Machine Learning Engineer Nanodegree

Capstone Project -- TalkingData AdTracking Fraud Detection Challenge

Chang Chia-Hua

Nctueric@gmail.com

2109.05.30

I. Definition

Project Overview

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, China is the largest mobile market in the world and therefore suffers from huge volumes of fraudulent traffic. TalkingData, China's largest independent big data service platform covers over the 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 many clicks, but never end up installing apps. With this information, they have built an IP blacklist and device blacklist.

Fraud behavior detection is endless competition between the developer in the online advertise company and developer in some automation code tool. As a code developer, we are truly like to develop some tools to replace human manipulation on computer or other mobile devices; it will be helpful to improve productivity for many companies. However, the tool will also use to something not for productivity but for profits, such as fraud click. Most of them are clicking by some computers instead of human; it just develops an algorithm to learn to click like human behavior. Therefore, it is interesting to me to study how to build a model to predict it is fraud click even I believe some person may be study how to cheat the model. It is an endless competition.

Problem Statement

The problem is quite clear: **how to predict willing of the users download the apps to install after they click the advertisement.** In other words, what we need is to develop a model to separate the clicks into the meaningful clicks and fraud clicks. Currently, the talking has some methods to predict the fraud click, but it still needs to improve. When we solve this problem, it could provide a better method to improve their prediction accuracy and they will always be one-step ahead of fraudsters. Our goal is to develop an algorithm/model which and precisely predict whether a user will download an app after clicking a mobile app ad based on the recorded properties of that user. These is a binary question, only true or false. The performance will be evaluate on area under the Receiver operation characteristic (ROC) curve between the predicted probability and the observed target. The quality of our model is depending on how small difference between our predictions and truths. After that, the model could use to distinguish between meaningful clicks and fraud clicks and reduced the amount of wasted money caused by fraudulence.

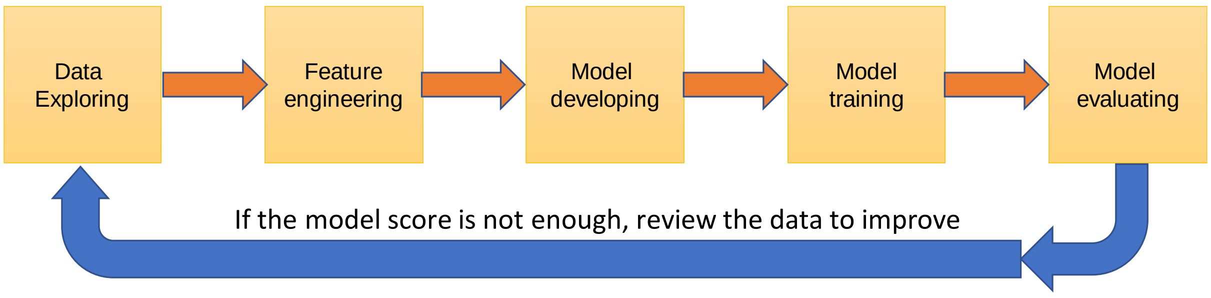


Fig.1 The model developing workflow.

In this project, my workflow will follow a traditional model developing as the Fig. 1 shown. The first priority is investigating the data and developing some basic ideas about the interrelationship between some different features or native properties of each feature. Base on the exploring results, we can create some new features based on the existing features. As the next step, we need to check and clean our data. Because sometimes our data will contain some missing data or repeating data. Therefore, in this data preprocess steps for our model developing, we will need to huddle data about the missing value, outlier value, normalize for numeric features, or do some PCA processing to transport feature value. If we have the categorical feature or text format feature, more data preprocessing techniques will to be consider.

In this project, I will try to focus on building two kinds of classifiers: one is based on the neural network, and the other is based on the random forest. For the neural network, I will construct 2- 3 fully connected layers and try different activation functions for hidden layers. Then the output layer will pass through a sigmoid function for converting the output values as probabilities. Other parameters like optimizer and loss function(s) are to be tuning.

For the random forest, it is an ensemble learning method, which operate by constructing a multitude of decision trees as collecting the contributions from many weak learners. It keeps the advantages of decision tree and makes the final mode more general. That is, with proper parameter setting of each decision trees, random forest can effectively avoid overfitting the training data. Grid search method will be optional for finding the best combination of model's parameters.

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 as "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 $\mathrm{TPR(T)} = \int_T^\infty f_1(x)d(x)$ and the false positive rate is given by $\mathrm{FPR(T)} = \int_T^\infty f_0(x)d(x)$. 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)

Once we have finished training our models, we will use the evaluation metric, AUC, to evaluate our training models. The models will be review thoroughly to check if anything need to improving, and the Kaggle’s officical evaluation method will be taken into account. Once our model is get the acceptable score is at least over 0.91 (outstanding discrimination), we can finish that model.

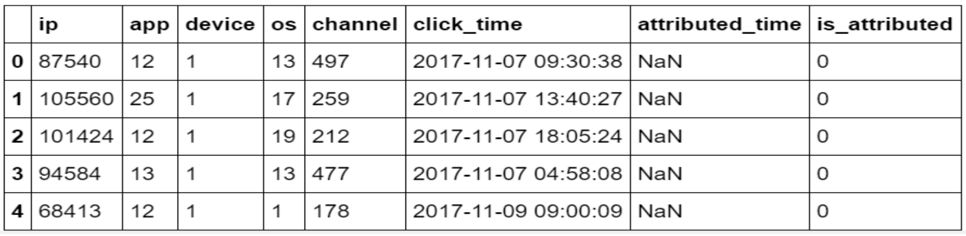
Ref: https://en.wikipedia.org/wiki/Artificial_neural_network
Ref: https://en.wikipedia.org/wiki/Random_forest

II. Analysis

Data Exploration

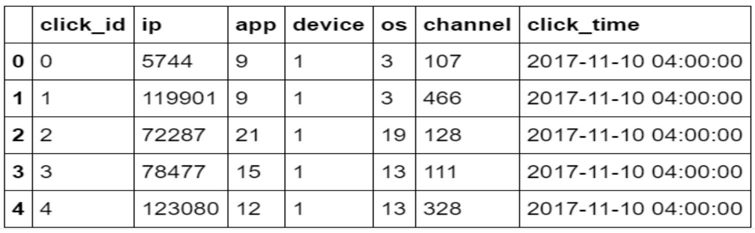
In this competition, **our objective is to predict whether a user will download an app after clicking a mobile app advertisement**. To support your modeling, they have provided a generous dataset covering approximately 200 million clicks over 4 days! These data has downloaded on the website: https://www.kaggle.com/c/talkingdata-adtracking-fraud-detection
The dataset file descriptions:

- train.csv - the training set (184903890 rows)



	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
- test.csv



	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

These data has eight features in each row of training data and training sample data.

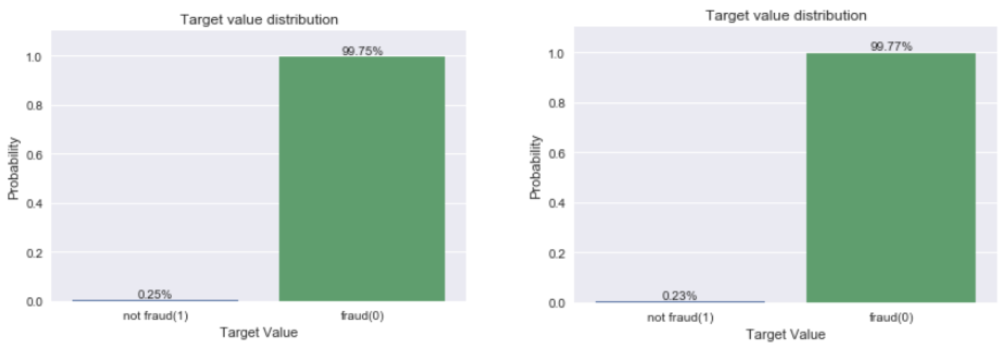
Training features:

- ip: ip address of click.
- app: app id for marketing.
- device: device type id of user mobile phone.
- 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

Target features:

- is_attributed: the target that is to be predicted, indicating the app was downloaded

These data have seven training features that are used to predict the click is a fraud or not, and one target feature 'is_attributed' that indicated the app was downloaded. The training data has total 184903890 rows, and the sample data has 100000 rows. By calculating the target distribution form these two dataset, both of these data has the same probability distribution about the fraud as shown in the fig. (a) and fig.(b). We can find the fraud probability is extremely high to 99.75%. Therefore, it may need re-sampling process to balance these data if our training model is bad to predict the results. Because these training data are extremely large for our personal computer to training, we will use the sampling data to train and divided into three group training, validation, testing. After our training process, we will use the test data download from Kaggle to do the final evaluation to verify our model performance.



(a) (b)

Fig.2 The target value distribution in the whole training data(a) and the sample data(b).

Exploratory Visualization

During the data exploring, I remove the feature 'attributed_time' from the training dataset. Because our testing data does not include this feature, so this feature cannot include in our training set to build our model for prediction. Apart from this, the "attributed_time" is highly related to the internet environment of the user, it may be a meaning less or low impact feature to decide our model. Therefore, I only focus to analysis the other six features.

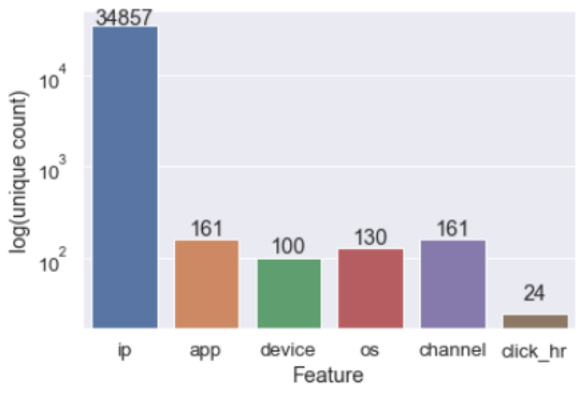


Fig.3 the unique values for each feature.

In the Fig.3, the 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 features 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 each device connected to the Internet. If two devices are connecting to the Internet in the same time, they definitely will not 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. As a result, 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 many 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 many mobile ad publishers in China, but 200 is a reasonable number of mobile ad publishers.

As shown in the Fig.4, the histograms demonstrated the 10 most frequent values of click data with reference to 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 is 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. They can be iOS or some Android system. It is 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. They often feel bored when they are working so they like to use their mobile in the daytime. We only show the percentage of odd bins due to the limited space of bar chart.

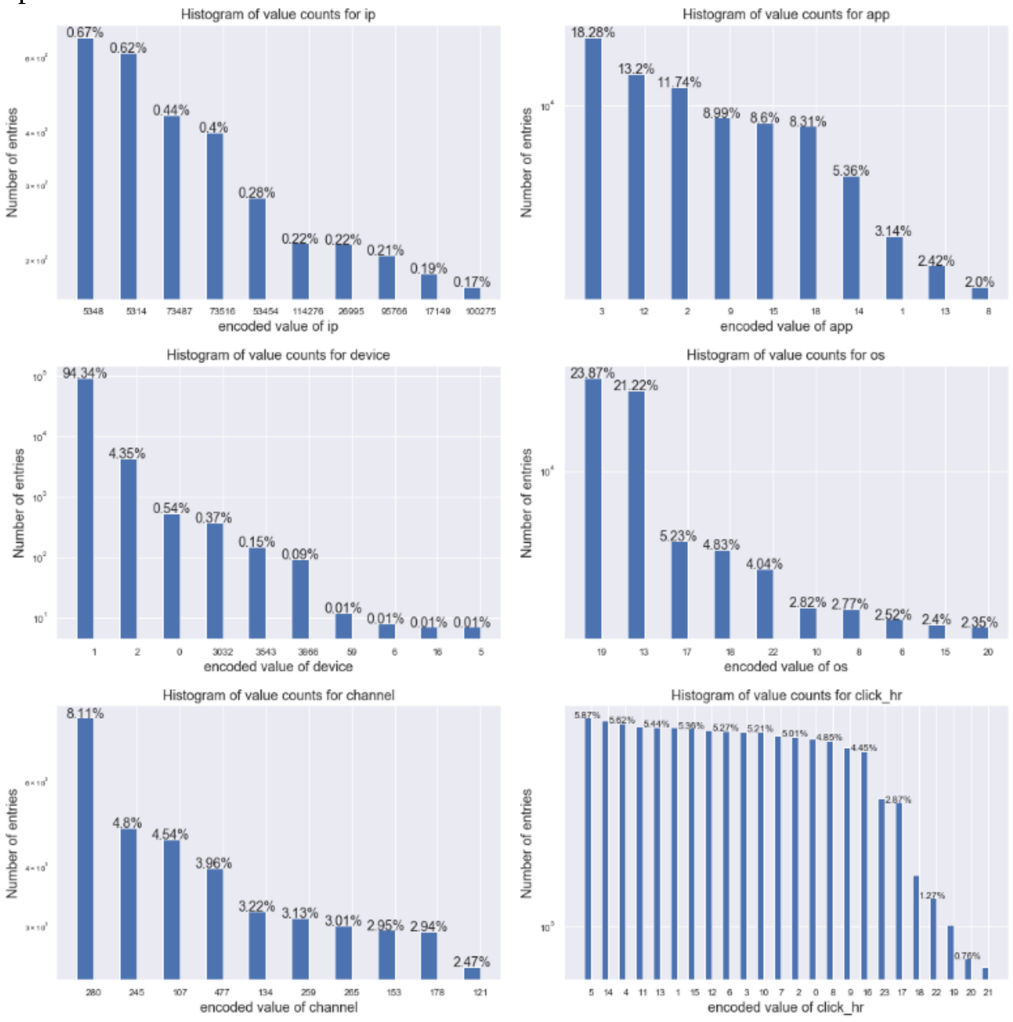


Fig.4 The 10 most frequent values of click data with reference to discrete categorical features, 'ip', 'app', 'device', 'os' , 'channel' and click_hr.

By converting our data into heat map as the fig. 5 shown, we can clearly figure out the correlations between different features. The correlation between any two features is high if the value of the corresponding block is above 0.5. It is obvious that features 'device' and 'os' are highly correlated with each other as we have expected before. However, it is 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.

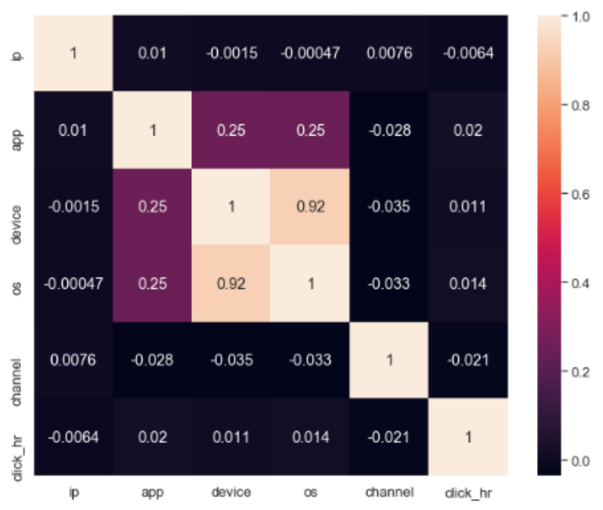


Fig.5 The heat map of the correlations of six critical features.

Algorithms and Techniques

Two model structures will be evaluate for this project. One is the **Random Forest** and the other is the '**Neural Network**'. The Random Forest is an ensemble-learning algorithm. The combination of learning models (learners) increases the classification accuracy and makes itself more robust. For Random Forest, the learner is a Decision Tree. A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. Generally, it also make lower cost of calculation and can handle both continuous values and categorical variables of data. However, Decision Tree tends to over-fit the data, and it does not generally learn something from the data but just only memorize them. It may make terrible prediction on those un-training 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. Besides, such tree-based algorithm can handle either numerical or categorical data as input quite well. The process flow of Random Forest algorithm is shown in the Fig.6.

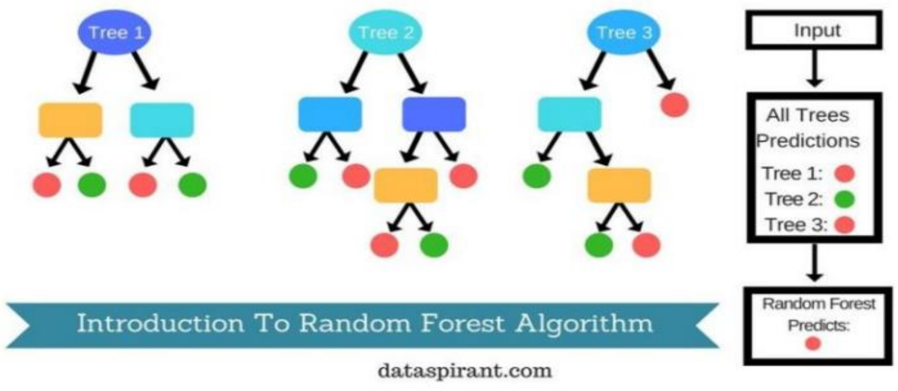


Fig.6 The process flow of Random Forest algorithm. (Ref from dataaspirant.)

Ref : http://dataaspirant.com/2017/05/22/random-forest-algorithmmachine-learing/

The following hyper-parameters can be tune 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.

The second algorithm, Neural Network, or so-called Multi-layer Perceptron (MLP), 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 pass through a sigmoid function for converting the output values as probabilities. Furthermore, the following hyper-parameters can be tune 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

Due to the large size of our sample data and al so the full data, mini-batch gradient descent method would be our choice when we train our neural network classifier. Besides, a useful technique 'grid search' will be included in this analysis for finding the best combinations of hyper-parameters of a model.

As mentioned previously, our dataset was extremely imbalanced that only very few fraction of data are not labeled as fraudulent. Such imbalanced distributed data may induce the biased prediction of model. Many machine learning algorithms place tend to put more emphasis on learning from data observations, which occur more commonly. Resampling strategies are necessary, such as over-sampling and under-sampling. Therefore, three kinds of data samples will use for training our random forest classifier and neural network classifier, respectively.

1. The original data sample.
2. New data sample generated by over-sampling algorithm, Synthetic Minority Oversampling Technique (SMOTE). The SMOTE generates new observations by interpolating between observations in the original dataset.
3. New data sample generated by under-sampling algorithm, 'Tomek's links'. It will implement a heuristic, which will clean the dataset based on Tomek's links. 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.

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.

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. 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

Because the original full data is too large, the data sample '**train_sample.csv**' will be used for training and validation/testing in this analysis. 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:
    - Over-sampling data produced by SMOTE.
    - Under-sampling data produced by 'Tomek's link' method.

Implementation

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

- Random Forest ensemble method
- Neural Network (Multi-Layer Perceptron)

The implementation is in three 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 evaluated by using AUC. The value of AUC varies from 0 to 1. Our goal is to chasing a model with higher AUC value.

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

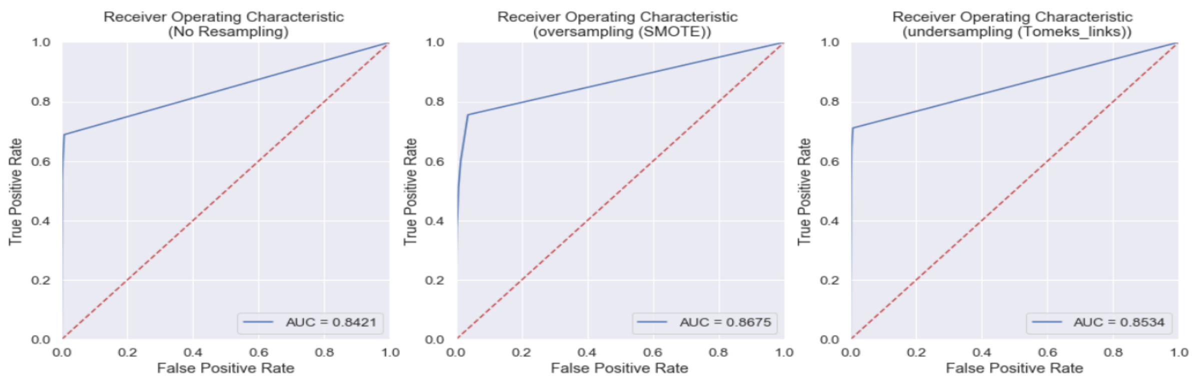


Fig.7 The AUC value of three models (Random Forest model) evaluation that are training with three datasets, original data, over-sampling and under-sampling data.

The visualization of Random Forest modeling results are shown in the Fig.7. These results 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 ~ 0.86, but still less than the benchmark one, which has 0.911 AUC score. There are still something to be improvements.

In the second part, let us 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=0.004. 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=3

The visualization of MLP modeling results are shown in the Fig.8. These models are also evaluated by using testing set of data sample, the MLP classifiers trained by using different training data samples show a little bit different accuracy, ~99.78% for ordinary sample and new sample produced by under-sampling. The one trained with oversampling data set shows less accuracy ~97.4%. The AUC scores of different MLP classifiers vary from ~0.50 to ~0.65. The classifier trained with oversampling data set perform relatively better than the other two with ~0.65 AUC score. The one trained with under-sampling data set only give a not-that-good prediction, ~0.50 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 is obvious that there are also still something to improving our MLP classifier(s).

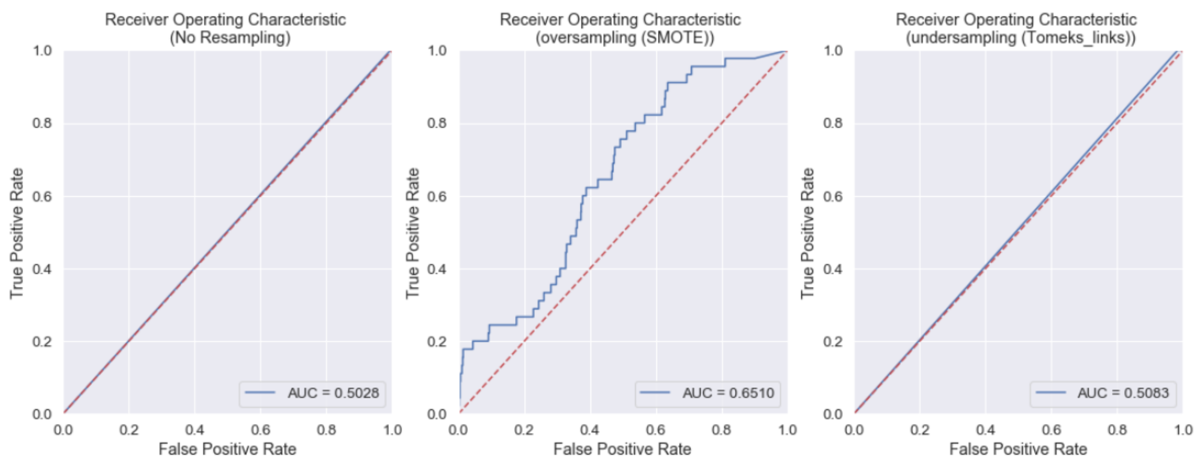


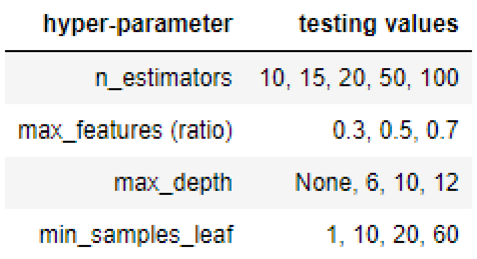
Fig.8 The AUC value of three models (MLP model) evaluation that are training with three datasets, original data, over-sampling and under-sampling data.

In summary of our initial modeling, we 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. When we using oversampling data set to train our MLP classifier, it shows such there is 30% improvement in predicting the testing split of the original data sample. Therefore, we modified our modeling strategy based on the results. The random forest classifier will trained without taking resampling into account. 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, which uses an estimator and a set of hyper-parameters to generate model candidates and evaluates them with proper/user-defined metric score. Hence, we 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

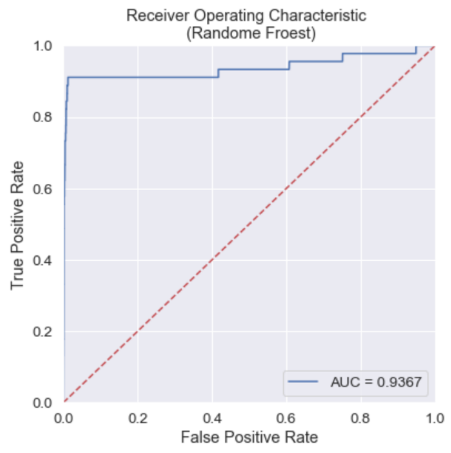
For the solver of optimization algorithm, the 'lbfgs' stands for 'Limited-memory BFGS' while 'sgd' is the abbreviation of 'Stochastic gradient descent'. The 'logistic' function of the hyper-parameter 'activation' is also known as the sigmoid function.

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:

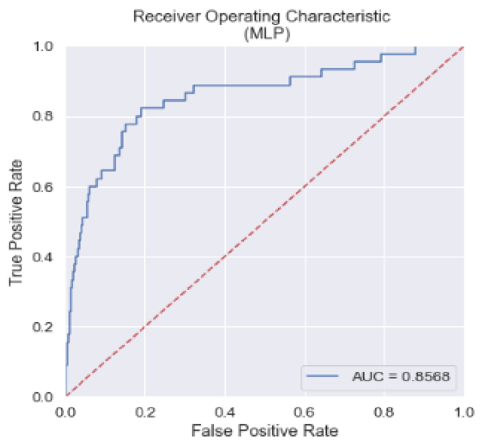
Random Forest	Multi-layer Perceptrons
n_estimators = 50	solver = 'adam'
max_features = 0.3	activation = 'relu'
max_depth = 12	hidden_layer_sizes = (128, 128)
min_samples_leaf = 20	learning_rate_init = 0.005

IV. Results

Model Evaluation and Validation



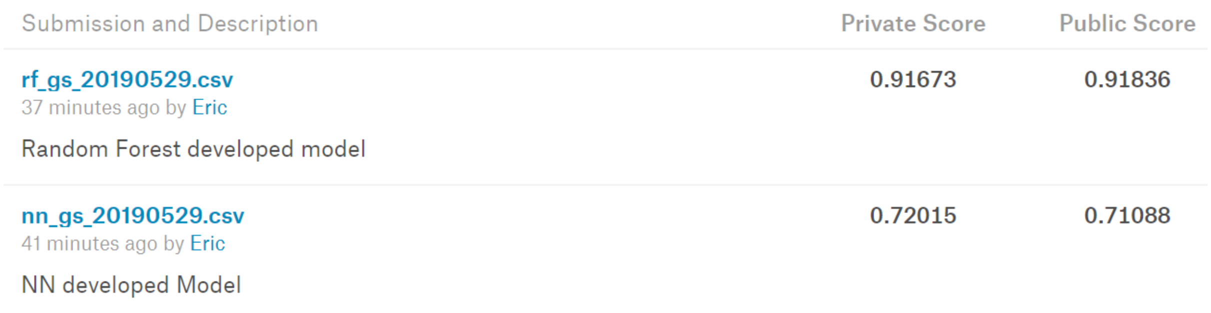
(a)



(b)

Fig.9 The tuned model evaluations of the random forest model (a) and MLP model (b).

In the Fig.9, 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 0.8568. This score is still quite far from saying that we have a good result by comparing it to the benchmark model. Currently, 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.



Submission and Description	Private Score	Public Score
rf_gs_20190529.csv 37 minutes ago by Eric Random Forest developed model	0.91673	0.91836
nn_gs_20190529.csv 41 minutes ago by Eric NN developed Model	0.72015	0.71088

The final AUC scores of two kinds of classifier evaluated by using Kaggle testing data (test.csv). The random forest classifier receives 0.9183 AUC score while the MLP classifier performances a little bad with only 0.7108. It is 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. Although the final AUC score, 0.9183, does beat the benchmark model, it is actually not a very good result in this Kaggle competition. 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 8-10% improvements even we only tuned few hyper-parameters and very limited searching space. By comparing the evaluation results with the usage of different testing date set, our final random forest classifier was quite robust with only 2% difference. However, we only used ~5% of the original training data set to establish our model for solving this classification problem. 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 visualization of feature importance are shown in the Fig. 10. The importance ratio 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 importance 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. The useful, fun or well-designed app will definitely be downloaded frequently. The third important feature 'IP' somehow can be treated as a unique identification of any user. Although sometime people will not always use the same IP to connect to the Internet. 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 is 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 is 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.

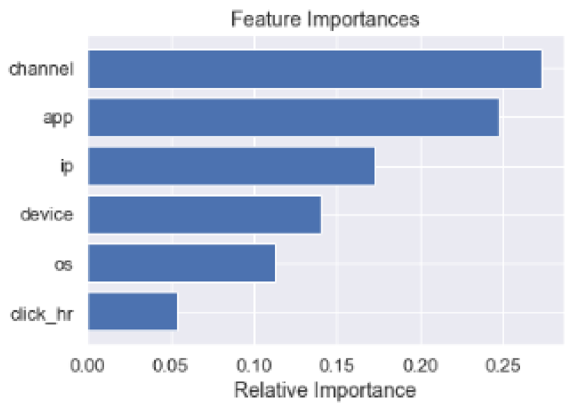


Fig.10 The relative importance for each features.

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

With experience and prior knowledge and the outcome of AUC scoring on training data sample, we observed that random forest classifier performed better than the MLP classifier in this analysis. However, this did not 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. Different kinds of classification problem should always be reexamined like the data exploration and data preprocessing. However, the basic workflow is in common use. Therefore, 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 additional scoring metric like precision-recall curve.

In fact, we can still further improve 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.