Capstone project report: Facebook recruiting iv robot vs human.

Project overview

Robot vs human is a project which aims at predicting human or robot bidder based on their behavior, the motivation is to make the fair competition on the bidding website. The problem involves the data mining theories and techniques, which we could try to solve it with the prediction algorithm of machine learning. The data is provided by Facebook and the personal information was all encoded for privacy reasons.

Problem Statement

This is a **supervised binary classification problem**, we need to judge one bidder between a human or a robot based on limited bidding records. The basic account information and the bidding records of all the bidders in consideration are given as the initial data. However, in this project, there is no pre-built feature at all, one must construct their own features with the bidding records provided. This posed the first challenge of this project: **constructing all your features for training by yourself.** It makes the problem more difficult because for the most data mining problem, we know the famous saying "garbage in, garbage out"[1]. Hence the input data is much more crucial than the classification algorithm.

My pipeline of solving this problem is concluded as below:

1. Understand as much as possible about the given data.

2. Explore the data to analysis the possible different features which could distinguish the human and robot in the training site.

3. Construct the features based on the analyses ahead.

4. Choose several algorithms and compare their performance with our features.

5. With the possibly most promising algorithms, continue with necessary features selections techniques, hyperparameter tuning process, etc. In order to improve the performance

6. Evaluate the final performance with Leader Board and goes back to the former steps to improve the results.

Metric

Submissions are evaluated on the area under the ROC curve [2].

The **receiver operating characteristic curve**, i.e. **ROC curve**, illustrates the diagnostic ability of a binary classifier system.

The ROC curve is created by plotting the **true positive rate** (TPR) against the **false positive rate** (FPR).

$$\text{TPR} = \frac{True\ Postive}{True\ Positive + False\ Negative}$$

$$\text{FPR} = \frac{False\ Postive}{True\ Negative + False\ Positive}$$

The out put score of our model will be "Area Under Curve" (AUC), which is the area of the curve of ROC. AUC is equal to the probability that a classifier rank a randomly chosen positive instance higher than a randomly chosen negative one.

Analysis

Data Exploration

The data is from online platform, and is inclusive of two datasets, the following description of dataset is directly from the Kaggle competition set.

- One is a **bidder dataset** that includes a list of bidder information, including their id, payment account, and address.
  - train.csv, test.csv
- The other is a **bid dataset** that includes 7.6 million bids on different auctions, which are all made by mobile devices. Each bid has a fixed increment of dollar amount.
  - bid.csv

For the bidder dataset

Features	**Type**	**Description**
bidder_id	nominal	Unique indentifier of a bidder.

Features	Type	Description
payment_account	nominal	Payment account associated with a bidder. These are obfuscated to protect privacy.
address	nominal	Mailing address of a bidder. These are obfuscated to protect privacy.

label	Type	Description
outcome	numerical	Label of a bidder indicating whether or not it is a robot. Value 1.0 indicates a robot, where value 0.0 indicates human.

The outcome was half hand labeled, half stats-based. There are two types of "bots" with different levels of proof:

- Bidders who are identified as bots/fraudulent with clear proof. Their accounts were banned by the auction site.
- Bidder who may have just started their business/clicks or their stats exceed from system wide average. There are no clear proof that they are bots.

For the bid dataset

Features	Type	Description
bid_id	nominal	unique id for this bid
bidder_id	nominal	Unique identifier of a bidder (same as the bidder_id used in train.csv and test.csv)
auction	nominal	Unique identifier of an auction
merchandise	nominal	The category of the auction site campaign, which means the bidder might come to this site by way of searching for "home goods" but ended up bidding for "sporting goods" - and that leads to this field being "home goods". This categorical field could be a search term, or online advertisement.
device	nominal	Phone model of a visitor
time	numerical	Time that the bid is made (transformed to protect privacy).

Features	**Type**	**Description**
country	nominal	The country that the IP belongs to
ip	nominal	IP address of a bidder (obfuscated to protect privacy).
url	nominal	url where the bidder was referred from (obfuscated to protect privacy).

By preliminary exploring of the records, the basic numbers are shown in below,

Number of bidders in train.csv	Number o bidders appear in bid.csv	Number of bidders in train.csv and bid.csv	Number of bidders in test.csv	Number of bidders in test.csv and bid.csv
2013	6614	1984	4700	4630

Noted that not all bidders listed in train.csv or test.csv appear on the bid.csv. For the training side, it means we could only use 1984 bidders as training set, it is a relatively small sample set, which have potential difficulties in generalization.

In the test.csv, we also notice that not all bidders for testing are recorded in the bid.csv, which could cause the submitting error on Kaggle. This had been noted as a bug by Facebook, and all the missing bidders were asked to be noted -1 as their prediction in the submission.csv.

As small as a training set it is, the number of human and robots are shown as below,

Number of human bidders	Number of robot bidders
1881	103

We notice that this is also a highly imbalanced sample set, which could potentially have different feature distribution from the test samples. This implies overfitting threats even before the project.

Since the time pattern of bidders' behavior maybe important, there are some basic time characters of the data,

Total number of the different time points	Presumed duration of the bid records	The minimum encoded time stamp	The maximum encoded time stamp
776529	9 days	9631916842105263	9772885210526315

By intuition, we need to figure out some features that are distinguishable enough to the human and robot bidders. We mainly try to compare different features in human and robot set separately, by visualization and quantification.

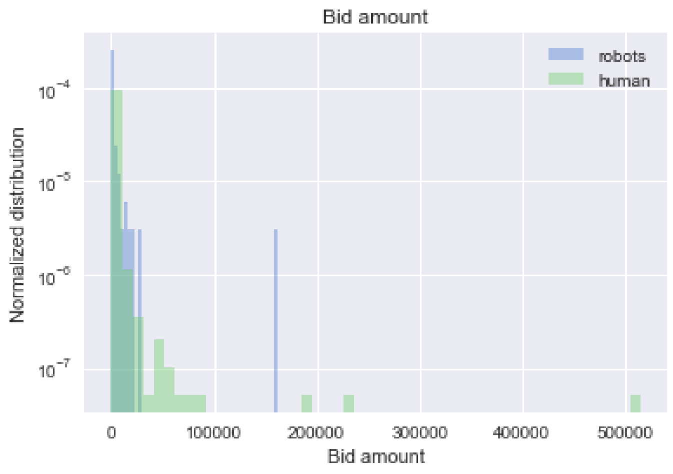
Exploratory Visualization

According to the records provided by the bid.csv, we are inspired by the categories initially provided by the records. The feature exploration starts with the following categories:

- Bids number
- Devices
- Urls
- IPs
- Time intervals
- Auctions

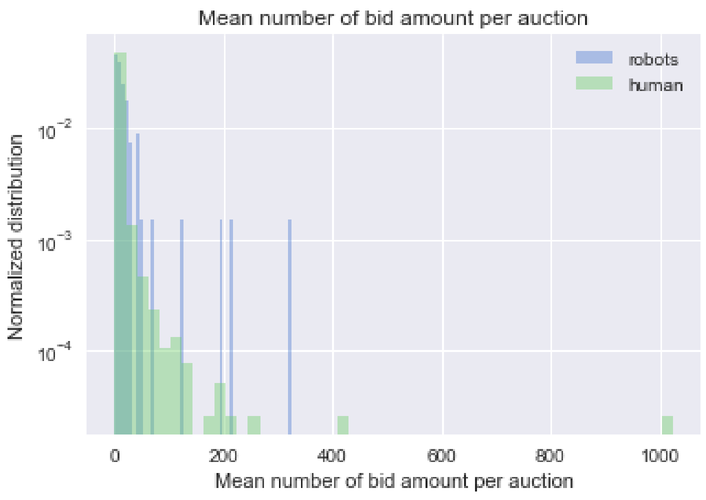
The basic idea is to firstly analysis the potential differences between human and robot by these categories and try to compare the differences visually and quantitively between human and robot. Visually, we will plot the value distribution of human and robot for the interested feature and check the difference. Quantitively, we will use population stability index (PSI) [4] to show how significant two distributions are.

The first simple idea is that the robot definitely performs differently in bids number than the human. Robot could be considered as more efficiently, thus the bids number could be less, on the other hand, individual human may perform by emotion and with less specific objective. This may result with more general large number bids and small bids.

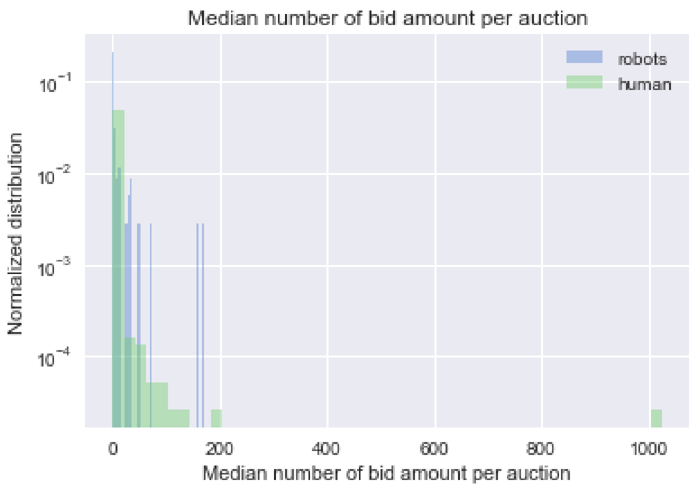


The figure above indeed shows some of presumed patterns, the PSI=2.3, which indicates a significant difference in bids number.

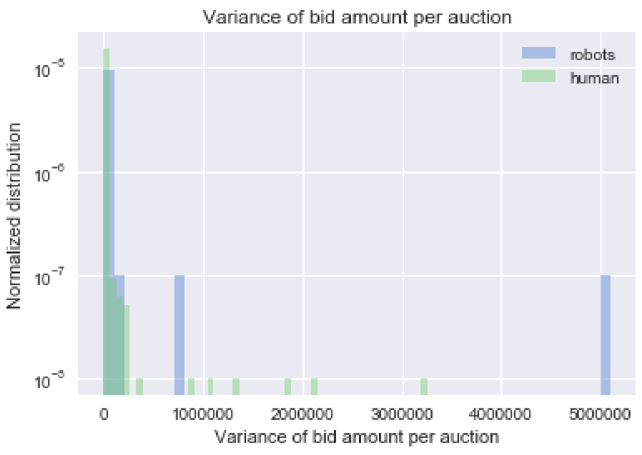
Following the idea that the robots are more efficient and dedicated to win the bid of interested auctions, the number of bid amount per auction could cause the major differences too. Here we considered the mean value, the median value and the variance of the bid amount per auction,



PSI=2.58



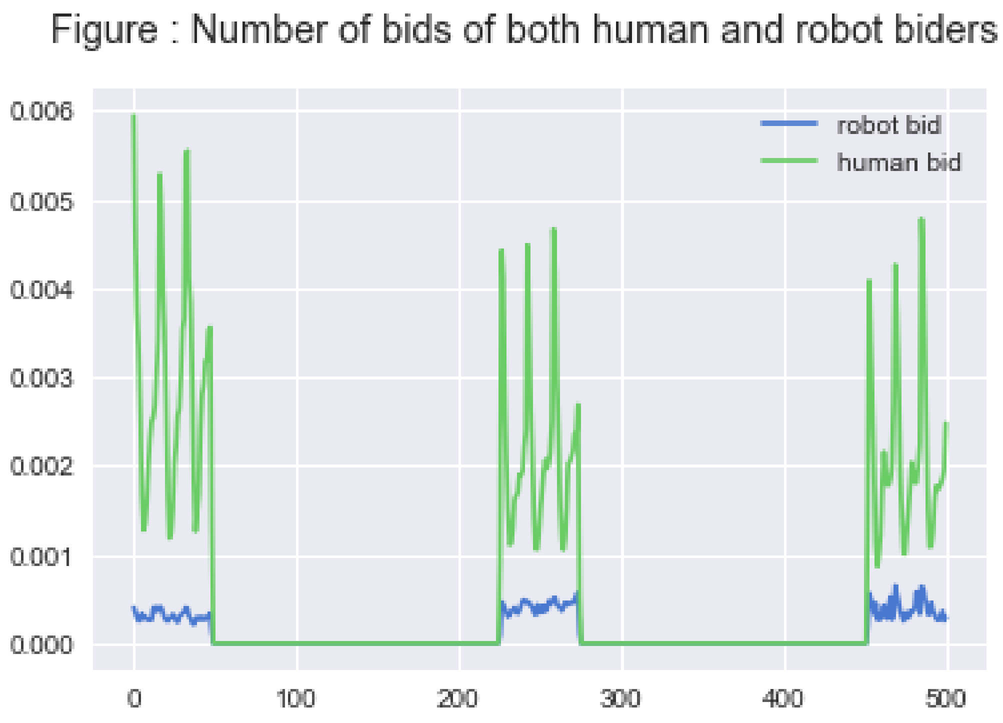
PSI=1.52



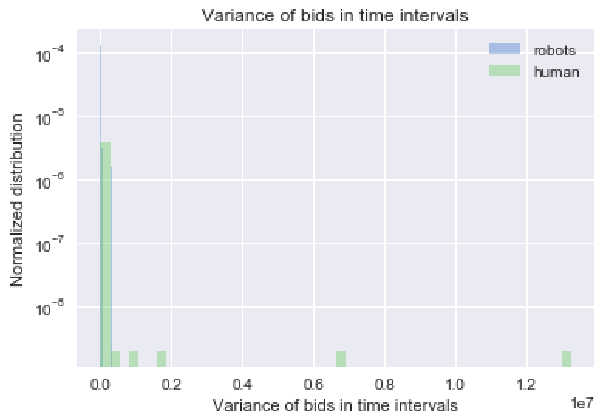
PSI=2.17

We can see from both the figure and the PSI results, that the differences are fairly obvious.

Other bid pattern interested is the pattern with time, here we firstly looked at the bid number (normalized by total records number) by time (here we divided the records into 500 equal time intervals),



We could see that this is an interesting figure, the bidding behavior mainly concentrate in three days (presumably). Despite of the natural difference of amplitude (there are more human bidders), the useful difference here may be the 'oscillation amount', which could be presented by the variances.

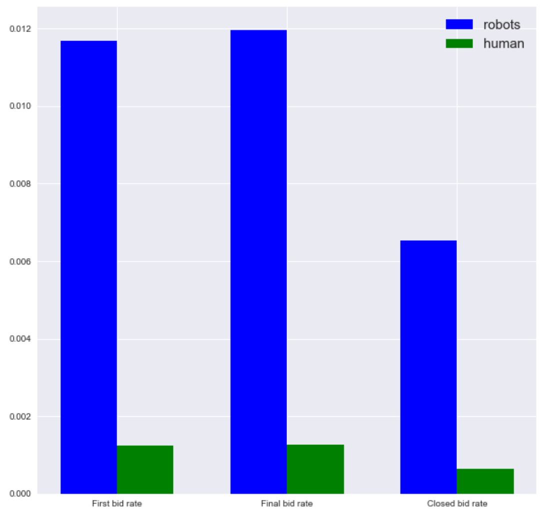


PSI=2.7

The figure of variances above is confirm our assumption, this feature may worth consideration. What's more, it shows that the time interval character maybe a useful perspective.

In the following, similar idea has been considered with devices, urls, IPs and auctions. The main clue is that the robot should be more goal-oriental and resourceful. Thus, the devices, urls, IPs or time intervals used for each auction or bid may be different. Therefore, similar steps were taken for these possible features, the comparing figures and PSI could be found in the data_investigation.ipynb file.

Besides these, we also could think about the possible unique behaviors by the robots. They are determined as we considered, thus, they may be more likely to start a bid, keep biding on it and at least more likely to finish the last bid. Hence, we took a glance of what are the rate of first bid, last bid and even closed bid for robot and human bidders.



It is rather clear that the robots are much more likely to start the bid, finish the bid, or start and finish all by itself. Thus, these could also be an potential feature construction direction worth considering.

The features we have discussed so far with considerable PSI values are listed below.

Mean number of bids for each auction by every bidder.
Median number of bids for each auction by every bidder.
Variance of bids for each auction by every bidder.
Number of bids
Variance of bid amounts in time intervals
Number of unique devices.
Mean number of unique devices used for auctions by every bidder.
Mean of same urls used per auction.
Variance of same urls used per auction.
Bids number in 9 time intervals by each bidder.
Total bid time intervals for all auctions.
Mean bid time intervals between every bids for the auctions.
Total bid time periode (last bid time step- first bid time step) within all 9 days by every bidder.
Mean unique auction interested in time intervals
Unique auctions interested by each bidder
First bid rate of individual.
Final bid rate of individual.
Closed (conducted both first and final bids) bid rate by individual.

These are the features we intend to feed into the model of binary classification, Next we would like to proceed with a quick model test run to see what are these features like and layout our way in the future steps.

Algorithms and Techniques

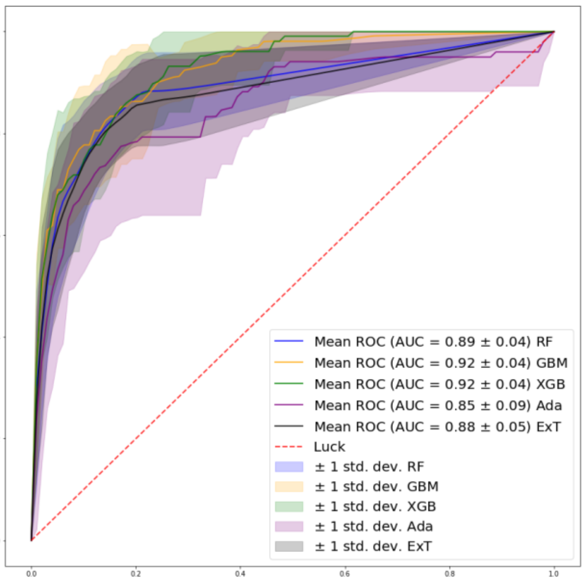
With the features proposed in the previous step, I constructed these features with pandas in python3. Before doing any other feature selection stuff, I start with simple log transformation [5] and MinMaxScaler [6] in order to ease the pain of highly skewed data. Since we have already see from the former distributions that these features seem to be quite skewed.

With the features preliminarily preprocessed, here we tried several ensemble learning algorithms to see their performance all together. Such as XGBoost [7], GradientBoost [8], RandomForest [9], Adaboost [10] and ExtraTrees [11].

Simply put, ensemble algorithms come from the same basic idea, that is to combine the power of many individual weaker predictors and make the outcome performance impressive. The weak predictor is generally decision trees. The ensemble methods tested here are mainly based on bagging and boost, which are two main ensemble techniques.

- Bagging: This approach will build many bootstrap replicates of the data, and each train a classifier and finally vote for the results with all these classifiers. This approach could do better in reducing variance.
- Boost: Each classifier is trained based on the results of former training result of another classifier. During this process, the 'better performed' classifier and bad predicted samples will be weighted up. Meanwhile the 'poorly performed' classifier and good predictions will be weighted down. This approach could do better in reducing bias.

XGBoost, GradientBoost and Adaboost applied the boost approach, while RandomForest and ExtraTrees. Generally, ensemble algorithms are powerful tools in the data mining projects. And comparing to other black box algorithm like SVM and Neural Network, these tree-based ensemble algorithms could have shown the importance of each features, thus lead to more explainable results.



With the constructed features, I have firstly fed them into these algorithms to compare. The figure above shows the ROC curve and AUC value of each algorithm. The features are preprocessed with log transformation and MinMaxScalar, meanwhile every curve is the average value of 10-fold cv result, the standard deviation of cv results are noted as the shadows around curves.
From the quick comparison, we could see that the XGBoost and Gradient Boost has stepped out in performance. XGBoost is like a high-performance version of Gradient Boost, therefore we are going to continue with XGBoost.

We mainly used scikit-learn package [12] and also dmlc XGBoost package [13] in this project. For visualization we used seaborn [14] and matplotlib [15] package

Benchmark

This competition project has no benchmark to begin with, therefore I would like to start with top500 of the Leader Board as a benchmark.

ranking	Public Leaderboard	Private Leaderboard
500	0.86087	0.88260

The LB score is obtained by the participants in this competition, however, no initial codes were provided by the organizer. The features are analyzed and constructed by every participant him/herself.

Methodology

Data Preprocessing

As shown in the code file, in the data process stage, I implemented 4 functions to transfer and select the constructed features.

- **prep()**:
    - Log transformation
    - MinMaxScaler

- **psi_f()**:
    - Population stability index is for comparing the feature distributions between train data and test data, in order to do the feature selection so that the distribution of features in training set are close to the testing set.

- **f_imp()**:
    - Select the important features according to the algrithm.

- **drop_one_bid_bots()**:
    - Ruling out the robot bidders features which only has one bid record from the training set.

In the train sample, there are 5 robot bidder who only have one bid record. This is considered as the outliers and it may be confusing for our training models. Therefore, we have firstly removed these 5 robots from the train sample. The bidder ids are list below:

'bd0071b98d9479130e5c053a244fe6f1muj8h'
'91c749114e26abdb9a4536169f9b4580huern'
'74a35c4376559c911fdb5e9cfb78c5e4btqew'
'f35082c6d72f1f1be3dd23f949db1f577t6wd'
'7fab82fa5eaea6a44eb743bc4bf356b3tarle'

As we have seen in the data exploration, the train sample is rather small comparing to the test sample, there is a great chance that the generalization will not be so ideal. What needs to be considered is whether the features that we constructed distributed the same in both train sample and test sample. By inspecting this, we hope to ensure the generalization capability of our features.

In This step, we have used the population stability index (PSI) [16] to compare the feature value distribution between train set and the test set. We have tested if the differences of PSI are larger than 0.1:

PSI score	
mean_num_auction_bids	0.030848
median_num_auction_bids	0.009824
var_num_auction_bids	0.022333
number_of_bids	0.023714
var_bids_num_per_time_interval	0.015419
num_unique_devices	0.021357
mean_unique_devices_per_auction	0.017881
mean_same_url_per_auction	0.018286
var_same_url_per_auction	0.021267
max_urls_per_auction	0.017570
mean_same_ip_per_auction	0.022581
var_same_ip_per_auction	0.013638
bids_day1	0.013923
bids_day2	NaN

PSI score	
bids_day3	NaN
bids_day4	NaN
bids_day5	0.014247
bids_day6	NaN
bids_day7	NaN
bids_day8	NaN
bids_day9	0.024754
tot_bid_time_all_auctions	0.035251
mean_bid_time_interval	0.031656
mean_unique_auction_interested_in_time_intervals	0.029676
num_of_unique_auctions	0.020626
tot_bid_periode	0.026407
first_bid	0.004467
final_bid	0.013532
f_f_bid	0.001867

From this table, we have almost every feature selected, despite of several NaN features that are actually all 0.

In the data exploration step, we could notice from the figures that the features are somehow quite skewed and are in different scales. In order to make them feasible to the XGBoost algorithm, we would like to ease the affect of skewed data by doing the log-transformation and MinMaxScale. Every feature has been preprocessed like this after the PSI checking.

Finally, since we used the ensemble algorithm, the feature importance are computed and by far we noted that we can achieve the best result with an importance threshold > 0.001 (all of these feature considered).

Implementation

I used the xgboost package to import XGBoostClassifier. A **modelfit** function was implemented to fit the training set with Repeated Stratified KFold. I used this method to try to reduce the overfitting. The cv_fold is set to 5 and the number of repeating cv is 50. Larger number seems not do any better in the current situation. In this function, a cv function is included so that a optimum number of estimators could be determined if wanted.

I have firstly fed my preprocessed data into the XGBoostClassifier with a set of starting parameters:

learning_rate=0.001, n_estimators=500, max_depth=5, min_child_weight=1, gamma=0, subsample=0.8, colsample_bytree=0.8;

Meanwhile, since the binary samples are highly imbalanced, I used the parameter scale_pos_weight to ease the tendency of overfitting. scale_pos_weight is set to the ratio of human bidders numbers to robot bidder numbers.

By the starting trial, we could obtain a mean AUC score around 0.9254. Then I submitted directly to the Kaggle website, I got a **Private Score of 0.90864** and a **Public Score 0.89176.** The screen shot of the submission is shown below.

submission_2018-7-24.csv 0.90864 0.89176
29 minutes ago by BoQu
add submission details

Refinement

Parameter tuning

I followed a general parameter tuning instruction for XGBoost,[17]. The turning procedure is concluded as below:

1. Start with a relatively high learning rate, here we start as the initial model learning_rate=0.001, an optimum number of estimators is already obtained in the early step, we start with n_estimator=16.
2. Tune tree-sepcific parameters, order by: max_depth, min_child_weight, subsample, gamma, colsample_bytree. The optimum parameters are shown by the table below, related visualization could be found in the code file.

max_depth	min_child_weight	subsample	gamma	colsample_bytree
17	1.1	0.5	0.1	0.3

3. Tune regularization parameters such as reg_alpha and reg_lambda:

Reg_alpha	Reg_lambda

0.0	1.0

4. Lower the learning rate again and determine the best n_estimators:

Learning rate	n_estimators
0.0005	101

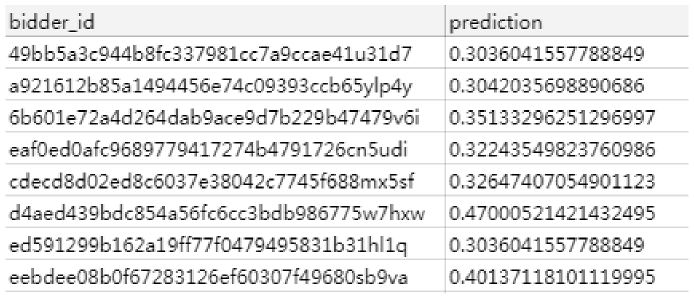
With these tuning procedures, we have a new score on the leader board of **Private Score: 0.9115** and **Public Score: 0.89172**

submission_2018-7-25_4.csv 0.91150 0.89172
12 hours ago by BoQu
add submission details

We could see an improvement of the LB score by the parameter tuning, even through the affect is not that hefty.

Model ensemble

Based on the parameter tuning, we would like to further improve the performance, and model ensembling seems to be a useful technique that worth a shot. Here I referred an ensembling guide for Kaggle [18]. Since the probability outputted by a very small learning rate is very close like this:



bidder_id	prediction
49bb5a3c944b8fc337981cc7a9ccae41u31d7	0.3036041557788849
a921612b85a1494456e74c09393ccb65ylp4y	0.3042035698890686
6b601e72a4d264dab9ace9d7b229b47479v6i	0.35133296251296997
eaf0ed0afc9689779417274b4791726cn5udi	0.32243549823760986
cdecd8d02ed8c6037e38042c7745f688mx5sf	0.32647407054901123
d4aed439bdc854a56fc6cc3bdb986775w7hxw	0.47000521421432495
ed591299b162a19ff77f0479495831b31hl1q	0.3036041557788849
eebdee08b0f67283126ef60307f49680sb9va	0.40137118101119995

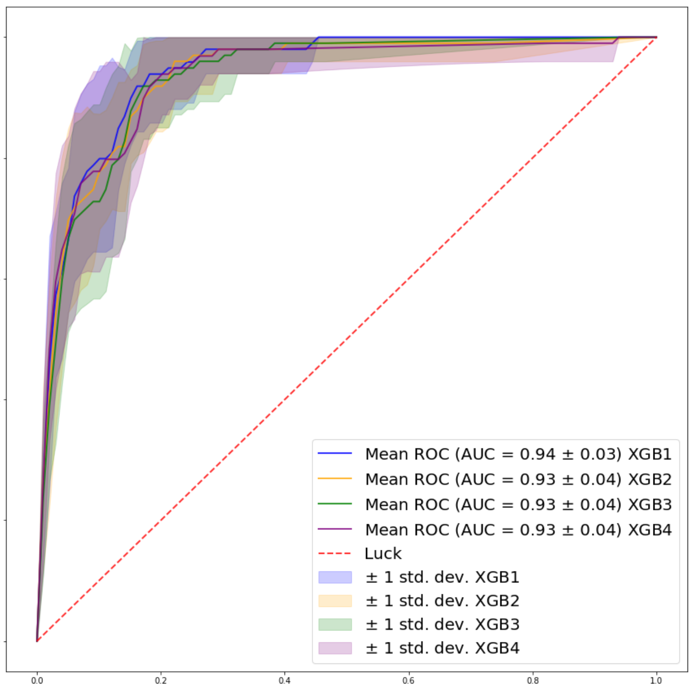
Therefore, a **rank averaging method** [18] is used here for 4 XGBoost predictors with different parameters, a small improvement has been achieved with **Private Score: 0.91184** and **Public Score: 0.89741**

submission_2018-7-25_5_ens.csv 0.91184 0.89741
11 hours ago by BoQu
add submission details

Results

Model Evaluation and Validation

XGBoost should be a relative robust algorithm, here I tried to compare the output of 4 XGBoost models with different parameters, with the ROC curves and AUC scores. Here cross-validation of 10 k-folds are used.



We can see from the figures above, that these 4 XGBoost has similar curves, standard deviation from cross validation and similar scale of AUC scores too.

However, I think due to the imbalance of train sample, the overfitting problem is still an issue for now, but with XGBoost algorithm we still got the same trend of improvement by looking at the cross-validation score.

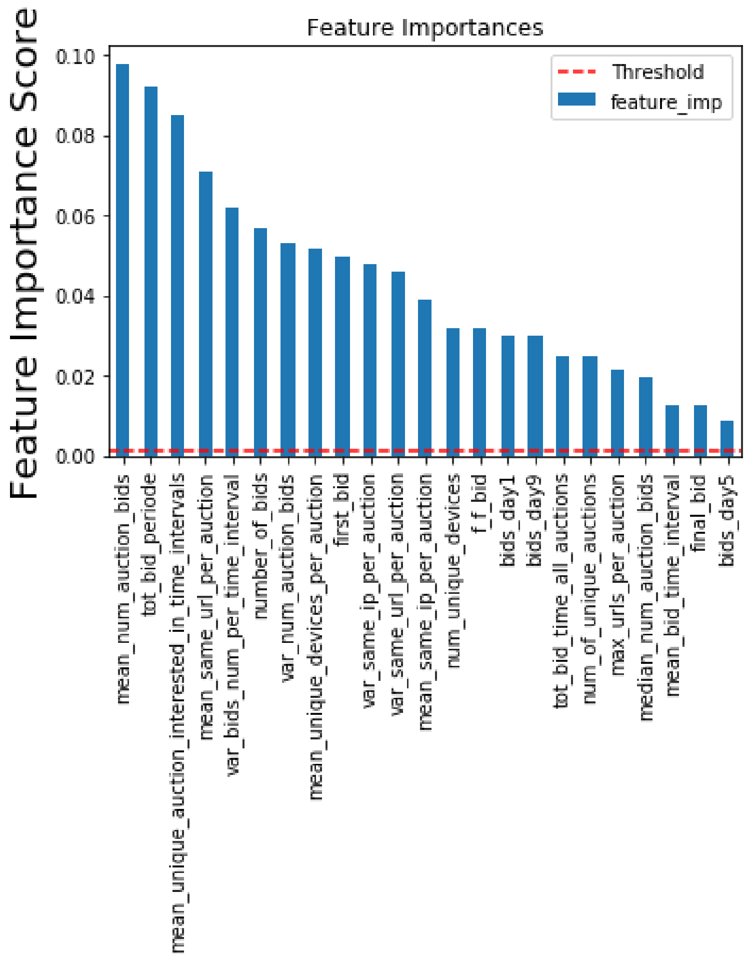
Justification

It is not fair to say that XGBoost could definitely out-performance than another ensemble algorithm. In fact, the 2nd place participant used RandomForest for his match and reached the LB score of nearly 0.94. The most important element of this project is the feature construction, not the choice of certain algorithm. It appears that only the good features can result in better performance.

Conclusion

Free-Form Visualization

The importance of different features is shown below:



We can conclude that the importance score is somewhat associated with the PSI score between human and robot in the data exploration step. Yet it seems that I still have a lot of space for improvement by finding more significant features, so that my LB score can be further leveled up.

Refection

In this project, I was trying to solve a binary classification problem with highly imbalance train sample, and all the features for training should be created by my own. This should be the most difficult part. I went through numerous trails with the feature selection, parameter tuning and ensembling part, but none of them seems to have a notable impact as the features themselves.

The result by now is better than the low standard benchmark but the further improvements should still be focus on the features, this report can be considered as the 'so far' result that I could achieve.

In this project, the analysis techniques of the features is very interesting to me, even through they did not make a huge different due to the lack of enough representative features, they are valuable knowledge to understand. Moreover, as my first Kaggle project, it builds up a pipeline for me to engage more data mining project in the future.

Improvement

I think that the most crucial improvement should be made in features. I truly would like to ask for the advices from Udacity reviewers and I am thirst for some help. Meanwhile, some tips from the discussion page of Kaggle could also do great help, however I avoid to read in the former participant experience or code before the pass of my capstone project.

Reference

[1] Garbage in, garbage out: https://en.wikipedia.org/wiki/Garbage_in,_garbage_out

[2] Receiver operating characteristic: https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[3] Kaggle project website: https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot

[4] Population stability index: https://www.listendata.com/2015/05/population-stability-index.html

[5] Numpy log: https://docs.scipy.org/doc/numpy/reference/generated/numpy.log.html

[6] MinMaxScaler: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[7] XGBoost: https://xgboost.readthedocs.io/en/latest/#

[8] Gradient Boosting: https://machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/

[9] Random Forest: https://en.wikipedia.org/wiki/Random_forest

[10] Adaboost: https://en.wikipedia.org/wiki/AdaBoost

[11] Extra-Trees: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.65.7485&rep=rep1&type=pdf

[12] Scikit-learn: http://scikit-learn.org/stable/index.html

[13] dmlc XGboost: https://github.com/dmlc/xgboost

[14] Seaborn: http://seaborn.pydata.org/

[15] Matplotlb: https://matplotlib.org/

[16] Population stability index: http://ucanalytics.com/blogs/population-stability-index-psi-banking-case-study/

[17] Guide to parameter tuning in XGBoost:
https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

[18] Kaggle ensembling guide: https://mlwave.com/kaggle-ensembling-guide/