Machine Learning Engineer Nanodegree

Capstone Project -- TalkingData AdTracking Fraud Detection Challenge

Ju-Ying Chen
chenjyi@tsmc.com

I. Definition

Project Overview

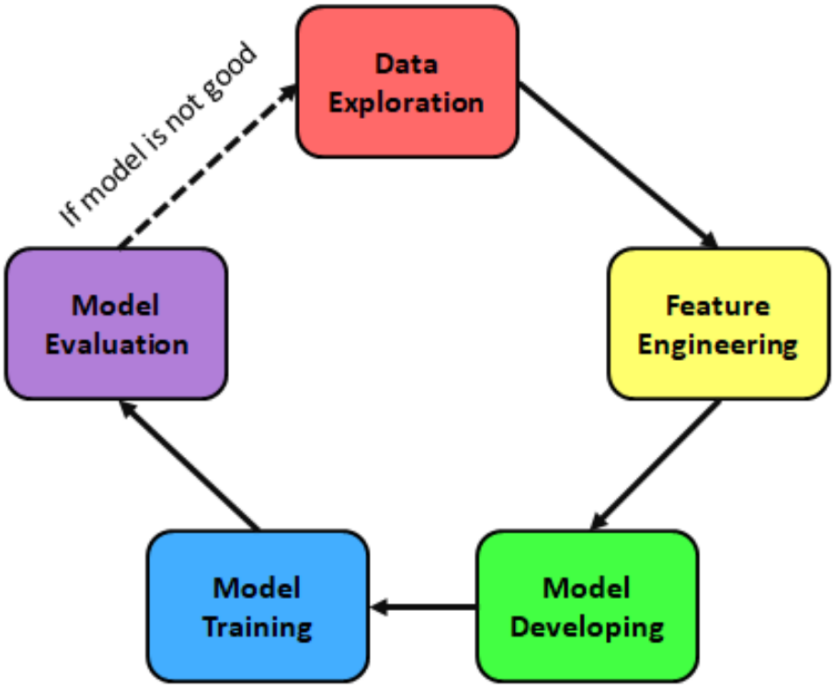
Fraud risk is everywhere, and for companies that advertise online, click fraud can happen at an overwhelming volume, resulting in misleading click data and wasted money. Ad channels can drive up costs by simply clicking on the ad at a large scale. With over 1 billion smart mobile devices in active use every month, China is the largest mobile market in the world and therefore suffers from huge volumes of fraudulent traffic. There is a good research, “Detecting Click Fraud in Online Advertising: A Data Mining Approach”, it summarized lots of observations and analyzed the fraud click detection. It also addressed some important issues in data mining and machine learning research, including highly imbalanced distribution of the output variable, heterogeneous data, and noisy patterns with missing and unknown values.

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 have built an IP blacklist and device blacklist. Kaggle competition provides a generous data set covering approximately 200 million clicks over 4 days. All of the necessary data sets can be found and download from: https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data (https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data)

There are about hundreds million recorded data and each data entry is with 8 features. It contains some features like: IP address of click, App. ID for marketing, operation system (OS) version ID of user mobile phone, device type ID of user mobile phone, the target that is to be predicted, indicating the app was downloaded, etc. Some features of data are encoded already. Except of two features are recorded in UTC time format.

Problem Statement

The problem is simple: **how to know whether a user will really download an app after clicking a mobile app ad? That is, how to distinguish between meaningful clicks and fraud clicks?** Currently, TalkingData does have some methods to prevent click fraud. But there are still rooms for improvement. While successful, they can then always be one step ahead of those fraudsters. So our goal is to **develop an algorithm which can precisely predict whether a user will download an app after clicking a mobile app ad based on the recorded properties of that user**. The performance of model will be evaluated on area under the Receiver operating characteristic (ROC) curve between the predicted probability and the observed target. The smaller differences between our predictions and truths, the better our solution model will be. Then such model can be used to distinguish between meaningful clicks and fraud clicks and reduced the amount of wasted money caused by fraudulence.



By following a very traditional but useful work flow, we first approach this problem by investigating the data. Through this exploratory data analysis, we can establish some basic ideas about the interrelationship between different features or the natural properties of each feature itself. We can even create some new features based on the existing features.

Next, we have to check and clean the data. Maybe sometime our data will contain lots of different values or even missing values. So in feature engineering of data for our model developing, we will handle the problem of missing value and outliers, and/or normalize numeric features. If we have any categorical feature or text format feature, additional data preprocessing techniques will be included. For developing a proper model for our project, we now will split the whole training data set into to three pieces: one for training, another for validation and the other for testing. This step is for cross-validation.

Basically, two kinds of classifiers will be built in this project: one is based on the neural network; the other is based on the random forest. Grid search method will be optional for finding the best combination of model's parameters. Once we have finished training the models. They will be evaluated by using the evaluation metric, area under the Receiver operating characteristic curve (AUC). Models will be check thoroughly to see if there is anything insufficient.

Kaggle's official evaluation will be also taken into account. Depending on the performance of these models, we maybe have to go to some previous step to see if there is anything missing or wrong. Once we have an acceptable model (the one with better performance on evaluation and testing), whose score is at least over 0.92, we can stop and publish that model.

Metrics

Evaluation metric will be the area under the Receiver operating characteristic (ROC) curve between the predicted probability and the observed target. Such metric is also called AUC. AUC as a further interpretation of ROC is a very straightforward and easy-understanding metric of a binary classifier system. Since now we are trying to establish a model to predict whether a user will download an app after clicking a mobile app or not. This is exactly a binary classification problem. Given a threshold parameter T, the instance is classified as "positive" if $X > T$, and "negative" otherwise. X follows a probability density $f_1(x)$ if the instance actually belongs to class "positive", and $f_0(x)$ if otherwise. Therefore, the true positive rate is given by $TPR(T) = \int_T^\infty f_1(x)dx$ and the false positive rate is given by $FPR(T) = \int_T^\infty f_0(x)dx$.

The ROC curve plots parametrically $TPR(T)$ versus $FPR(T)$ with T as the varying parameter. Then the AUC is simply the area under the ROC. Generally, we can judge our model through the value of AUC like follows:

- AUC=0.5 (no discrimination)
- 0.7 ≦ AUC ≦ 0.8 (acceptable discrimination)
- 0.8 ≦ AUC ≦ 0.9 (excellent discrimination)
- 0.9 ≦ AUC ≦ 1.0 (outstanding discrimination)

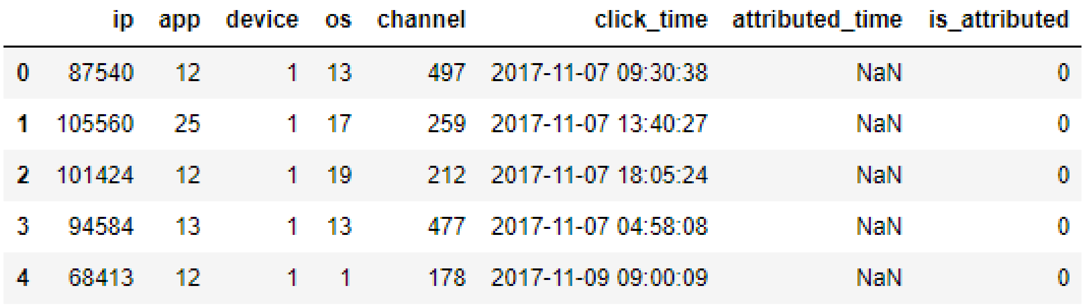
Ref: https://en.wikipedia.org/wiki/Receiver_operating_characteristic (https://en.wikipedia.org/wiki/Receiver_operating_characteristic)

II. Analysis

Data Exploration

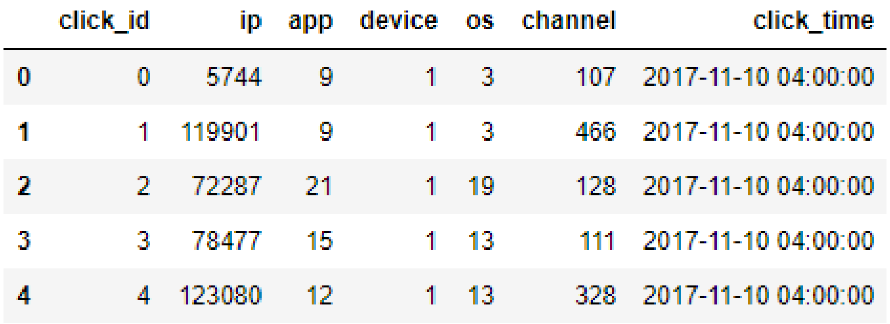
Input training data can be download from: https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data (https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection/data)

- **train.csv** - the training set. A quick view of the table content is shown below:



	ip	app	device	os	channel	click_time	attributed_time	is_attributed
0	87540	12	1	13	497	2017-11-07 09:30:38	NaN	0
1	105560	25	1	17	259	2017-11-07 13:40:27	NaN	0
2	101424	12	1	19	212	2017-11-07 18:05:24	NaN	0
3	94584	13	1	13	477	2017-11-07 04:58:08	NaN	0
4	68413	12	1	1	178	2017-11-09 09:00:09	NaN	0

- **train_sample.csv** - 100,000 randomly-selected rows of training data, to inspect data before downloading full set.
- **test.csv** - the official testing data set for Kaggle Leaderboard ranking



	click_id	ip	app	device	os	channel	click_time
0	0	5744	9	1	3	107	2017-11-10 04:00:00
1	1	119901	9	1	3	466	2017-11-10 04:00:00
2	2	72287	21	1	19	128	2017-11-10 04:00:00
3	3	78477	15	1	13	111	2017-11-10 04:00:00
4	4	123080	12	1	13	328	2017-11-10 04:00:00

The size of the data (train.csv) is really quite big. It maybe takes much time use the whole training data set and the whole testing data set for our model development and self-evaluation due to limited memory. There are **184903890** data entries and each data entry is with **8** features. It contains some features like: IP address of click, App. ID for marketing, operation system (OS) version ID of user mobile phone, device type ID of user mobile phone, the target that is to be predicted, indicating the app was downloaded, etc. Some features of data are encoded already. Except of two features are recorded in UTC time format. There is a data sample (train_sample.csv) provided for quick inspection.

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

Note that ip, app, device, os, and channel are encoded because these are sensitive data. Although we maybe will loss some useful information here.

The test data for Kaggle Leaderboard ranking is similar, with the following differences:

- **click_id**: reference for making predictions

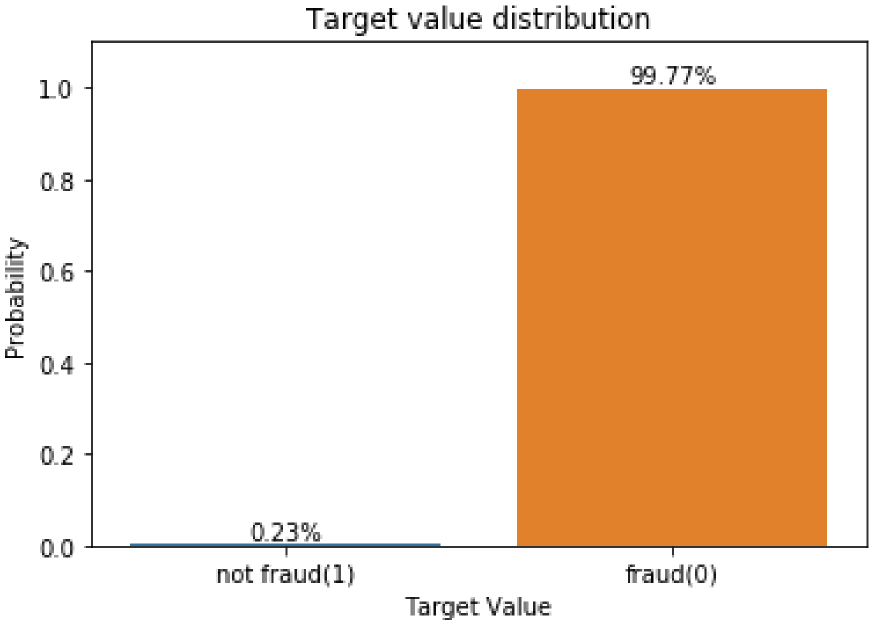
And there is no '**attributed_time**' information in test.csv.

Target Variable

- **is_attributed**: the target that is to be predicted, indicating the app was downloaded (not included in the test data for Kaggle Leaderboard ranking)

Our training data will contain only 'ip', 'app', 'device', 'os', 'channel', 'click_time' and 'is_attributed' or their derivatives.

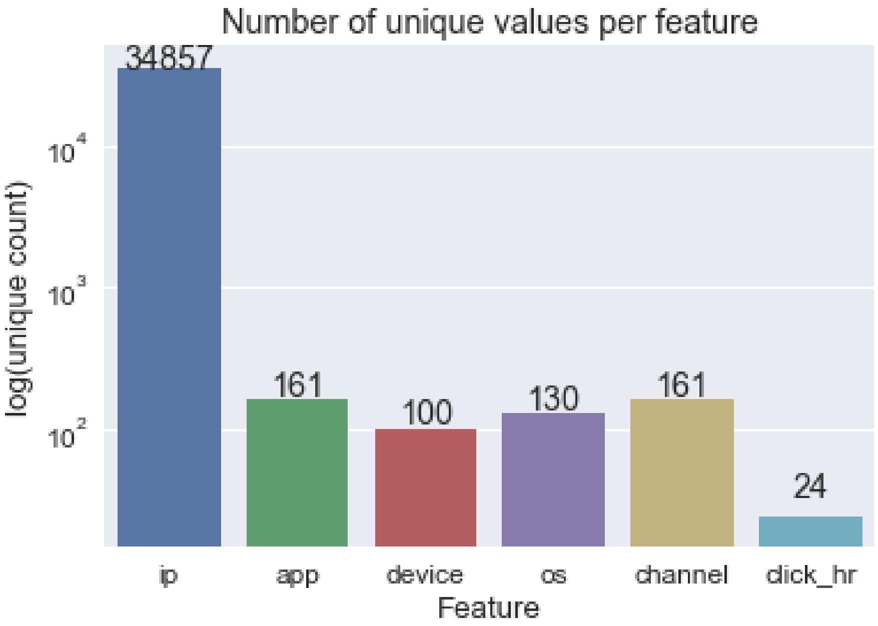
Next, we start to dig into the data by investigating the distributions or properties of different features. Because the size of train.csv is too large, the train_sample.csv, 100,000 randomly-selected rows of training data, will be much more suitable for us to inspect data. For the rest part of this section, all the results are based on this 100,000 data sample.



After a quick check of target value distribution as shown above, we do have an extremely imbalanced dataset. About 99.77% of the dataset are labeled as fraud click and only about 0.23% are not fraud click. Hence, we maybe need to include some data re-sampling strategies for handling this imbalanced label distribution.

The official testing data doesn't contain "attributed_time" feature. So we will drop this feature out from the training data. Currently, we have six features '', Since the original content of the feature "click_time" is recorded in UTC format. It would be more meaningful to round the values of "click_time" down to an hour of the day. Then we can check to see if there is any special pattern of "click_time" and also investigate the correlation between features.

Exploratory Visualization

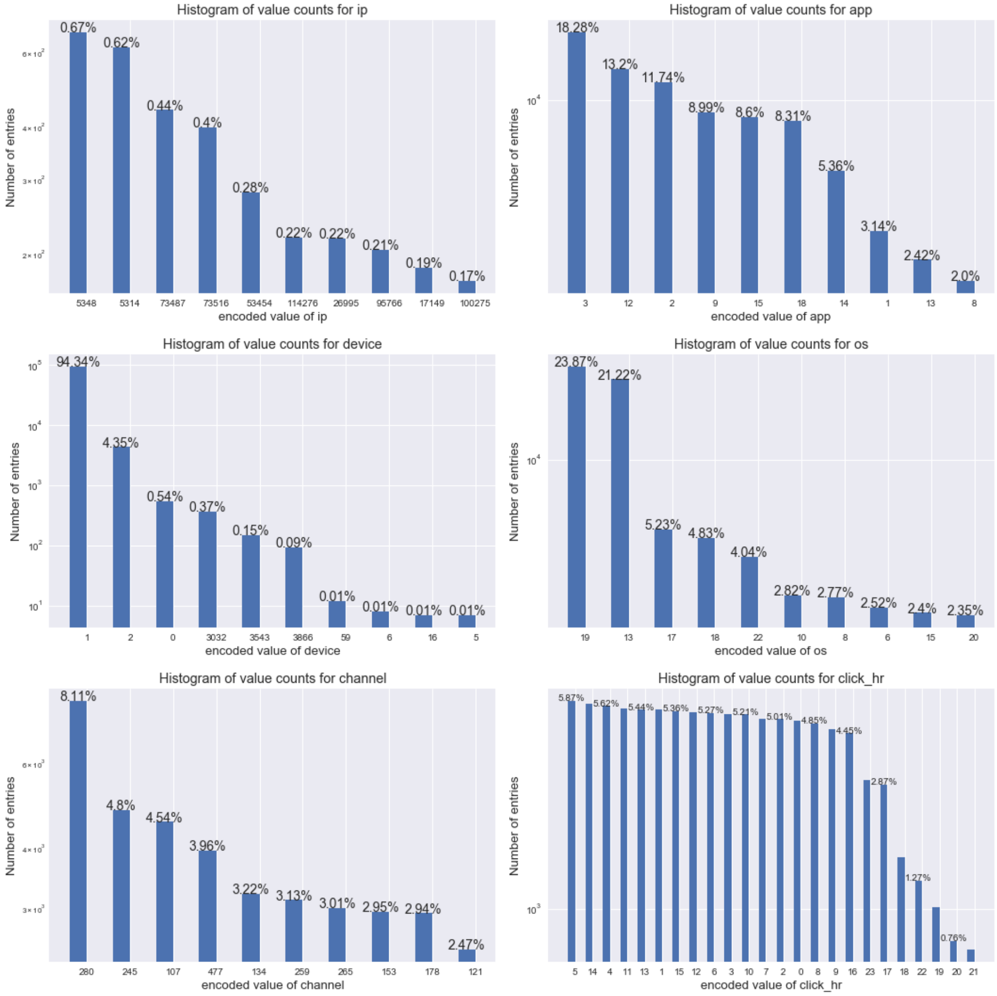


From the above histogram, we can know that the except of the feature 'ip', which has the most categories, 34857, and the feature 'click_hr', which has surely 24 categories, the other features like 'app', 'device', 'os', and 'channel' only have about hundreds or thousands categories. This is quite consistent with our understanding of the meaning of these feature in the real world.

The IP address can be IPv4 or IPv6. An IPv4 address has a size of 32 bits. IPv4 addresses are usually represented in dot-decimal notation, consisting of four decimal numbers, each ranging from 0 to 255, separated by dots, e.g., 172.16.254.1. IPv6 is a more complicated format of IP address in order to provide more addresses. The IP address is unique for eack device which is connected to the Internet. If two devices are connecting to the Internet in the same time, they definitely won't use the same IP address.

The 'app' feature means the mobile applications which users are using to access some kinds of products. From my point of view, not all kinds of mobile applications will access some specific products. So it's quite reasonable that there are only hundreds of them are included here. The 'device' feature is the device type id of user mobile phone (e.g., iphone 6 plus, iphone 7, huawei mate 7, etc.). There are also lots of brands on the market and Each brand normally will sell many kinds of mobiles year by year. The 'os' feature stands for the OS version of user mobile phone. Roughly, there are mainly two kind of operation systems of mobile phone, iOS and Android. There are also branches and subversion of them. Each mobile company will sometime develop their own operation system based on the iOS or Android. The 'channel' feature stands for the channel ID of mobile ad publisher. Maybe there are lots of mobile ad publishers in China. but 200 is a reasonable number of mobile ad publishers that one product are correlated with.

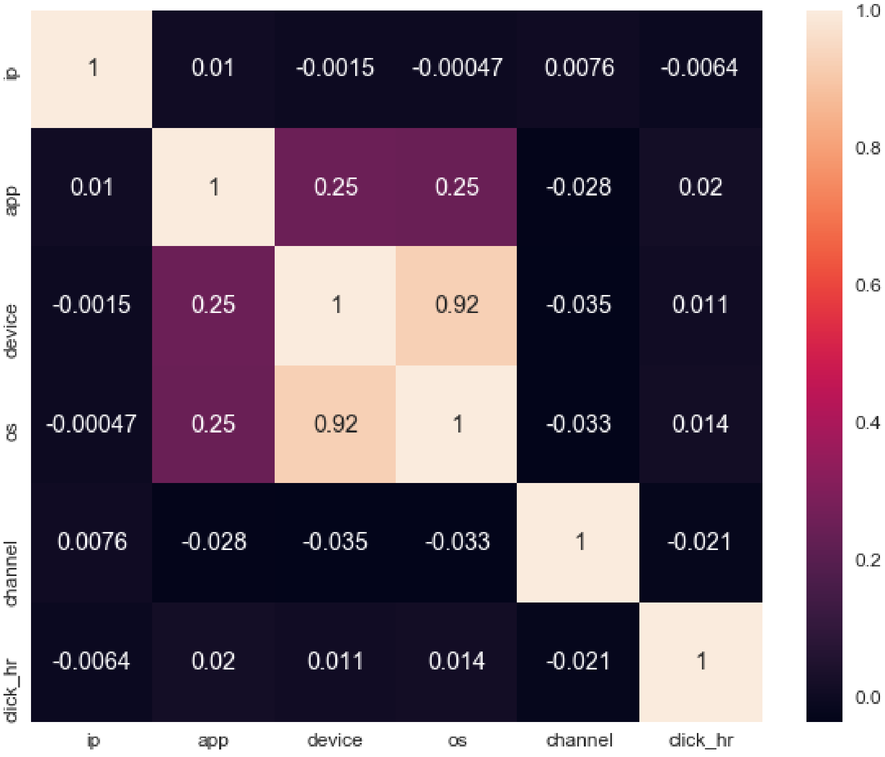
Ref: https://en.wikipedia.org/wiki/IP_address (https://en.wikipedia.org/wiki/IP_address)



The above histograms show the 10 most frequent values of click data w.r.t. discrete categorical features, 'ip', 'app', 'device', 'os' and 'channel', except the lower right one which shows the all click data in each hour of a day. It's quite obvious there are some dominant entries for 'device' and 'os'. For the middle left bar chart of 'device', the first entry is in the majority, which is ~94.34%. By taking the second entry (~4.35%) into account. Maybe this tells us that the market share of some mobile phone are dominant in China. It can be iphone, mi or huawei. For the middle right bar chart of 'os', we can also easily find the first two entries, 23.87% and 21.22%, are relatively higher than the others. This is also quite consistent with my understanding. They can be iOS or some Android system. It's also quite possible that these two features, 'device' and 'os', are correlated with each other since the type of operation system is often determined by the mobile phone. For example, you can only use iOS on iphone and Android on the other mobile phones like mi, sony, huawei, etc.

For the lower right bar chart of 'click_hr', we can find the distribution of click data is quite even among every hour of a day except those hours of evening. There is no precise definition for evening in terms of clock time, but it is considered to start around 5 p.m. - 6 p.m and to last until nighttime or bedtime. This is quite strange for me because originally I think people will have more time to use mobile when they get off duty. This is a little counterintuitive for me. Maybe people have less time to use their mobile because they have to be with their family or do other things like sports in the evening. And they often feel bored when they are working so they like to use their mobile in the daytime. It should be noted that we only show the percentage of odd bins due to the limited space of bar chart.

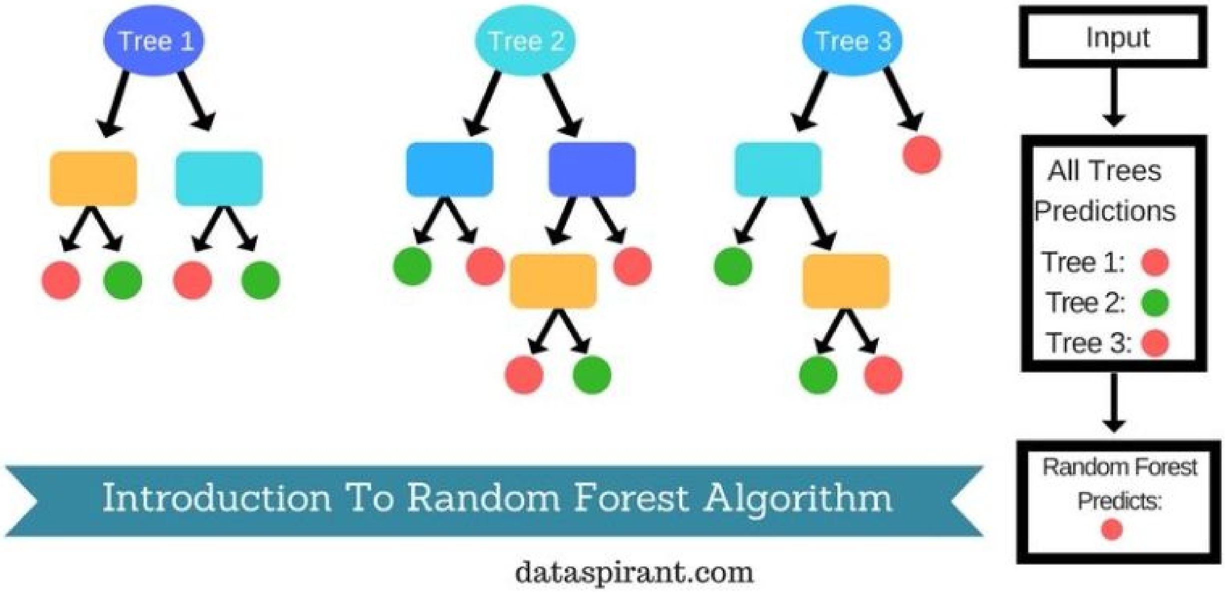
Ref: https://en.wikipedia.org/wiki/Evening (https://en.wikipedia.org/wiki/Evening)



From the above heat map, we can clearly figure out the correlations between different features. Let's say the correlation between any two features is high if the value of the corresponding block is above 0.5. Then we can maybe claim that we do find some pairs of features which exhibit some degree of correlation. It's obvious that features 'device' and 'os' are highly correlated with each other as we have expected before. But it's also quite interesting that there are very small correlation between the feature 'app', 'os' and 'device'. This can be explained by that there are some apps which are OS or device exclusive limited version.

Algorithms and Techniques

Basically, two kinds of classifiers will be built: one is based on the '**Random Forest**'; the other is based on the '**Neural Network**'. The former algorithm, 'Random Forest', is an ensemble learning algorithm. The basic idea of ensemble is quite straightforward. The combination of learning models (learners) increases the classification accuracy and makes itself more robust. For 'Random Forest', the learner is a 'Decision Tree'. 'Decision Tree' is very easy to understand and interpret when there are few decisions and outcomes included in the tree. Generally, it also make lower cost of calculation and can handle both continuous values or categorical variables of data. But 'Decision Tree' tends to overfit the data, that is, it doesn't generally learn something from the data but just only memorize them. It somehow can make terrible prediction on those unseen data (testing). Through combining each individual tree into a 'forest', we take all the contributions for each tree into account like 'averaging/voting their results'. The variance among these trees decreases when compared to a single decision tree. And we can then have an overall better model. Besides, such tree-based algorithm can handle either numerical or categorical data as input quite well. The following image shows the process of 'Random Forest':



(picture credit: DataSpriant)

The following hyper-parameters will/can be tuned to optimize the Random Forest classifier:

- The number of trees in the forest.
- The number of features to consider when looking for the best split.
- The maximum depth of the tree.
- The minimum number of samples required to be at a leaf node.

It should be noted that I am not going to change the parameter 'criterion'. I will use the default setting of this parameter. That means the basic tree learner would be based on CART (classification and regression tree) algorithm and Gini impurity.

The latter algorithm, 'neural network', or so-called Multi-layer Perceptrons (MLPs), is also a very primary algorithm for solving a linear or non-linear classification problem. 2-3 fully connected hidden layers will be come after the input layer. Then the output layer will be passed through a sigmoid function for converting the output values as probabilities. Furthermore, the following hyper-parameters can be tuned to find the best combination for our Neural Network classifier:

- The number of neurons of each hidden layer
- The activation function of each hidden layer
- The optimization algorithm used for training the neural network
- The step-size in updating the weights

And due to the large size of our sample data and also the full data, mini-batch gradient descent method would be our choice when we train our neural network classifier.

Ref.: https://en.wikipedia.org/wiki/Random_forest (https://en.wikipedia.org/wiki/Random_forest)
Ref.: http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/ (http://dataaspirant.com/2017/05/22/random-forest-algorithm-machine-learing/)
Ref.: http://scikit-learn.org/stable/modules/neural_networks_supervised.html (http://scikit-learn.org/stable/modules/neural_networks_supervised.html)

Besides, a useful technique '**grid search**' will be included in this analysis for finding the best combinations of hyper-parameters of a model. The hyper-parameters cannot be directly learned from the regular training process but can only be fixed before the actual training process begins.

As mentioned before, our dataset was extremely imbalanced. Only very few fraction of data are not labeled as fraudulent. Such imbalanced distribution of data can possibly induce the biased prediction of model. For example, the data of majority classes can take much more attentions when we try to training a classifier. This is known as frequency bias. Many machine learning algorithms place tend to put more emphasis on learning from data observations which occur more commonly. **Resampling strategies** such as over/undersampling are necessary here. Three kinds of data samples will respectively be used for training our random forest classifier and neural network classifier:

- The original data sample
- New data sample generated by oversampling algorithm, Synthetic Minority Oversampling Technique (SMOTE). The SMOTE generates new observations by interpolating between observations in the original dataset.
- New data sample generated by undersampling algorithm, 'Tomek's links'. This kind of undersampling method is more like a cleaning under-sampling techniques. It will implement an heuristic which will clean the dataset based on Tomek's links. A Tomek's link between two samples of different class x and y is defined such that there is no example z as follows: $d(x, y) < d(x, z)$ or $d(y, z) < d(x, y)$, where d() stands for the distance between the samples and x and y are from different class respectively. That is, a Tomek's link only exists when the two samples are the closest neighbors of each other. When we find a pair of samples with a Tomek's link, the default setting of this algorithm will remove the one from the majority class.

It should be noted that this analysis focuses on finding the best strategy for establishing a most powerful or at least a not-that-bad classifier with the tools and algorithms we are going to try to use and implement. We are not going to find the best oversampling or undersampling method but will only try one of oversampling and undersampling respectively. There are still many resampling methods for further study but will be not include in this study.

Validation results produced by different classifier-sample combinations will be compared to each other and help us to determine the most suitable one for our analysis.

Ref.: https://www.jeremyjordan.me/imbalanced-data/ (https://www.jeremyjordan.me/imbalanced-data/)
Ref.: http://contrib.scikit-learn.org/imbalanced-learn/stable/index.html (http://contrib.scikit-learn.org/imbalanced-learn/stable/index.html)

Benchmark

This project is actually taken from one of Kaggle competitions. They provide a benchmark model which is developed by a random forest method. The score of this benchmark model is 0.911. And the score is evaluated on area under the Receiver operating characteristic (ROC) curve between the predicted probability and the observed target.

III. Methodology

Data Preprocessing

Since the original full data is too large, the data sample '**train_sample.csv**' will be used for training and validation/testing in this analysis. And the final score will be evaluated by using the full data in order to compare our results with the official benchmark model. Before implementing the learning algorithms, several steps for preprocessing our sample data are necessary and listed below:

- Rounding the values of 'click_time' down to an hour of the day, '**click_hr**'. (This step had been done in the section of 'Data Exploration')
- The order data sample will be randomized/shuffled before splitting it into a training set and a validation set. The proportion of the dataset to include in the train split is 80%. The rest part will be included in our validation/testing set. It should be noted that the training and testing sets hold the same target value distribution as the original data sample.
- Generating two kinds of new data samples based on the original data sample:
  - oversampling data produced by SMOTE.
  - undersampling data produced by 'Tomek's link' method.

Implementation

Two kinds of algorithms for solving classification problem will be implemented here:

- random forest ensemble method
- multi-layer perceptron (neural network)

The implementation is basically done in 3 steps:

1. Create a pipeline function to execute the model with different samples and record the metrics.
2. Create models based on the above two algorithms respectively and pass them to the pipeline function we established in step 1 for training and testing.
3. Visualizing the metrics for evaluating performances of different models.

As mentioned before, the performance of a model is basically evaluated by using AUC (the area under the Receiver operating characteristic (ROC) curve between the predicted probability and the observed target). The value of AUC varies from 0 to 1. Our goal here is to chasing a model with higher AUC value. Although there are some potential misleading problems when people evaluate the model performance totally based on AUC. But we are not going to focus on this part. They maybe can be some further studies.

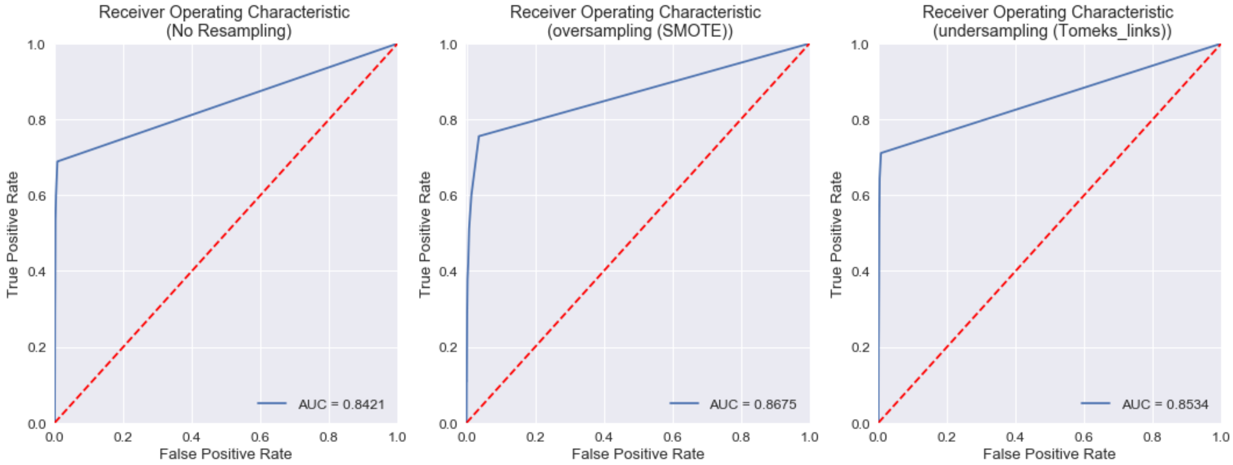
Ref: https://academic.oup.com/bioinformatics/article/26/6/822/244957 (https://academic.oup.com/bioinformatics/article/26/6/822/244957)
Ref: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1466-8238.2007.00358.x (https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1466-8238.2007.00358.x)

The prototype of our random forest model is established directly by using the module 'RandomForestClassifier' implemented in the widely-used free machine learning library sklearn. In this prototype model, the values of selected tunable hyper-parameters are followed by using default setting:

- The number of trees in the forest, `n_estimators=10`

- The number of features to consider when looking for the best split, `max_features=sqrt(n_features)`
- The maximum depth of the tree, `max_depth=None`. This means nodes are expanded until all leaves are pure or until all leaves contain less than 2.
- The minimum number of samples required to be at a leaf node, `min_samples_leaf=1`

Ref.: http://scikit-learn.org/stable/ (http://scikit-learn.org/stable/)

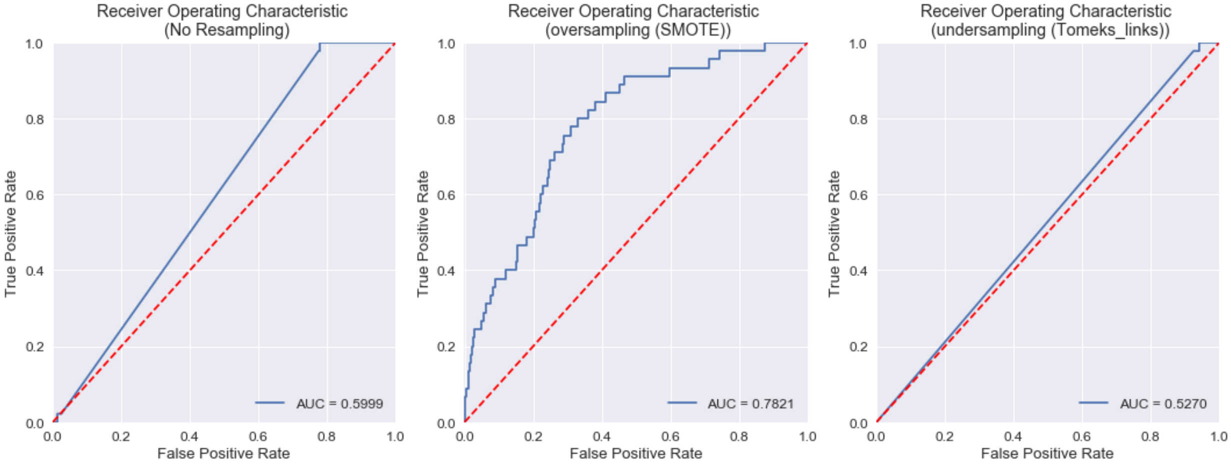


From the above visualization of modeling results, which are evaluated by using testing set of data sample, the random forest models trained by using different training data samples can almost reach the same accuracy, ~99.7%. The AUC scores are also quite good, 0.84 ~ 0.87, but not enough to be compared with the benchmark one, which has 0.911 AUC score. And there are still rooms for improvements.

Now let's turn to the neural network. The prototype of our MLP classifier is established by using the module 'MLPClassifier' implemented in sklearn library. As mentioned before, we will only build a neural network with 2 to 3 hidden layers. For MLP classifier, the values of selected tunable hyper-parameters are followed by using the following setting:

- The number of neurons of each hidden layer, 64 neurons for first hidden layer and 32 neurons for the second hidden layer.
- The activation function of each hidden layer, `activation='relu'`, the Rectified Linear Unit (ReLu).
- The optimization algorithm used for training the neural network, `solver='adam'`, the Adaptive Moment Estimation method (Adam).
- The step-size in updating the weights, `learning_rate_init=0.001`. A constant learning rate is adopted here.
- The size of mini-batches of training data for optimization, `batch_size=64`
- The maximum number of iterations, `max_iter=2`.

Ref: https://en.wikipedia.org/wiki/Rectifier_(neural_networks (https://en.wikipedia.org/wiki/Rectifier_(neural_networks))
Ref: https://arxiv.org/abs/1412.6980 (https://arxiv.org/abs/1412.6980)



From the above visualization of modeling results, which are also evaluated by using testing set of data sample, the MLP classifiers trained by using different training data samples show a similar accuracy, ~99.7% for ordinary sample and new sample produced by undersampling. The one trained with oversampling data set shows less accuracy ~94.2%. The AUC scores of different MLP classifiers vary from ~0.52 to ~0.78. The classifier trained with oversampling data set perform relatively better than the other two with ~0.78 AUC score. The one trained with undersampling data set only give a not-that-good prediction, ~0.53 AUC score, which is only a little bit better than random guessing. Of course, none of them can be good enough to be compared with the benchmark one, which has 0.911 AUC score. It's quite obvious that there are also still rooms for improving our MLP classifier(s).

We can also have a quick conclusion of the resampling strategy. There is no significant difference contributed by using resampling samples for training a random forest classifier with current setting. But when we using oversampling data set to train our MLP classifier, it shows such there is 10~15% improvement in predicting the testing split of the original data sample. So from now on, in this analysis, the random forest classifier will always be trained without taking resampling into account. And the oversampling method will be adopted as the most proper treatment of the extremely imbalanced data for training a random forest MLP classifier.

Refinement

The GridSearchCV technique implemented in the library `sklearn` uses an estimator and a set of hyper-parameters to exhaustively generates model candidates and also evaluates them with proper/user-defined metric score. Hence, user can know the most optimal combination of hyper-parameters systematically. Two kinds of algorithms in this analysis will be tuned separately.

For our random forest classifier, a searching grid is built as follows:

hyper-parameter	testing values
n_estimators	10, 15, 20, 50, 100
max_features (ratio)	0.3, 0.5, 0.7
max_depth	None, 6, 10, 12
min_samples_leaf	1, 10, 20, 60

On the other hand, the best combination of our MLP classifier's hyper parameter will be searched according to the following table:

hyper-parameter	testing values
solver	'lbfgs', 'sgd', 'adam'
activation	'logistic', 'tanh', 'relu'
hidden_layer_sizes	(16, 16), (32, 32), (64, 64), (128, 128)
learning_rate_init	0.001-0.01, step sizz=0.001

For the solver of optimization algorithm, the 'lbfgs' stands for 'Limited-memory BFGS' while 'sgd' is the abbreviation of 'Stochastic gradient descent'. The 'logistic' function of the hyper-parameter 'activation' is also known as the Sigmoid function.

Ref.: https://en.wikipedia.org/wiki/Limited-memory_BFGS (https://en.wikipedia.org/wiki/Limited-memory_BFGS)
Ref.: https://en.wikipedia.org/wiki/Stochastic_gradient_descent (https://en.wikipedia.org/wiki/Stochastic_gradient_descent)
Ref.: https://en.wikipedia.org/wiki/Activation_function (https://en.wikipedia.org/wiki/Activation_function)

The process of tuning hyper-parameters of a model can be extremely time-consuming. It depends on how many possible values they can be. The following table shows the results of refinement:

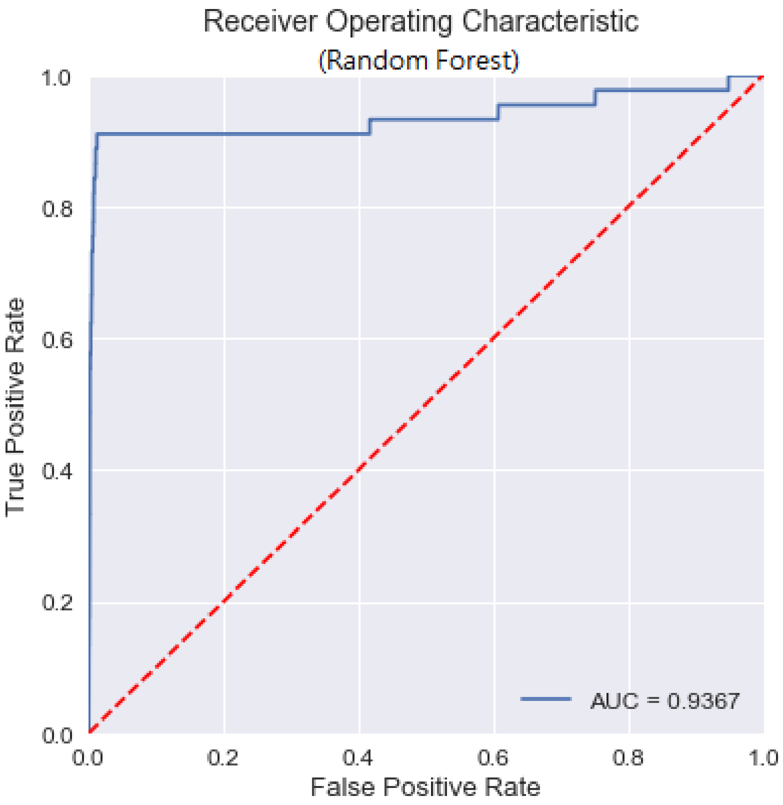
Random Forest	Multi-layer Perceptrons
n_estimators = 50	solver = 'adam'
max_features = 0.3	activation = 'relu'
max_depth = 12	hidden_layer_sizes = (128, 128)
min_samples_leaf = 20	learning_rate_init = 0.005

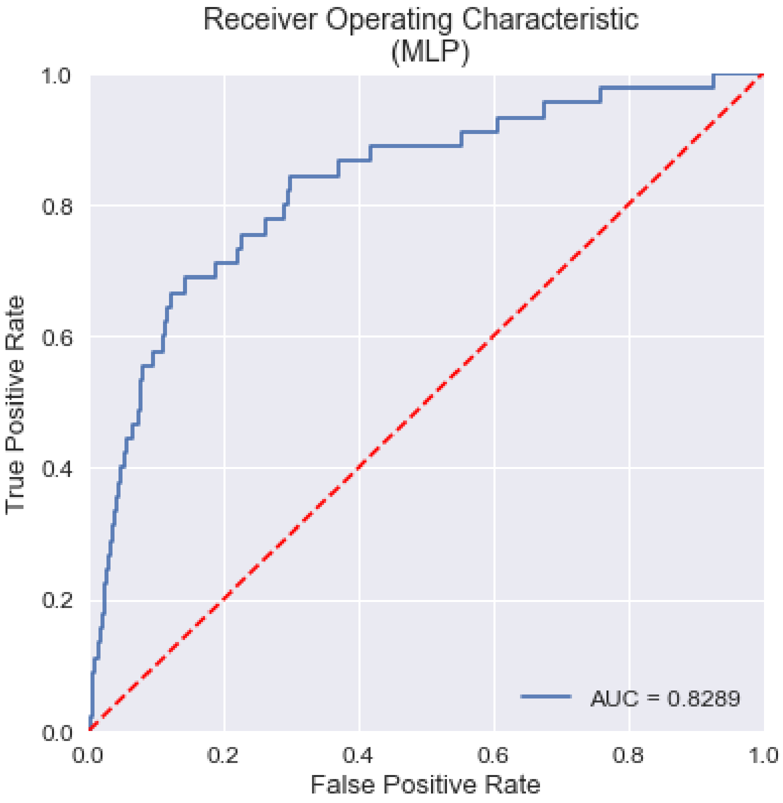
The improvement of our random forest classifier in the testing split of data sample is ~11% on AUC scoring while the performance gain of tuning MLP classifier is ~15% on AUC scoring. From the following picture, the AUC score of our final random forest classifier is 0.9367 which seems already better than the benchmark model's performance. The AUC score of MLP classifier after tuning is 0.8384. This score is still quite far from saying that we have a good result by comparing it to the benchmark model. But so far, all of these results are only based on the training data sample. In next section, our two classifiers will be evaluated by using the official testing data set.

IV. Results

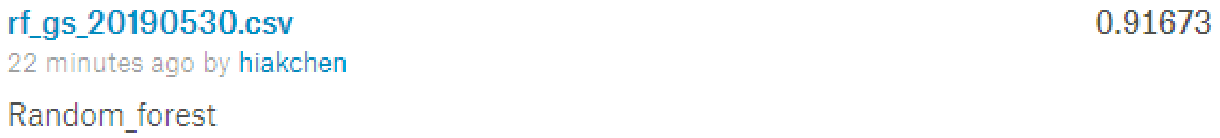
Model Evaluation and Validation

After refinement, the improvement of our random forest classifier in the testing split of data sample is ~12% on AUC scoring while the performance gain of tuning MLP classifier is ~10% on AUC scoring. From the following picture, the AUC score of our final random forest classifier is 0.9367 which seems already better than the benchmark model's performance. The AUC score of MLP classifier after tuning is 0.8289. This score is still quite far from saying that we have a good result by comparing it to the benchmark model. But so far, all of these results are only based on the training data sample.

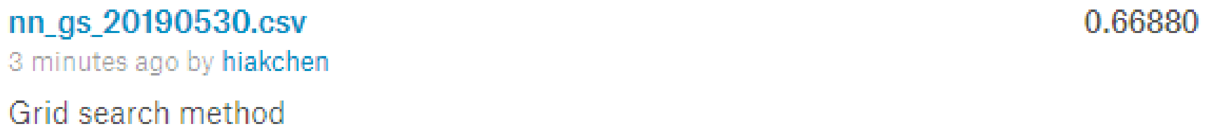




Below we can see the final AUC scores of two kinds of classifier evaluated by using Kaggle testing data (test.csv). This help us to judge whether the model generalizes well to unseen data or not. The random forest classifier receives 0.9167 AUC score while the MLP classifier performances a little bad with only 0.6688. It's quite obvious that our random forest classifier shows less difference between evaluating by testing split of sample data and official testing data set. It seems the model based on random forest algorithm can be our final choice to beat the benchmark model under current setting.



rf_gs_20190530.csv 0.91673
22 minutes ago by hiakchen
Random_forest



nn_gs_20190530.csv 0.66880
3 minutes ago by hiakchen
Grid search method

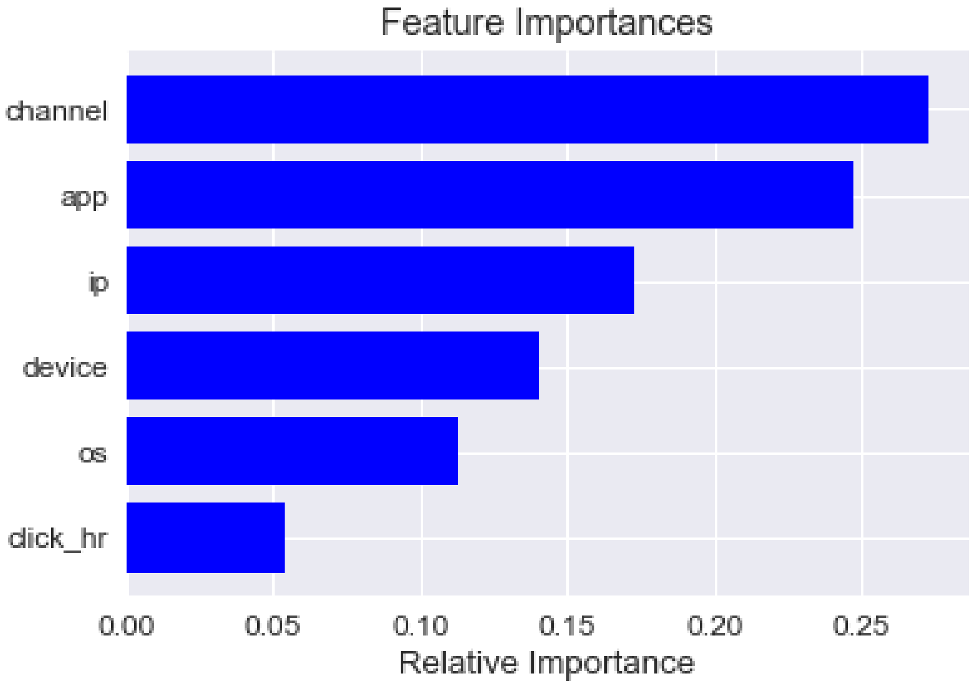
Although the final AUC score, 0.9167, does beat the benchmark model, it is actually not a very good result in this Kaggle competition. The current ranking is 3220-th of 3951 teams. The top AUC scoring is 0.9843 and this means we still have long way to go.

Justification

It seems we had a luck shot to get a model which could just beat the benchmark model with very small improvements. There were already about 10-15% improvements even we only tuned few hyper-parameters and also very limited searching space. And, by comparing the evaluation results with the usage of different testing date set, we could say that our final random forest classifier was quite robust with only 2% difference. Also, tt should be always noted that we only used ~5% of the original training data set to establish our model for solving this classification problem. So I think the model of random forest classifier is trusty and aligns well with our expected solutions outcomes with an acceptable correctness.

V. Conclusion

Free-Form Visualization



The above picture shows the visualization of feature importances, which are extracted from our final random forest classifier. By investigating the important predictors in their order of importance, it can be observed that the feature 'channel' and 'app' are the two most important roles in our modeling. The features 'ip', 'device', and 'os' were less important and share quite closed relative importances to each other. The feature 'click_hr' is the most unimportant one.

All the above results are quite consistent with my understanding. Since we are dealing with the clicking data of a mobile device, the advertisements provided by different mobile ad publishers have different styles even they are try to sale the same product. So people maybe will be attracted to click those spectacular advertisements and download their app. The second important feature 'app' deserve to be valued. An useful, fun or well-designed app will definitely be downloaded frequently. The third important feature 'IP' somehow can be treated as an unique identification of any user. Although sometime people will not always use the same IP to connect to the Internet. But it. And if today we are trying to identify any fraudulence, blocking the IP, which is suspected with their abnormal behavior, is the most efficient to reduce the risks of being hacking or other harmful thing like fraudulence. It's quite reasonable our random forest classifier choose this feature to distinguish between the fraudulent and non-fraudulent clicks. The features 'device' and 'os' are less important. This also makes sense. It's equally possible that every kind of devices or OS can be used to make fraudulent or non-fraudulent clicks. The most unimportant feature is 'click_hr'. This maybe just tells us that people can make click fraud anytime if they want.

Reflection

The workflow of this project could be summarized using the following steps:

1. Defined the problem and collect data
2. Data exploration and preprocessing
3. Created a benchmark model
4. Trained the model with processed data.

5. Refined the model until the best combination of hyper-parameters was found
6. Evaluation and validation

There were two most difficult challenges in this analysis. The first one was the highly imbalanced distribution of data. And the second was how to choose an proper algorithm which could be best suited for the problem we wanted to solve. Also due to the lack of enough computing source, it was not easy for us to use the whole bunch of official training data. Training a model with insufficient data would reduce the prediction ability of our classifiers.

With experience and prior knowledge and also the outcome of AUC scoring on training data sample, we observed that random forest classifier performed better than the MLP classifier in this analysis. It should be noted that this didn't mean the random forest algorithm was always perform better than the neural network in classification problem. One more interesting thing here is that the significant improvements made by tuning the hyper-parameters of each classifier.

To sum up, the final model and solution did fit my expectations for the problem. And of course different kinds of classification problem should always be reexamined like the data exploration and data preprocessing. But the basic workflow is in common use. Hence our model is enough to be treated as a general setting to solve these types of problems.

Improvement

Generally, our model can be improved by trying the following strategies:

- Further Engineering features in order to explore much more hidden information of the data. Maybe some unsupervised clustering algorithm like Principal component analysis (PCA) or t-distributed Stochastic Neighbor Embedding (t-SNE) can be employed in this analysis.
- Training the model with much more data.
- Increasing the searching space of tuning the hyper-parameters
- Using a additional scoring metric like precision-recall curve.

In fact, there are also some improvements we can do for neural network algorithm. A less flexible module of neural network implemented in the library `sklearn` is employed in our original approach. The architecture of a neural network plays an important role in either regression or classification problem. We can try to use some state-of-the-art library like `tensorflow` to build a more complicated neural network and train it with more data. There are already some successful Kaggle cases which can prove this aspect.

Ref.: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html (http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)
Ref.: http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html (http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html)
Ref.: http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html (http://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html)
Ref.: https://www.tensorflow.org/ (https://www.tensorflow.org/)