predicted that global digital ad expenditure will reach US$225 billion — accounting for 44% of total ad expenditure. The sheer size of this market tempts criminals and hackers into creating technology and techniques to steal money from the advertisers. It is also estimated that in 2019 ad fraudsters will steal $5.8 billion from advertisers, however, because some types of ad fraud are very hard to detect, and the technology to protect advertisers is immature, the actual figure may be much higher.

In this project, I trained and tested a binary classifier capable of predicting whether a given click on an advertisement by a mobile user would result in the advertised app to be downloaded or not. I trained the model using the Light gradient boosting algorithm (Light GBM), and the dataset was provided by the big data firm TalkingData for their Kaggle competition.

Problem Statement

Talking Data is one of the most prominent big data platforms, covering more than 70% of mobile devices' activities across China. It is a third-party platform which provides analytics for small and medium-sized mobile applications. They handle 3 billion clicks per day, of which 90% are potentially fraudulent. Their strategy is to measure the journey from click to install and flag the IP’s/devices that generate fake clicks which never lead to installing the app. This helped them to build a portfolio for all the fraudulent IP’s and devices to be considered for blacklisting. Some mobile applications us Pay-per-click (PPC) model which help direct traffic to their applications. The sole purpose is having more clicks that will help to have more installation of their applications. By having millions of clicks which are actually fraudulent will lead the advertisers to pay a hefty amount of money as they are charged based on each click. The other reason fraudster does click spamming is that by bombarding the system with millions of clicks, malicious parties tries to affect the daily operation of mobile applications as it increases the load on the server.

In the AdTracking Fraud Challenge of Kaggle, they want to determine the probability of app download by the user after clicking the ad. Low probability means it is a click fraud and high probability means a genuine user. This model would be helpful to increase their solution's accuracy in identifying fraudsters.

To start with, I tried to get some basic understanding of the sample training data to check for its characteristics such as balanced or imbalanced dataset and unique values of features available in the dataset, Click Vs install behaviour of IP's, daily, hourly trends of clicks. With the actual training and test data, I also generated new features instead of only using the available categorical features. Based on these categorical features, I tried to generate trends of clicks with different combinations such as: ip-app, ip-os, ip-app-os. Finally, I chose Light GBM algorithm to train the model as the dataset is too large and this algorithm is comparatively faster and more efficient than the Gradient Boosting algorithm. However, the training dataset is around 180 million records which takes longer time to train. I used Kaggle kernel to do the EDA and model training. Hence, I restricted the training samples to be 30 million and test samples to be 7.5 million. 10% of the training samples are used for validation.

Metrics

Accuracy cannot be considered a s a good metric for the model evaluation as the data is highly imbalanced and it is very easy to attain good accuracy value.

Area Under Curve (AUC) is one of the most widely used metrics for evaluation for highly imbalanced datasets like Adtracking fraud dataset. It is used for binary classification problem. AUC of a classifier is equal to the probability that the classifier will rank a randomly chosen positive example higher than a randomly chosen negative example.

AUC can be explained easily with true positive rate and false positive rate.

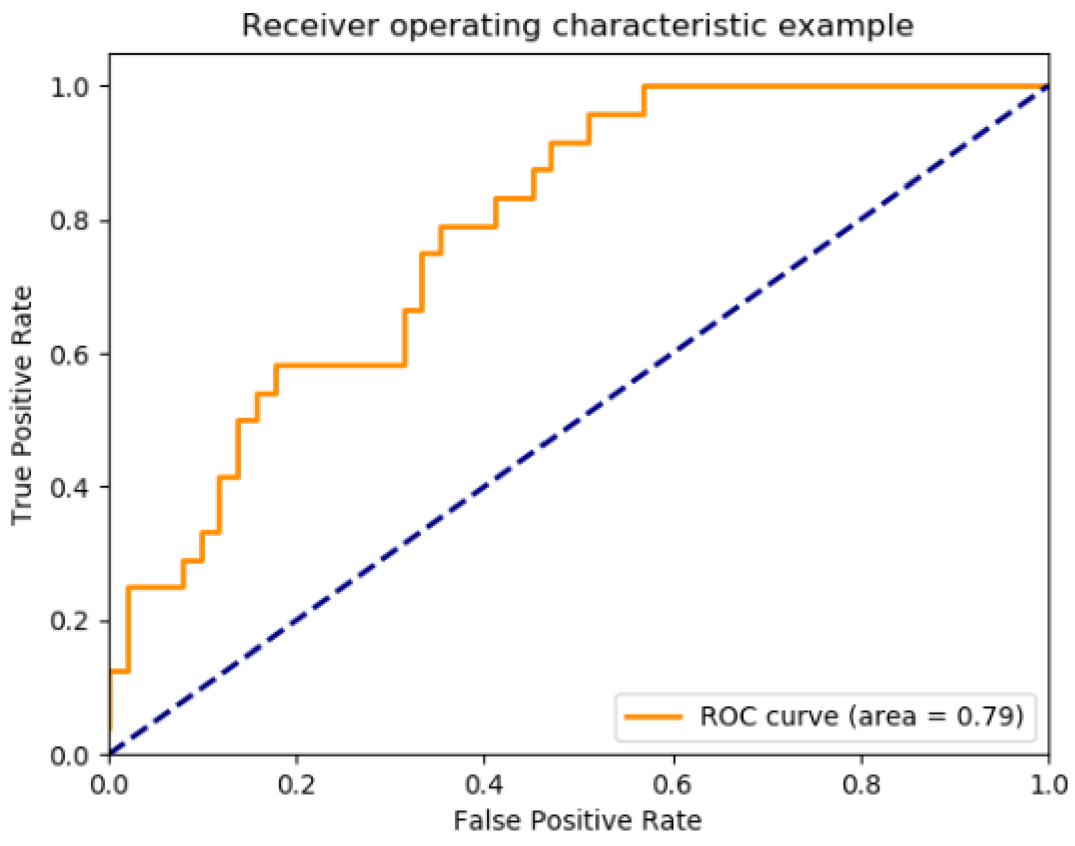
- True Positive Rate (Sensitivity): True Positive Rate is defined as TP/ (FN+TP). True Positive Rate corresponds to the proportion of positive data points that are correctly considered as positive, with respect to all positive data points.

$$TruePositiveRate = \frac{TruePositive}{FalseNegative + TruePositive}$$

- False Positive Rate (Specificity): False Positive Rate is defined as FP / (FP+TN). False Positive Rate corresponds to the proportion of negative data points that are mistakenly considered as positive, with respect to all negative data points.

$$FalsePositiveRate = \frac{FalsePositive}{FalsePositive + TrueNegative}$$

False Positive Rate and True Positive Rate both have values in the range [0, 1]. FPR and TPR both are computed at threshold values such as (0.00, 0.02, 0.04, …., 1.00) and a graph is drawn. AUC is the area under the curve of plot False Positive Rate vs True Positive Rate at different points in [0, 1].



As evident, AUC has a range of [0, 1]. The greater the value, the better is the performance of our model.

Reference: https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234

In this dataset, there is no reference test target values were provided to test the accuracy of the trained model. In the Kaggle competition also, grading is done by uploading the file containing the download probability of each click in the test data to Kaggle.

II. Analysis

Data Exploration

The dataset available in the Kaggle competition includes training set with more than 180 million rows of data, each has the timestamp of the click, IP addresses (in number code), device (in number code), device's operating system (in number code), app (in number code), channel (in number code), whether the click resulted in a download or not. The dataset also have dedicated test set having about 18 million clicks with each click associated with an ID and other information excluding the is_attrubuted field (app download or not label). It also includes a sample training set for data exploration with 100,000 records and features same as training data.

Number of samples:

- train.csv: 184,903,890
- test.csv: 18,790,469
- train_sample.csv: 100,000

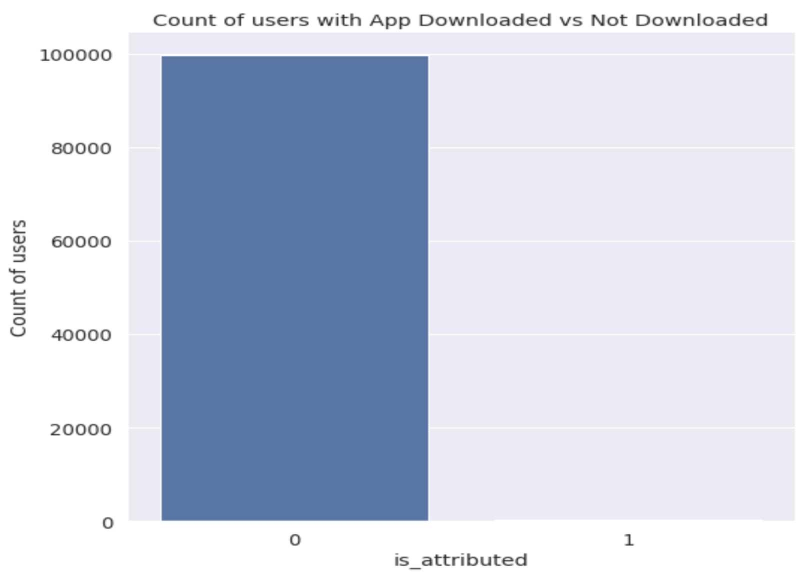
Features:

- Common in training and testing datasets:
    - ip : IP address from which the click was registered, encoded for privacy
    - app : the app whose advertisement was clicked on, encoded for privacy
    - device : label of the model of the device on which the click was made, number encoded
    - os : os version of user mobile phone, number encoded
    - channel : the channel on which the app advertisement was put and clicked, number encoded
    - click_time (yyyy-mm-dd hh-mm-ss): the time when the click was made
- Training set:
    - is_attributed: whether this click resulted in a download or not; 1 - yes, 0 – no, this act as the target variable.
- Testing set:

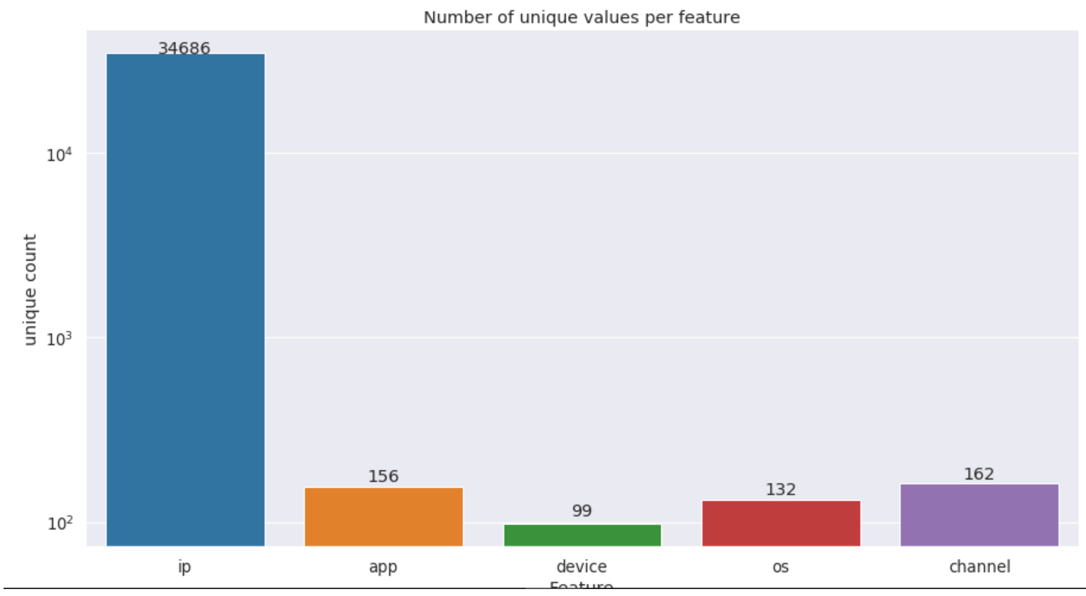
- click_id: used for identifying the click. This act as a reference for making predictions.

Observations:

- Sample training data shows that the data is highly imbalanced with even less than 0.3% clicks resulted in downloads.

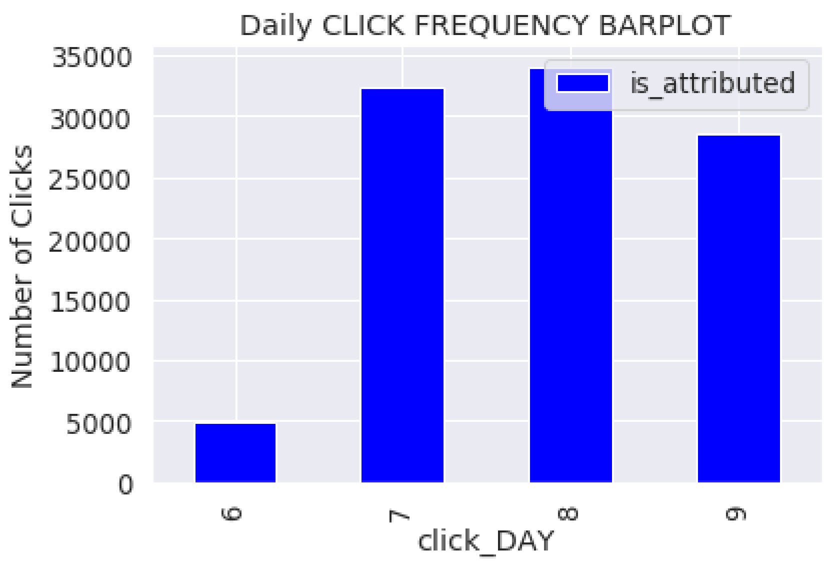


- I also tried to analyse the distribution of the different features already available in the dataset which is a good indicator for which features are of actual importance to find the fraud.

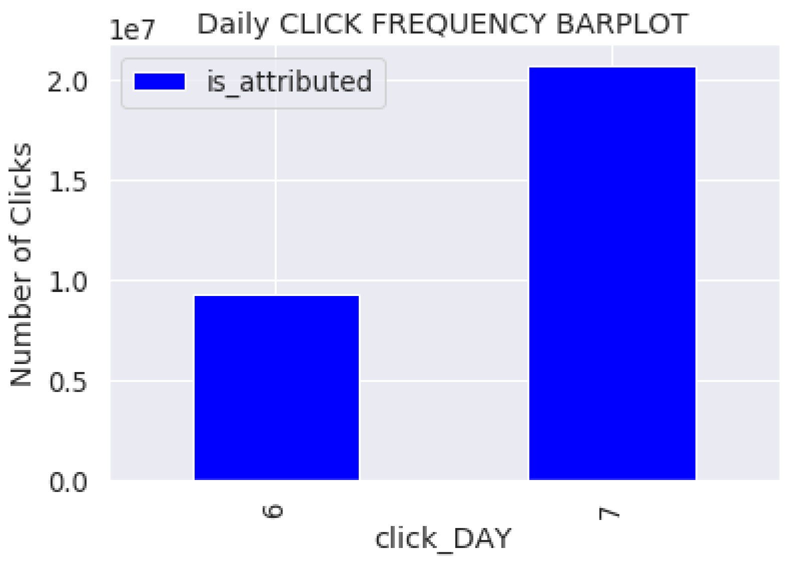


- The data was collected over the span of 4 days; the sample training set contains samples covering all the four days from 6/11/17 to 9/11/17. However, I have taken only 30 million records of the actual training data which does not have all the four days available.

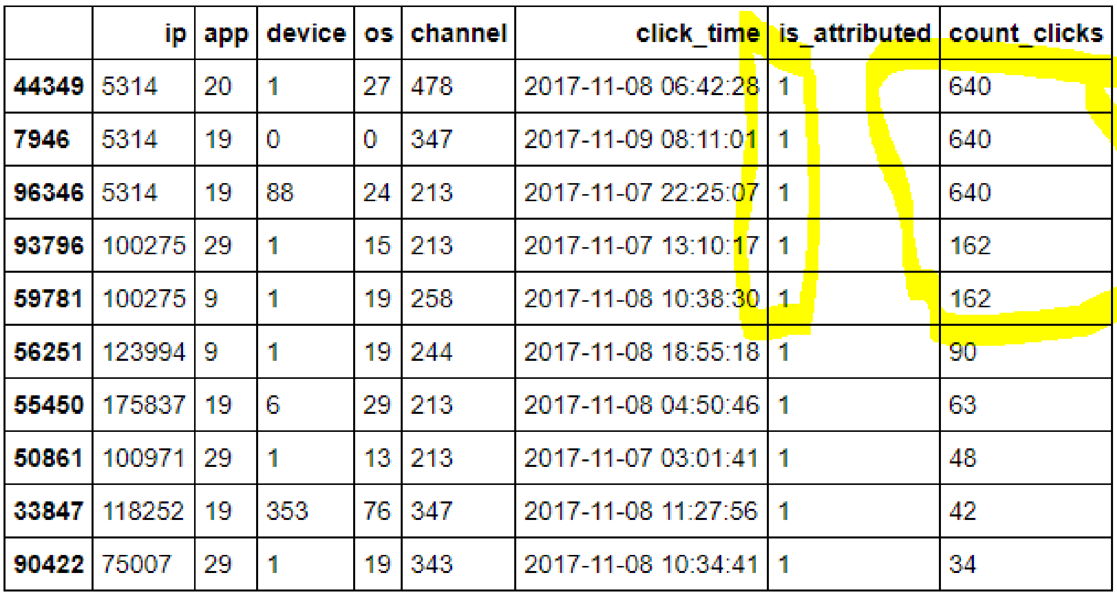
Training Sample day wise analysis:



Overall training data day wise analysis:



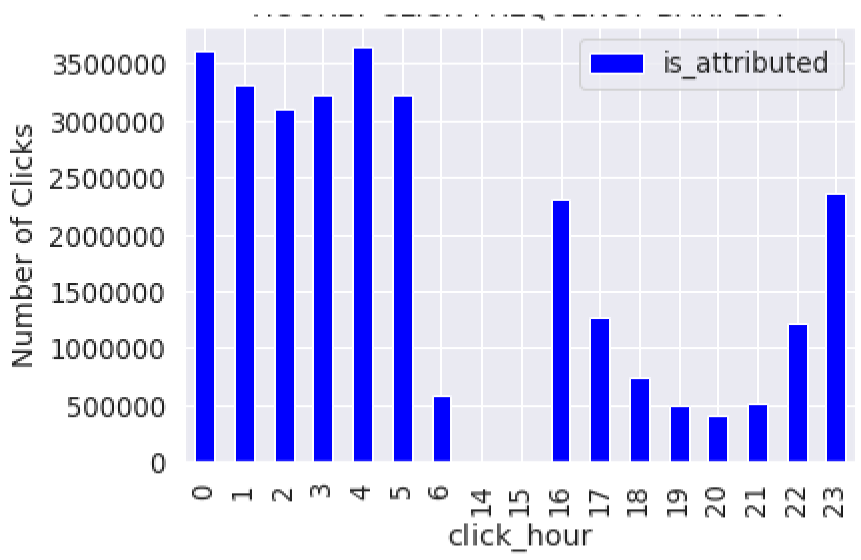
- I also tried to analyse the distribution of clicks with respect to IP and whether IP's with higher number of clicks do convert to installation of the app or not. The below graph shows the top IP with respect to number of clicks and whether the install happened or not.



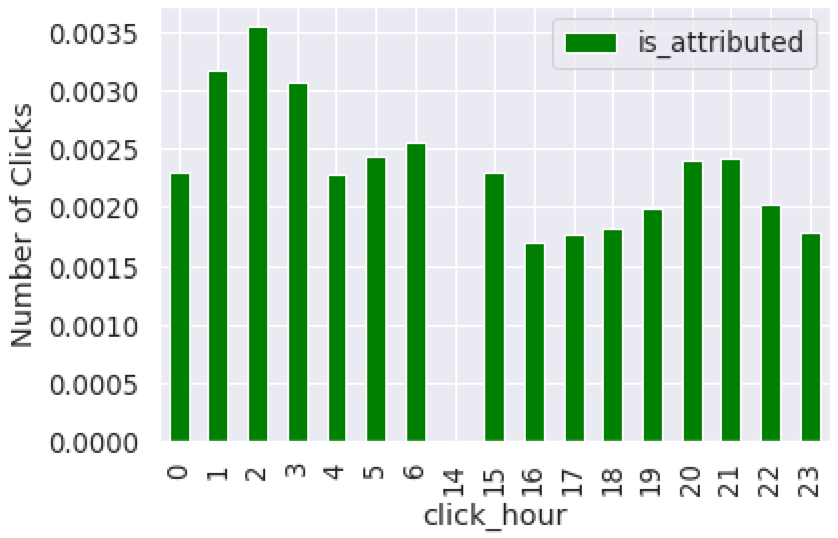
	ip	app	device	os	channel	click_time	is_attributed	count_clicks
44349	5314	20	1	27	478	2017-11-08 06:42:28	1	640
7946	5314	19	0	0	347	2017-11-09 08:11:01	1	640
96346	5314	19	88	24	213	2017-11-07 22:25:07	1	640
93796	100275	29	1	15	213	2017-11-07 13:10:17	1	162
59781	100275	9	1	19	258	2017-11-08 10:38:30	1	162
56251	123994	9	1	19	244	2017-11-08 18:55:18	1	90
55450	175837	19	6	29	213	2017-11-08 04:50:46	1	63
50861	100971	29	1	13	213	2017-11-07 03:01:41	1	48
33847	118252	19	353	76	347	2017-11-08 11:27:56	1	42
90422	75007	29	1	19	343	2017-11-08 10:34:41	1	34

- Test data have records for different days in comparison to training data, hence no common day wise trends (weekday/weekend) trend can be observed. Therefore, there is no point of deriving and using the day information of the click_time
- It is important to look at the hourly information of the click as I observed that the maximum clicks were happening during the odd hours of the day i.e. 2-4 am. Who gets up in the middle of the night and clicks on an application advertisement and then installs it. This feature is useful and is therefore considered for model training as well.

Clicks hourly trend:



Install Hourly trend:



- I am not considering just the IP value as a feature for this classification problem as IP is not specific for a user. It normally represents a group of users. Similarly, device, OS are also not specific to an individual user. Therefore, I have used these features to derive new features.
- While attributed_time is only available in the training dataset, so I am not considering this feature as it will only be useful to predict the install_time of the app. However, in this problem, I am trying to determine whether a click will result in installation of the app or not which is a binary classification problem.

Algorithms and Techniques

Light GBM is a fast, distributed, high-performance gradient boosting framework based on decision tree algorithm, used for ranking, classification and many other machine learning tasks. Since it is based on decision tree algorithms, it splits the tree leaf wise with the best fit whereas other boosting algorithms split the tree depth wise or level wise rather than leaf-wise. So when growing on the same leaf in Light GBM, the leaf-wise algorithm can reduce more loss than the level-wise algorithm and hence results in much better accuracy which can rarely be achieved by any of the existing boosting algorithms. Also, it is surprisingly very fast, hence the word 'Light'. Leaf wise splits lead to increase in complexity and may lead to overfitting and it can be overcome by specifying another parameter max-depth which specifies the depth to which splitting will occur.

Reference: https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/

There are a number of advantages of LightGBM which can be explained as follows:

1. **Faster training speed and higher efficiency:** Light GBM use histogram based algorithm i.e it buckets continuous feature values into discrete bins which fasten the training procedure.
2. **Lower memory usage:** Replaces continuous values to discrete bins which result in lower memory usage.
3. **Better accuracy than any other boosting algorithm:** It produces much more complex trees by following leaf wise split approach rather than a level-wise approach which is the main factor in achieving higher accuracy. However, it can sometimes lead to overfitting which can be avoided by setting the max_depth parameter.
4. **Compatibility with Large Datasets:** It is capable of performing equally good with large datasets with a significant reduction in training time as compared to XGBOOST.

Benchmark

There is no way to compare the results of the test set with any reference as the data available for the test set was to predict whether the given click would result into installation of the app. I have also downloaded the results of the test set and stored as a csv file.

However, one way to have a benchmark for this problem is model that can randomly guess whether a click would result into installation of the app or not. The guessed probability is either 0 or 1. On the validation dataset, the model scored 0.3523 of the ROC-AUC metrics measuring the closeness of the resulted probability graph versus that of the solution.

III. Methodology

Data Exploration

The given dataset had already been well-prepared and processed, all categorical features had been labelled and set to numbers, sensitive data encoded, and potentially missing data had been either filled or discarded. I have analysed the train_sample data to see the timely trends of the click such as how the click trends vary during specific hour of the day or how it varies with regard to days of the week. The hourly trend showed some fraudulent behaviour as how the maximum clicks happens during the odd hours. Similarly, the day information was not that useful because the training records are from 4 days of the week (weekday). If it would have covered both weekday and weekend, then we could have analysed this feature in detail. Month and year information was not at all worthy to consider as the data was from one month and one year. I also tried to analyse whether larger number of clicks from one source (IP) lead to install or not so that we can use that as a feature to represent the click fraud. The analysis shows that Ip's with higher number of clicks do install the app and it is not related to any click spam. However, this analysis is just based on the sample training data which is merely 100,000 records.

Data pre-processing and feature engineering

Before feeding the data to the model, I converted the time attributes to hour and daily attribute and derived new features combining the time attributes with other categorical features like ip, PP, OS AND DEVICE. The hour information was useful as it was showing some trends in the data during specific hours of the day which is not usual. The day information was not really useful as I extracted only 30 million samples of the day which could only cover 2 days out of the 4 days of click data. 4 day click information was available through the train_sample which had 100,000 records and covered all days of data.

Feature engineering

How many clicks per hour from an IP, clicks from same IP to same app?

Between a normal mobile user who clicks on an advertisement out of personal interest and a fraudster whose aim is to make as many empty clicks as possible, the fraudster would make many more clicks at a particular time of the day and from the same IP. Similarly, how many normal users click on the same advertisement from only one IP. This can be a fraudulent activity where large number of clicks to the same app from same IP. This is because in a typical case once the user has visited the ad, he/she would be able to make the decision immediately or after just a few more clicks unless he/she is very indecisive! Therefore, a fraudster would make hundreds or even thousands of clicks on the same ad and also during one particular time of the day compared to a handful by a user, and these could be a clear distinction for the model to pick up.

New features**: ip_hour_count, ip-app-combination**

For the first feature, I grouped the data based on the hour, and divided the number of clicks by the number of unique IP addresses.

The second feature is derived by grouping the data based on the app and then dividing the number of clicks by unique IP addresses.

Why did I only consider IP addresses and not device, OS, and channel? This is because a common "click farm" where fraudsters mass produce clicks can be set up with various devices and operating system. In recent years, mobile frauds seem to have evolved to be more sophisticated in order to avoid the countermeasures put up by authorities. They equip themselves with varying phones and tablets models in order to mislead advertisers into thinking that several users are interacting with their advertisements. However, it is not easy for fraudsters to hide or relay their IP addresses because in countries such as China where mobile frauds are prevalent, it is not easy to access services such as VPN or relay servers.

How many clicks for ip-app-os combination and ip-device-os combination?

Similar to the above approach, I have tried to generate two more features considering the app-os combination. How many clicks are from same IP to same app from same OS version? How many clicks from same, from same device-OS combination.

Training and model evaluation

I studied various implementation of this dataset on Kaggle and also tried myself to train this model initially with full dataset on the Kaggle kernel. The first challenge was to load the dataset and then explore it for feature generation. The data load almost took 2/3rd of the memory and the exploration of the data was so computationally intensive that the kernel died many times. Then I decided to reduce the number of samples to around 30 million and then started exploring and generating the features. The dataset occupied more than half of the system memory and generating each new feature was very slow. Then, I chose to stick to the Light GBM approach rather than going first with other boosting algorithms because of its several advantages of fast training speed, low memory usage, better accuracy than other boosting algorithms.

The model's performance is evaluated by its ROC-AUC score on the validation data. The model's parameters were as followed:

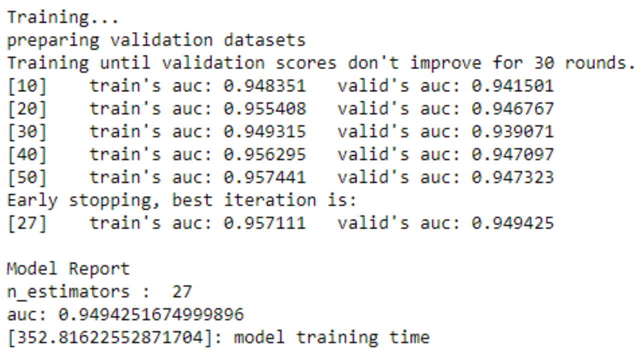
- Objective function: binary
- Metrics: AUC
- Boosting type: gdbt; traditional gradient boosting decision tree ((default_value)
- 'learning_rate': 0.01 (default_value)
- 'num_leaves': 31; Maximum tree leaves for base learners.
- 'max_depth': -1 ; (default_value) -1 means no limit
- 'min_child_samples': 20 (default_value) ; Minimum number of data needed in a child
- 'max_bin': 255; max number of bins to bucket the feature values.
- 'subsample': 0.6; Subsample ratio of the training instance.
- 'subsample_freq': 0 (default_value) ; Frequence of subsample
- 'colsample_bytree': 0.3; Subsample ratio of columns when constructing each tree.

- 'min_child_weight': 5; Minimum sum of instance weight (hessian) needed in a child (leaf)
- 'subsample_for_bin': 200000; Number of samples for constructing bins.
- 'min_split_gain': 0 (default_value) ; Minimum loss reduction required to make a further partition on a leaf node of the tree.
- 'reg_alpha': 0 (default_value) ; L1 regularization term on weights.
- 'reg_lambda': 0 (default_value) ; L2 regularization term on weights.
- 'nthread': 4; Number of parallel threads.
- 'verbose': 0; means the eval metric on the eval set is printed at each boosting stage.

IV Results and Conclusion

Model Evaluation and Validation

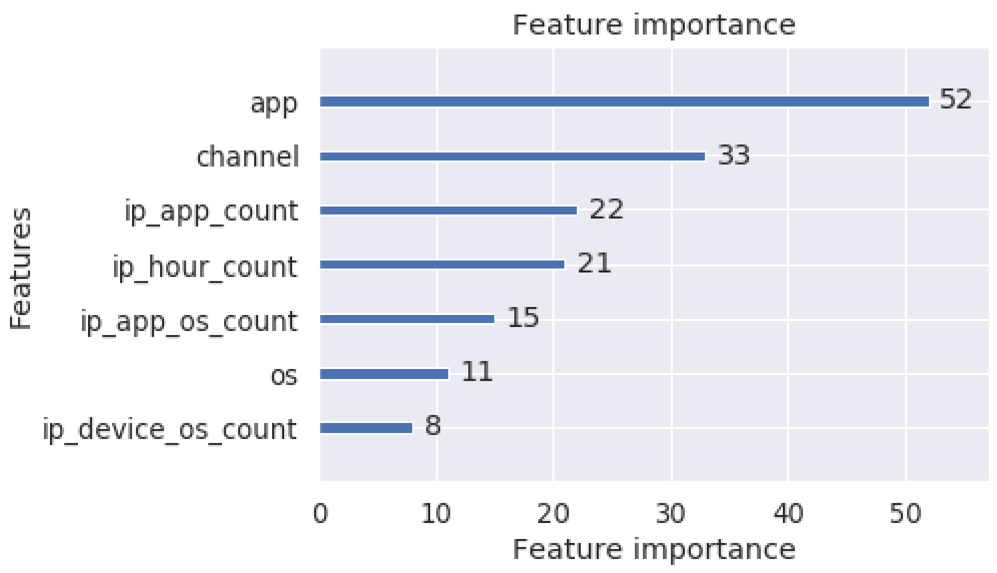
The final model design was built with the tuned hyperparameters as mentioned abov e, using Kaggle Kernel, trained on 30,000,000 data points with `ip_hour_count`,`ip_ap p_count`, `ip_app_os_count`, `ip_device_os_count`, `click_day` and `click_hour` f eatures along with other categorical features.
Score obtained on the validation set: 0.94295.



```
Training...
preparing validation datasets
Training until validation scores don't improve for 30 rounds.
[10]    train's auc: 0.948351   valid's auc: 0.941501
[20]    train's auc: 0.955408   valid's auc: 0.946767
[30]    train's auc: 0.949315   valid's auc: 0.939071
[40]    train's auc: 0.956295   valid's auc: 0.947097
[50]    train's auc: 0.957441   valid's auc: 0.947323
Early stopping, best iteration is:
[27]    train's auc: 0.957111   valid's auc: 0.949425

Model Report
n_estimators :  27
auc: 0.9494251674999896
[352.81622552871704]: model training time
```

I have also analysed the importance of the features fed to the training model, the feat ure importance is shown in the following figure:



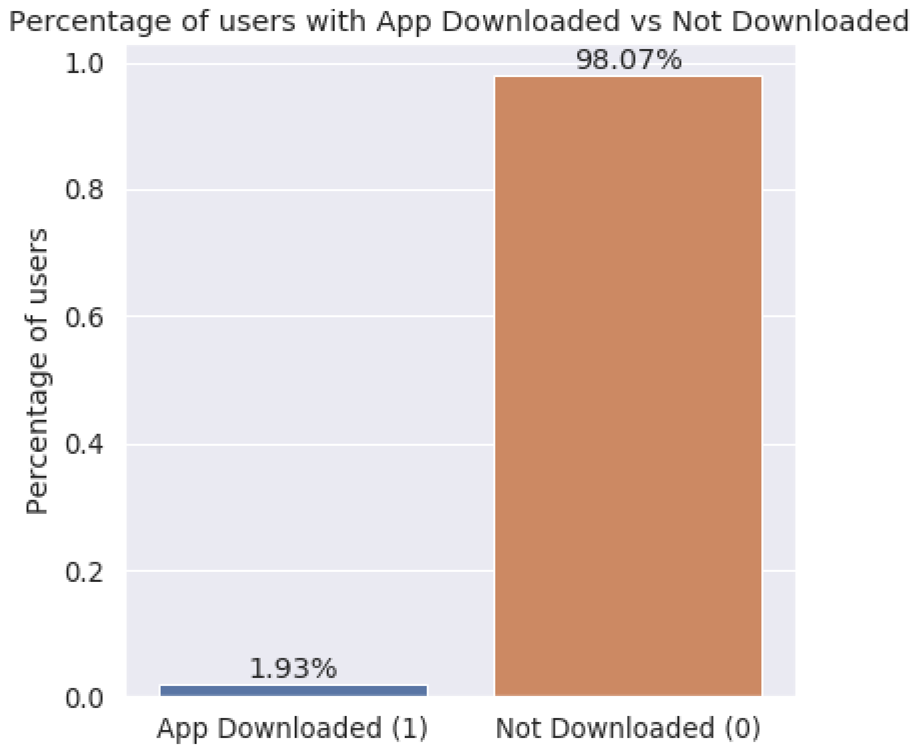
The newly added features ip_app_count and ip_hour_count actually was useful to predict the probability of the click leading to install. If these features were used on the whole data set instead of only 30 million records, the validation accuracy would have been much higher.
The prediction was done on around 7.5 million samples, the result of which shows that approximately 1.93% clicks will lead to actual installation of the app. The result could have improved if trained with all the data.

The final model design with Light GBM trained on 30 million samples of the data with additional features scored 0.9494 on the validation data, which is way above the score obtained with random guessing of 0.3523. This means that the final model performed exceptionally well with a training score of 0.9571 in comparison to the benchmark model. This performance could have been improved if memory and computation constraints were not there.

In its comparison to other Kaggle competitions also, I can say these results are encouraging enough to motivate me to work harder to achieve better performance. This also proved that my way of approaching towards the problem was in the right direction, although there is a lot to improve upon by including more features and training with larger samples.

V. Conclusion

This project turned out be a good learning experience to work with big data utilizing the available computational resources. The data available from Kaggle was clean, no missing data, properly processed, target set clearly defined. The features available were enough to generate new features that can be fed to a model for training considering the domain knowledge of the problem`. The challenge was to generate features for all of the training data available and then train it using a model. I started the feature engineering part with the whole dataset initially. But the RAM and CPU resources were enough to handle such a gigantic size of the data. Therefore, in order to fully optimize the computing power and resources at hand, I decided to reduce the dataset to only 30 million training records and around 3.5 million test samples. The test set performance shows that out of the 3.5 million test users, around 1.93% will install the app.



Improvement

There is so much that can be improved upon for this project, for example the same model design training on the whole dataset with all generated features present would no doubt yield a much higher performance.

A lot of additional features can be generated to test the performance of the dataset which I was not able to do because of limited resources.