Machine Learning Engineer Nanodegree

Capstone Proposal

Mostafa Osama

May 05th, 2019

Definition

Project Overview

Mobile ad fraud is one of the biggest challenges the mobile marketing industry is currently facing. Fraud risk is everywhere, but for companies that advertise online, click fraud can happen at an overwhelming volume, resulting in misleading click data and wasted money. Ad channels can drive up costs by simply clicking on the ad at a large scale. With over 1 billion smart mobile devices in active use every month.

A new report from Juniper Research has found that advertisers will lose an estimated $19 billion to fraudulent activities next year, equivalent to $51 million per day.

The model used in the project is eXtreme Gradient Boosting, the two reason to use XGBoost are the goal of the project:

1- Execution Speed
2- Model Performance

Problem Statement

TalkingData, China's largest independent big data service platform, covers over 70% of active mobile devices nationwide. They handle 3 billion clicks per day, of which 90% are potentially fraudulent. Their current approach to prevent click fraud for app developers is to measure the journey of a user's click across their portfolio, and flag IP addresses who produce lots of clicks, but never end up installing apps. With this information, they've built an IP blacklist and device blacklist.

In this Kaggle Competition we're challenged to build an algorithm that predicts whether a user will download an app after clicking a mobile app ad. To support your modeling, they have provided a generous dataset covering approximately 200 million clicks over 4 days!

To solve this challenge, firstly I attained some basic understanding of the training data that was provided to check for characteristics such as target concept balance and distribution/importance of features.

Metrics

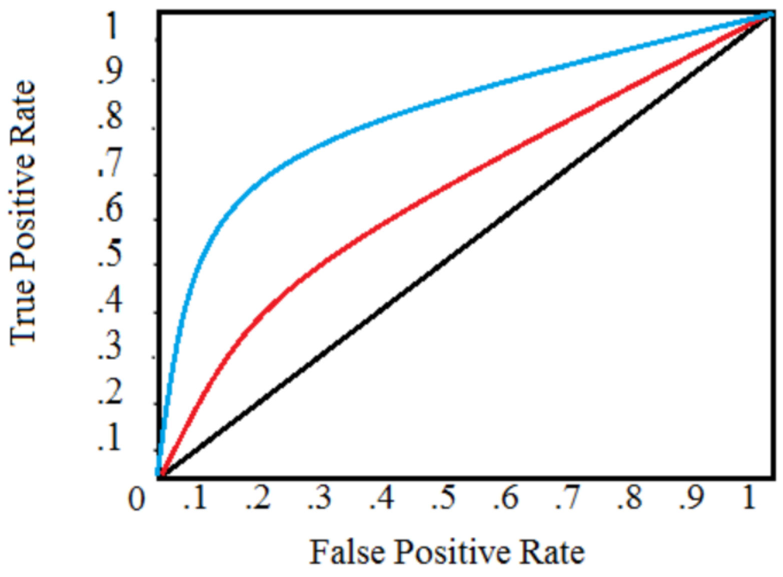
Evaluation done in the Kaggle competition based on the *area under the ROC* – Receiving Operating Characteristics – curve between the predicted probability and the observed target.

A receiver operating characteristic curve, or ROC curve, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

$$true\ positive\ rate = \frac{true\ positives}{true\ positives + false\ negatives} \qquad false\ positive\ rate = \frac{false\ positives}{false\ positives + true\ negatives}$$

A typical ROC curve is shown below:



Analysis

Data Exploration

The dataset provided by Talking Data on Kaggle competition includes approximately 200 million registered clicks over 4 days, split into training and testing sets. The training set contains more than 180 million rows of data, each has the timestamp of the click, number-encoded IP addresses, device numbered label code, device's operating system code, app code, channel code, whether the click resulted in a download or not, and time of download if applicable. The testing set contains about 18 million clicks with each click associated with an ID and other information excluding the download or not label and download time.

Files used:

- *train_sample.csv*, 100,000 randomly-selected rows of training data, to inspect data before downloading full se
- *test.csv*, the testing data

Data Fields:

Each row of the training data contains a click record, with 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, huawei mate 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

Note that **ip**, **app**, **device**, **os**, and **channel** are encoded.

The test data is similar, with the following differences:

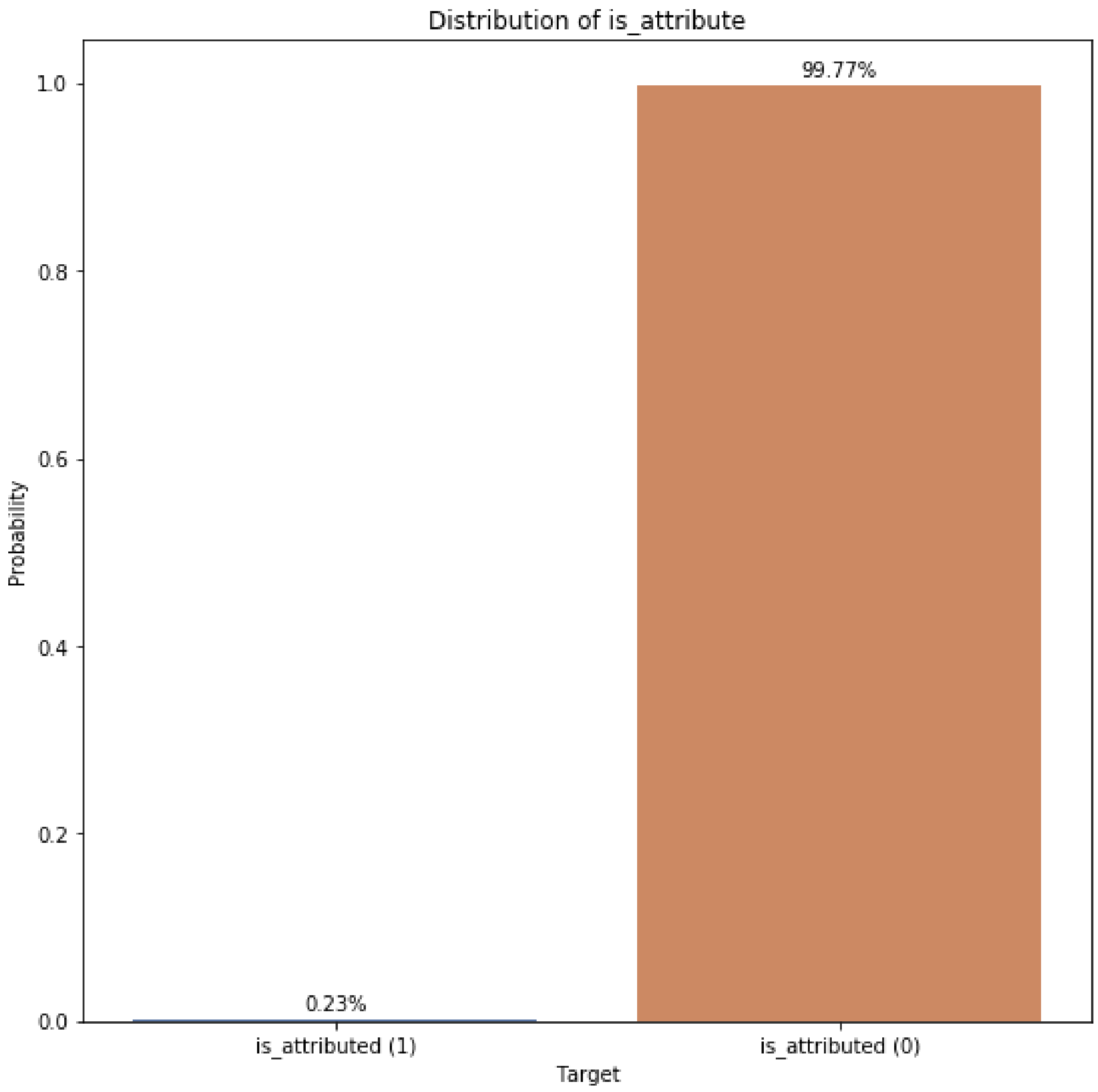
- **click_id**: reference for making predictions
- **is_attributed**: not included

Observation

The data is strongly imbalanced with only around 0.23% positive – clicks that resulted downloads.

Exploratory Visualization

The graph below illustrates how skewed the target concept is in the sample training data set;



Algorithms and Techniques

XGBoost - eXtreme Gradient Boosting - is one of the most popular machine learning algorithm these days. Regardless of the type of prediction task at hand; regression or classification.

XGBoost is well known to provide better solutions than other machine learning algorithms. In fact, since its inception, it has become the "state-of-the-art" machine learning algorithm to deal with structured data.

What makes XGBoost so popular?

- Speed and performance: Originally written in C++, it is comparatively faster than other ensemble classifiers.
- Core algorithm is parallelizable: Because the core XGBoost algorithm is parallelizable, it can harness the power of multi-core computers. It is also parallelizable onto GPU's and across networks of computers making it feasible to train on very large datasets as well.
- Consistently outperforms other algorithm methods: It has shown better performance on a variety of machine learning benchmark datasets.
- Wide variety of tuning parameters: XGBoost internally has parameters for cross-validation, regularization, user-defined objective functions, missing values, tree parameters, and scikit-learn compatible API etc.

Benchmark

From the public leaderboard the winner has Area under Receiving Operating Characteristics Curve score equal to 0.98349 – very amazing score.

Methodology

Data Preprocessing

As previously mentioned, 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.

Before being feed to the model, the time attributes need to be converted to a format that XGBoost can understand – year, month and day.

```
def dataPreProcessTime(df):
    df['click_time'] = pd.to_datetime(df['click_time']).dt.date
    df['click_time'] = df['click_time'].apply(lambda x: x.strftime('%Y%m%d')).astype(int)

    return df
```

Implementation

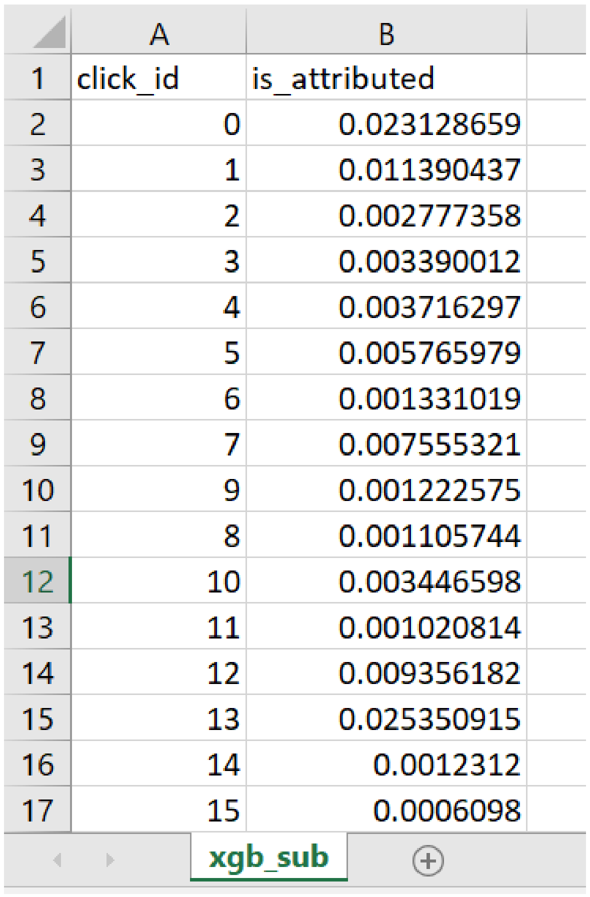
The software requirements for the implementation are as followed;

- Python => '3.6'
- Scikit-learn => '0.20.2'
- Pandas => '0.23.4'
- XGBoost => '0.82'
- Numpy => '1.16.0'
- Matplotlib => '3.0.2'

Parameters tuning used in XGBoost are as followed;

- eta – learning_rate: Step size shrinkage used in update to prevents overfitting
- max_depth: Maximum depth of a tree
- subsample: Subsample ratio of the training instances
- colsample_bytree: is the subsample ratio of columns when constructing each tree. Subsampling occurs once for every tree constructed
- colsample_bylevel: is the subsample ratio of columns for each level. Subsampling occurs once for every new depth level reached in a tree. Columns are subsampled from the set of columns chosen for the current tree
- min_child_weight: Minimum sum of instance weight (hessian) needed in a child
- alpha: L1 regularization term on weights. Increasing this value will make model more conservative
- objective [*binary:logistic*]: logistic regression for binary classification, output probability
- eval_metric [*auc*]: Evaluation metrics for validation data
- scale_pos_weight: Control the balance of positive and negative weights, useful for unbalanced classes

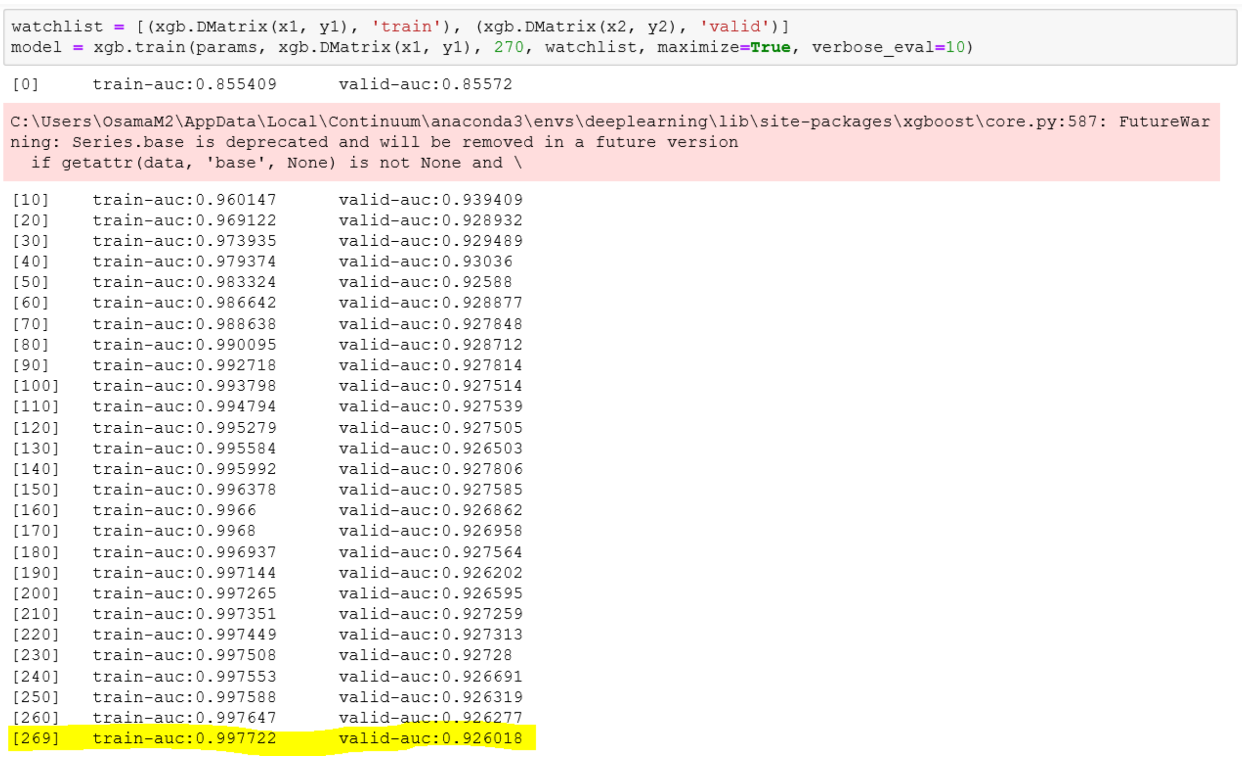
Sample of the output file as below



	A	B
1	click_id	is_attributed
2	0	0.023128659
3	1	0.011390437
4	2	0.002777358
5	3	0.003390012
6	4	0.003716297
7	5	0.005765979
8	6	0.001331019
9	7	0.007555321
10	9	0.001222575
11	8	0.001105744
12	10	0.003446598
13	11	0.001020814
14	12	0.009356182
15	13	0.025350915
16	14	0.0012312
17	15	0.0006098

Refinement

Data Matrix used in XGBoost. DMatrix is an internal data structure that used by XGBoost which is optimized for both memory efficiency and training speed.



```
watchlist = [(xgb.DMatrix(x1, y1), 'train'), (xgb.DMatrix(x2, y2), 'valid')]
model = xgb.train(params, xgb.DMatrix(x1, y1), 270, watchlist, maximize=True, verbose_eval=10)
```

```
[0]     train-auc:0.855409      valid-auc:0.85572
```

```
C:\Users\OsamaM2\AppData\Local\Continuum\anaconda3\envs\deeplearning\lib\site-packages\xgboost\core.py:587: FutureWar
ning: Series.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \
```

```
[10]    train-auc:0.960147      valid-auc:0.939409
[20]    train-auc:0.969122      valid-auc:0.928932
[30]    train-auc:0.973935      valid-auc:0.929489
[40]    train-auc:0.979374      valid-auc:0.93036
[50]    train-auc:0.983324      valid-auc:0.92588
[60]    train-auc:0.986642      valid-auc:0.928877
[70]    train-auc:0.988638      valid-auc:0.927848
[80]    train-auc:0.990095      valid-auc:0.928712
[90]    train-auc:0.992718      valid-auc:0.927814
[100]   train-auc:0.993798      valid-auc:0.927514
[110]   train-auc:0.994794      valid-auc:0.927539
[120]   train-auc:0.995279      valid-auc:0.927505
[130]   train-auc:0.995584      valid-auc:0.926503
[140]   train-auc:0.995992      valid-auc:0.927806
[150]   train-auc:0.996378      valid-auc:0.927585
[160]   train-auc:0.9966        valid-auc:0.926862
[170]   train-auc:0.9968        valid-auc:0.926958
[180]   train-auc:0.996937      valid-auc:0.927564
[190]   train-auc:0.997144      valid-auc:0.926202
[200]   train-auc:0.997265      valid-auc:0.926595
[210]   train-auc:0.997351      valid-auc:0.927259
[220]   train-auc:0.997449      valid-auc:0.927313
[230]   train-auc:0.997508      valid-auc:0.92728
[240]   train-auc:0.997553      valid-auc:0.926691
[250]   train-auc:0.997588      valid-auc:0.926319
[260]   train-auc:0.997647      valid-auc:0.926277
[269]   train-auc:0.997722      valid-auc:0.926018
```

Model Evaluation and Validation

By using two different approaches to estimate the ROC under the area, and find the best estimator.

For solution_1, we have used parameters tuning based on the guide for selecting parameters in XGBoost, and therefore we have achieved score as approximately as 0.99

For solution_2, we have used GridSearch – is the automatic process to tune, but you can tune them manually with validation and therefore we have achieved score as approximately as 0.81

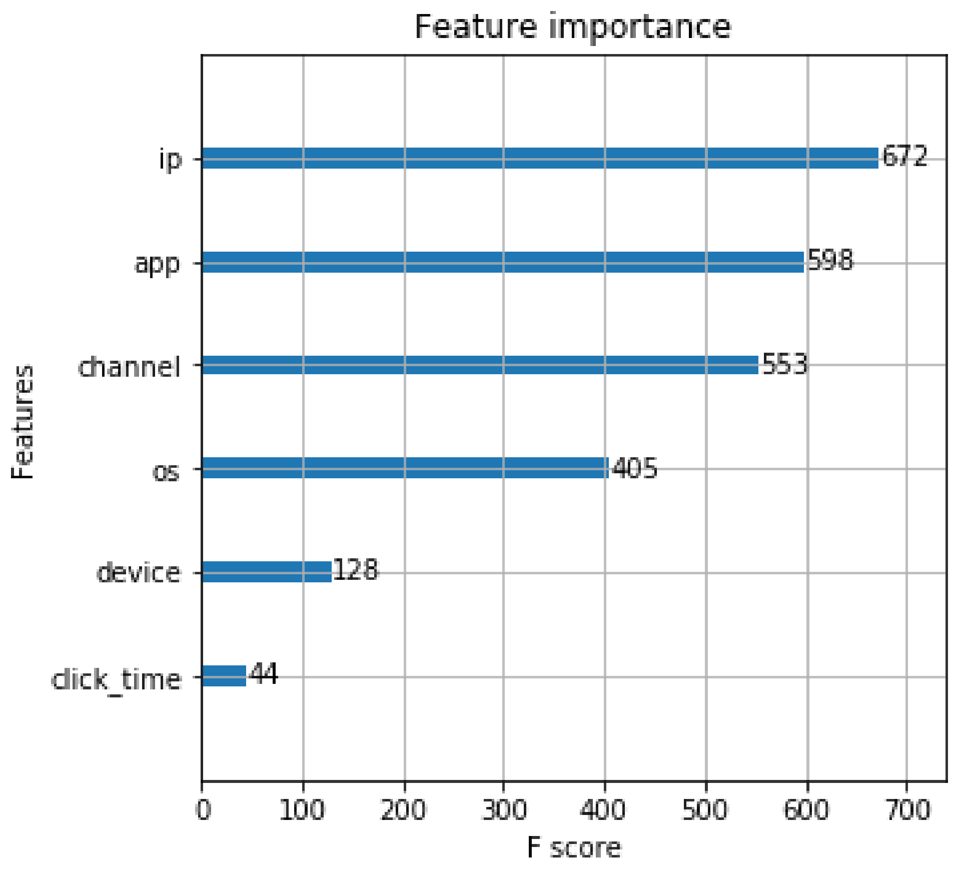
Justification

By not using the whole training dataset – *main reason is the slow internet connection and hardware limitation* – and just used the sample one (100,000 rows). In the boundaries of the competition, I can say these results are encouraging and show that the approach taken is the right direction, although there is so much to improve upon. Getting score around 0.99 is pretty motivated to use the model for the whole datasets.

Conclusion

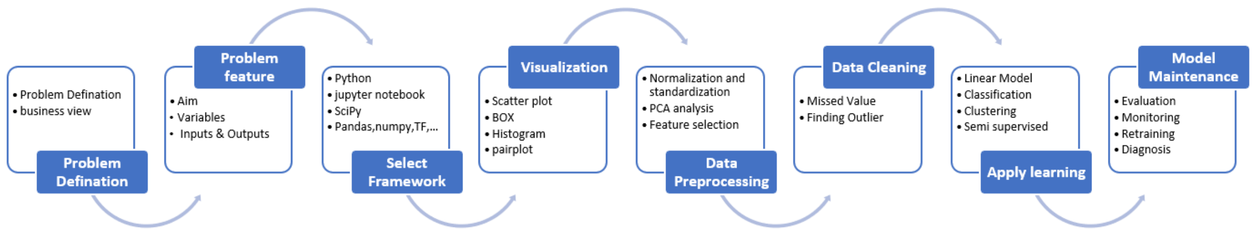
Free-Form Visualization

I fitted the data with an XGBoost classifier and used the plot_importance feature of XGBoost to see how much each feature has an impact on predicting the outcome of a click. The graph of feature importance is given below:



Reflection

You can see my reference workflow in the below image (Reference);



That helped me to understand the end-to-end ML process. The given data from TalkingData was clean, the target concept is clearly defined and the features gave me lots of room for engineering and research.

Data Matrix used in XGBoost, helped me to have optimized for both memory efficiency and training speed.

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.