

ISyE 6740: Computational Data Analysis

Project Report

Facebook: Bot or Not Classification

Alekhya Pyreddy
903211489

Himadeep Reddy Reddivari
903208781

Kirthana Hampapur
903198301

Contents

- Abstract4
- Keywords4
- Overview of the report4
- Introduction.....5
 - Problem Description and Dataset5
- Feature Engineering and Pre-processing.....6
- Classification Models and Results 17
 - Initial Model Development..... 17
 - Handling Class Imbalance 21
 - Ensemble Approach – Stacking 23
 - Two-Stage AdaBoost (TSAdaBoost)..... 24
 - Two-Stage AdaBoost Results 27
- Conclusions and Future Scope 28
- Workload Allocation 28
- References 28

List of Figures and Tables

Figure 1: Raw data sets	5
Figure 2: Raw bid data	6
Figure 3: Device Count for Humans and Bots	7
Figure 4: Median bid time for humans and bots	8
Figure 5: Mean time difference for humans and bots.....	9
Figure 6: Standard deviation of time difference for humans and bots	10
Figure 7: Number of bids for humans and bots.....	11
Figure 8: Country count for humans and bots.....	12
Figure 9: Auction participation for humans and bots.....	13
Figure 10: Number of auctions for humans and bots.....	14
Figure 11: Item count for humans and bots	15
Figure 12: Bot bids aggregated by country	16
Figure 13: Final predictors	16
Figure 14: Feature selection for logistic regression.....	17
Figure 15: ROC curve for SVM.....	18
Figure 16: Tuning the gamma for RBF Kernel and cost parameter for a binary class SVM.....	18
Figure 17: Size and decay parameter tuning for Neural Network Classifier	19
Figure 18: Tuning the number of trees in Random Forest	20
Figure 19: Tuning parameters for AdaBoost.....	20
Figure 20: Pseudo-code for the SMOTE algorithm	22
Figure 21: Pseudocode of AdaBoost	25
Figure 22: Flowchart of TSAdaBoost.....	26
Figure 23: Plot for stability analysis of our new TSAdaBoost algorithm	26
Table 1: Optimal tuning parameters and Model Performance in AUC	21
Table 2: Unconstrained and constrained weight obtained from stacking approach	24

Abstract

This project will present a way to detect bots in online auctions. The dataset is provided as part of Facebook recruiting competition on Kaggle. The data is not clean and in raw format has 7.6 billion alphanumeric observations with 0-1 response. One main point of this project is feature extraction. Various features were brainstormed and extracted by making a database in SQL and using RSQLite for obtaining statistics. Training dataset and testing dataset was prepared with the relevant features useful for prediction. Base models like Logistic Regression, Support Vector Machines, Neural Networks, Random Forests and Adaptive Boosting are implemented for initial prediction purposes. The results from base models justified another evident problem in the dataset. The training dataset has just 4% of the observations that are bots. We explore techniques to handle class imbalance and specifically use the Synthetic Minority Over-Sampling Technique (SMOTE) with under sampling from majority class to see if it can improve the results. No improvement in the results was observed and hence we adopted the ensemble learning approach of stacking for combining predictions of base models to improve AUC. AUC increased from 0.87 from best base model to 0.89 by using stacking. Further, to improve the AUC we explored more possibilities of other machine learning algorithms. In this process, we explain a new Two-Stage AdaBoost algorithm (TSAdaBoost) and implement it on our dataset with Random Forests and Neural Networks as the base models in the first stage and Adaptive Boosting in the second stage. TSAdaBoost improved the AUC from 0.89 to 0.91, thereby placing us in the top 25 on the private leaderboard for this completion on Kaggle. Stability analysis is conducted for 100 runs of TSAdaBoost to validate the algorithm.

Keywords

Bot-or-not classification, Feature Extraction, Class Imbalance, Synthetic Minority Over-Sampling Technique (SMOTE), Ensemble Learning – Stacking, Two-Stage Adaptive Boosting (TSAdaBoost)

Overview of the report

The Introduction section presents a brief description of the problem and the dataset. The Feature Engineering and Pre-processing section provides a detailed analysis of why a feature is important with appropriate evidence. Under the classification models section we explore various base models and present the results, consider class imbalance and use SMOTE to improve the AUC, adopt a stacking - ensemble learning approach and implement a new Two-Stage Adaptive boosting Algorithm after performing stability analysis. Conclusions and Future Scope are presented. The last section briefly discusses the workload allocation for the project.

Introduction

Online auctions are services where ordinary people, not professional participants, sell their items to those who are willing to pay the highest price. This system enables a competition between the potential buyers. Online auctions have expanded rapidly in the recent times with the technology boom and the superior level of connectivity that the internet provides. However, the advantages of technological innovations are accompanied by misuse and internet fraud. Auctions can be won by using robots and algorithm-based intelligence, which is unfair to the human competitors.

Customer satisfaction suffers critically because of the domination of the robots in online auctions. With auctions being hosted remotely on the internet, it is not possible to manually uncover the differences between a human bidder and robot bidder. Fundamental measures are required to mitigate these unfair practices. In this project, we seek to discover the characteristics of a robot bidder, or more colloquially known as ‘bot’.

Problem Description and Dataset

The problem at hand is to accurately classify whether bids placed online were placed by a human or a bot.

Bidder Information	Bid Information
<ul style="list-style-type: none">• Bidder id• Payment account• Address• Outcome – 0 for human, 1 for bot	<ul style="list-style-type: none">• Bidder id – alphanumeric data• Auction id – alphanumeric data• Merchandise – 10 categories• Device – phone model• Time - time that the bid is made (transformed)• IP Address - IP address of a bidder (transformed)• Country - country that the IP belongs• URL - which the bidder was referred from (transformed)

Figure 1: Raw data sets

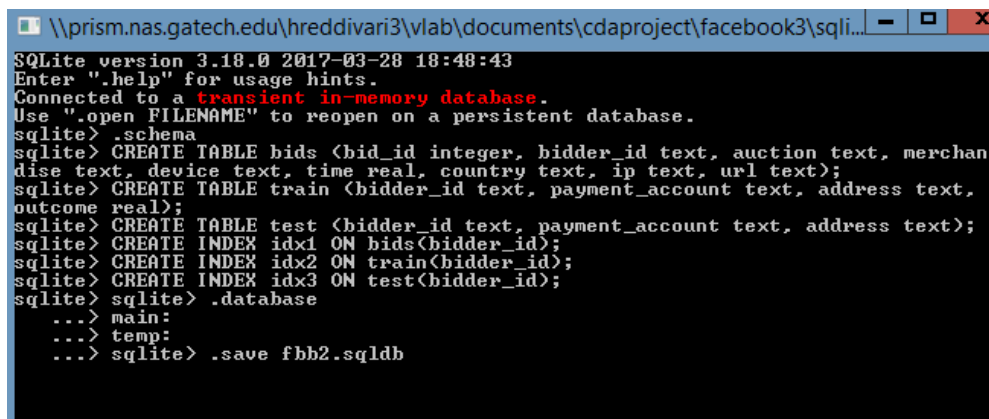
The raw data, originally derived from an online platform and obtained from [Kaggle](#) (in association with Facebook), consists of two data sets: Bid Information and Bidder Information, as shown in Figure 1. The ‘bid’ data, as indicated by the title, contains information **about 7.6 million bids** such as bidder identification number, auction identification number, merchandise type, device identification number, time at which the bid was placed, IP address of the bidder, country to which the IP address belongs and the URL which the bidder was referred from. The ‘bidder’ information including bidder identification number, payment account details, address and the binary outcome, i.e., whether the bidder was a human or a bot. An important observation that we made from the data set is that there is a lot of class imbalance that we will need to consider in our models.

Feature Engineering and Pre-processing

```
> biddersdata[which(biddersdata$bidder_id == "17a321c4a0d925ca80507effa52330ac5n5r7")][1:30,]
  bidder_id auction merchandise device time country ip url
1: 2351424 17a321c4a0d925ca80507effa52330ac5n5r7 qlnzb jewelry phone111 9631917789473684 rs 60.82.178.42 8alysey9wbk3zr
2: 2354077 17a321c4a0d925ca80507effa52330ac5n5r7 dvllu jewelry phone229 9631929105263157 ru 113.174.116.199 vasstdc27m7nks3
3: 2356802 17a321c4a0d925ca80507effa52330ac5n5r7 mtg9u jewelry phone22 9631939684210526 nl 26.251.100.185 5roro9tppf5ixp
4: 2358573 17a321c4a0d925ca80507effa52330ac5n5r7 strbn jewelry phone373 9631942526315789 cz 71.132.199.137 7pwsx2e55ckbccw
5: 2359449 17a321c4a0d925ca80507effa52330ac5n5r7 hi7or jewelry phone59 9631943947368421 za 101.172.68.99 sfpt2p7p8zj436v
6: 2367001 17a321c4a0d925ca80507effa52330ac5n5r7 tsbtt jewelry phone598 9631957421052631 de 96.110.81.233 vasstdc27m7nks3
7: 2368100 17a321c4a0d925ca80507effa52330ac5n5r7 lwgcc jewelry phone6 9631959315789473 uk 6.192.198.142 5roro9tppf5ixp
8: 2371667 17a321c4a0d925ca80507effa52330ac5n5r7 strbn jewelry phone300 9631965473684210 ph 111.3.138.93 h26spjh3n5pfxyu
9: 2373285 17a321c4a0d925ca80507effa52330ac5n5r7 lwgcc jewelry phone300 9631968315789473 ph 111.3.138.93 h26spjh3n5pfxyu
10: 2375435 17a321c4a0d925ca80507effa52330ac5n5r7 uadbw jewelry phone17 9631972315789473 ph 129.17.66.57 zbvszu3dnk70nk6
11: 2385466 17a321c4a0d925ca80507effa52330ac5n5r7 9ul86 jewelry phone469 9631991315789473 uk 6.192.198.142 h26spjh3n5pfxyu
12: 2390878 17a321c4a0d925ca80507effa52330ac5n5r7 strbn jewelry phone129 9632001894736842 cz 159.253.145.159 95ribz9d5cy7c1e
13: 2397251 17a321c4a0d925ca80507effa52330ac5n5r7 lx0hm jewelry phone33 9632015947368421 ph 15.220.160.145 vasstdc27m7nks3
14: 2397823 17a321c4a0d925ca80507effa52330ac5n5r7 bf7f2 jewelry phone1586 9632017210526315 us 157.12.54.124 g5r5308duswugad
15: 2406261 17a321c4a0d925ca80507effa52330ac5n5r7 9ul86 jewelry phone212 9632036263157894 uk 117.22.153.104 77piptyiezdpct9
16: 2410890 17a321c4a0d925ca80507effa52330ac5n5r7 c1my1 jewelry phone189 9632047526315789 my 11.139.116.12 vasstdc27m7nks3
17: 2412380 17a321c4a0d925ca80507effa52330ac5n5r7 fukqw jewelry phone1586 9632051421052631 us 133.183.43.4 5roro9tppf5ixp
18: 2417033 17a321c4a0d925ca80507effa52330ac5n5r7 cz87a jewelry phone1861 9632062842105263 ru 236.201.135.150 9bur171b8encqbo
19: 2417702 17a321c4a0d925ca80507effa52330ac5n5r7 9ul86 jewelry phone76 9632064526315789 ua 179.140.215.143 vasstdc27m7nks3
20: 2423098 17a321c4a0d925ca80507effa52330ac5n5r7 uadbw jewelry phone212 9632078210526315 uk 117.22.153.104 9168a3122fv7v3s
21: 2423117 17a321c4a0d925ca80507effa52330ac5n5r7 uadbw jewelry phone212 9632078263157894 uk 6.192.198.142 9168a3122fv7v3s
22: 2423532 17a321c4a0d925ca80507effa52330ac5n5r7 u9gsl jewelry phone53 9632079421052631 th 12.83.18.21 mfbwizdp1z3z98j
23: 2425235 17a321c4a0d925ca80507effa52330ac5n5r7 ip071 jewelry phone252 9632083736842105 bn 167.65.234.108 vasstdc27m7nks3
24: 2431476 17a321c4a0d925ca80507effa52330ac5n5r7 dvllu jewelry phone45 9632099157894736 ru 63.225.63.156 vasstdc27m7nks3
25: 2432624 17a321c4a0d925ca80507effa52330ac5n5r7 3klk9 jewelry phone252 9632102000000000 bn 167.65.234.108 9168a3122fv7v3s
26: 2433780 17a321c4a0d925ca80507effa52330ac5n5r7 lx0hm jewelry phone21 9632104842105263 by 81.54.92.252 vasstdc27m7nks3
27: 2433891 17a321c4a0d925ca80507effa52330ac5n5r7 3klk9 jewelry phone252 9632105157894736 bn 167.65.234.108 9168a3122fv7v3s
28: 2434784 17a321c4a0d925ca80507effa52330ac5n5r7 9ul86 jewelry phone258 9632107473684210 ph 233.172.46.204 vasstdc27m7nks3
29: 2436725 17a321c4a0d925ca80507effa52330ac5n5r7 9ul86 jewelry phone5 9632112526315789 ph 41.81.80.193 vasstdc27m7nks3
30: 2437534 17a321c4a0d925ca80507effa52330ac5n5r7 nxf13 jewelry phone1382 9632114842105263 tr 203.210.192.208 vasstdc27m7nks3
```

Figure 2: Raw bid data

The raw data cannot be directly used as input features to machine learning models. Most of the information is alphanumeric data (as shown in Figure 2) which does not make much sense, exclusively. The concepts of feature extraction need to be employed to derive the features from the data set and then used as inputs to the models. An important point to note while extracting features is that there is no information about how the classification (whether human or robot) of a bid in the training set has been made. There could be errors in this classification, but since we have no means of discerning this, we have deemed it to be out of the scope of this project. Performing computation on this 7.6 million observations data is not convenient in R. Holding all the data on memory is possible but some other options could be better. We explored the possibility of creating an SQL database with this data and using RSQLite library to query from the database and generate relevant features. The same is shown as a screenshot from the SQL command prompt as below.



```
\\prism.nas.gatech.edu\hreddivari3\vlab\documents\cdaproject\facebook3\sqli...
SQLite version 3.18.0 2017-03-28 18:48:43
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .schema
sqlite> CREATE TABLE bids (bid_id integer, bidder_id text, auction text, merchan
dis text, device text, time real, country text, ip text, url text);
sqlite> CREATE TABLE train (bidder_id text, payment_account text, address text,
outcome real);
sqlite> CREATE TABLE test (bidder_id text, payment_account text, address text);
sqlite> CREATE INDEX idx1 ON bids(bidder_id);
sqlite> CREATE INDEX idx2 ON train(bidder_id);
sqlite> CREATE INDEX idx3 ON test(bidder_id);
sqlite> sqlite> .database
...> main:
...> temp:
...> sqlite> .save fbb2.sqldb
```

Some features are easily identifiable such as the number of auctions placed by a single bidder, the number of devices and IP addresses used for a single bidder id number and the number of distinct countries that the bidder is seen to bid from. All these quantities are high for robots and these can be used as flags to identify suspicious behavior.

Some other features require more clarity before they can be effectively used as features in modeling. Time at which the bid was placed, as a stand-alone quantity does not contribute much information to the classification problem. Hence, we express time in terms of time between consecutive bids, which, intuitively should be lower for a bot. Similarly, the country of the bidder, coupled with country crime statistics leverages the demographic information on the potential to commit online fraud.

In the interest of understanding the behavior of bots, we plot histograms of the data, grouped by bidder type, i.e., whether human or bot.

Firstly, we observe, from Figure 3, that the device count is significantly more varied for bots than it is for human bidders. While very few humans bid from over 250 devices, a suggestively huge portion of the bots bid from over 250 devices.

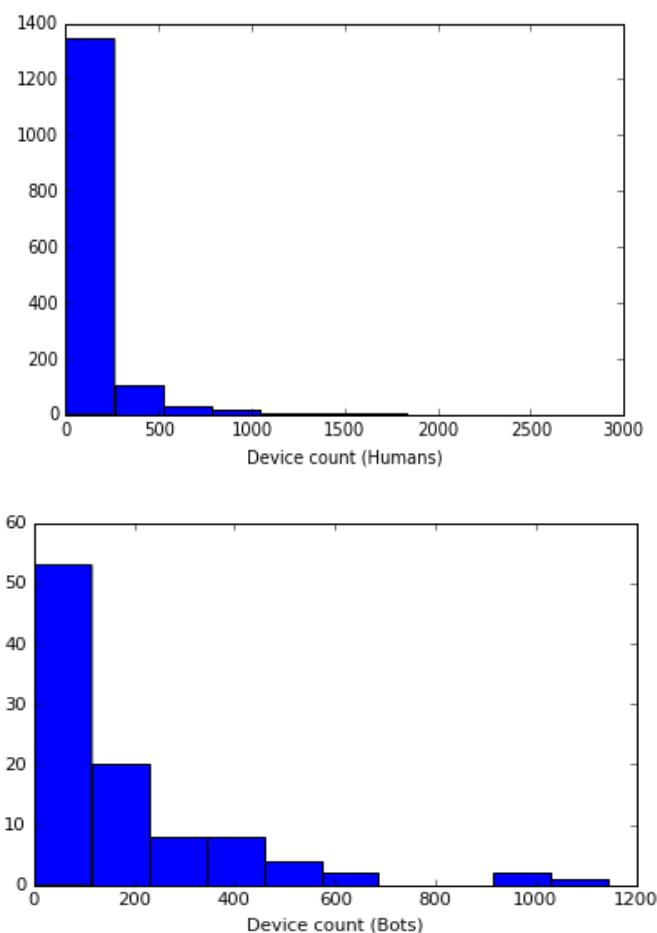


Figure 3: Device Count for Humans and Bots

The median of the bid times, grouped by category is seen in the Figure 4. It is apparent that the human bidders have an approximately normal distribution of median bid times. The median is distinctly located at the 50th percentile for the human bidders, whereas, for the bots, the median bid time is distributed throughout, with a sharp peak at the 50th percentile.

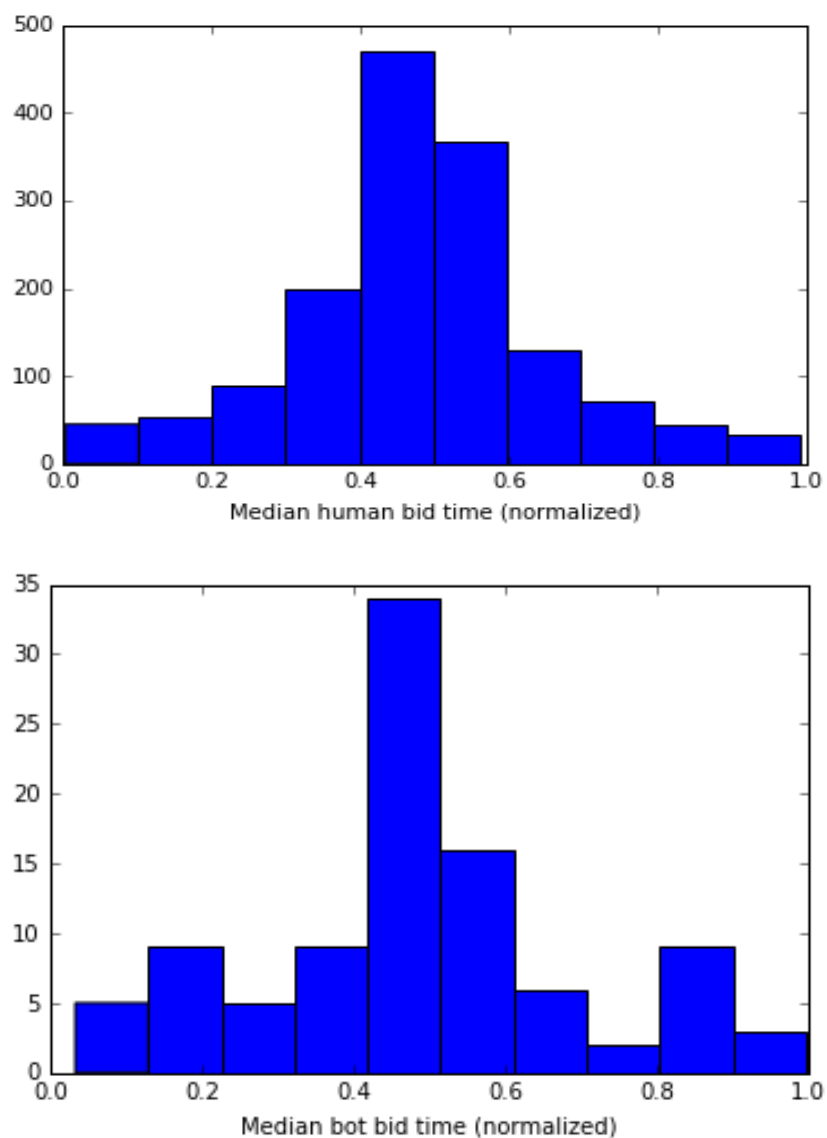


Figure 4: Median bid time for humans and bots

The mean and standard deviation of the difference in times between the bids is then studied, as shown in Figure 5. These graphs highlight the difference in the bidding strategies of humans and bots. The mean time difference for majority of the bots is concentrated towards the smallest division of time. This mean is more varied for the humans; we can thus infer that the bots bid more frequently than the humans and thus, their mean time difference is almost negligible.

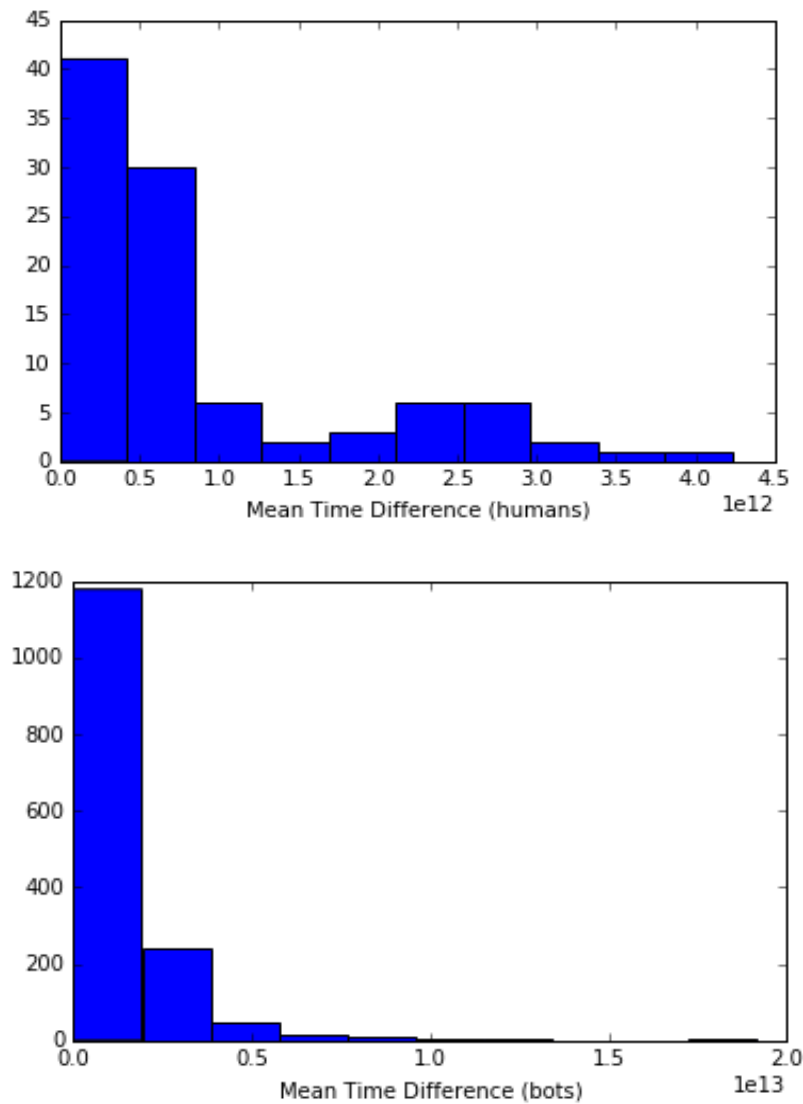


Figure 5: Mean time difference for humans and bots

When we look at the histograms (as seen in Figure 6) that compare the standard deviation, we can confirm our perception about the time difference variation.

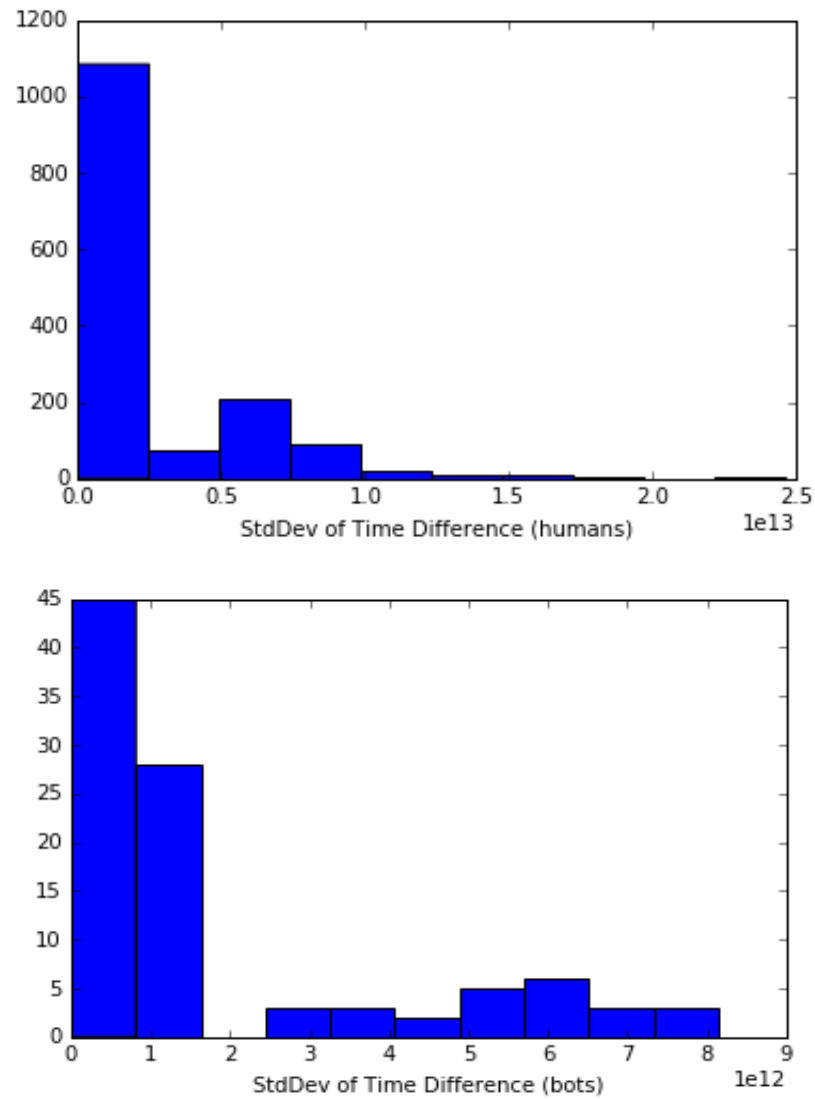


Figure 6: Standard deviation of time difference for humans and bots

From Figure 7, we conclude that the number of bids made by the human bidders is less varied than the number of bids made by the bots. This is concurrent to our observation that bots bid a lot and they bid fast. This strategy is followed by all bots and this, we believe, is the key to their advantage over human bidders.

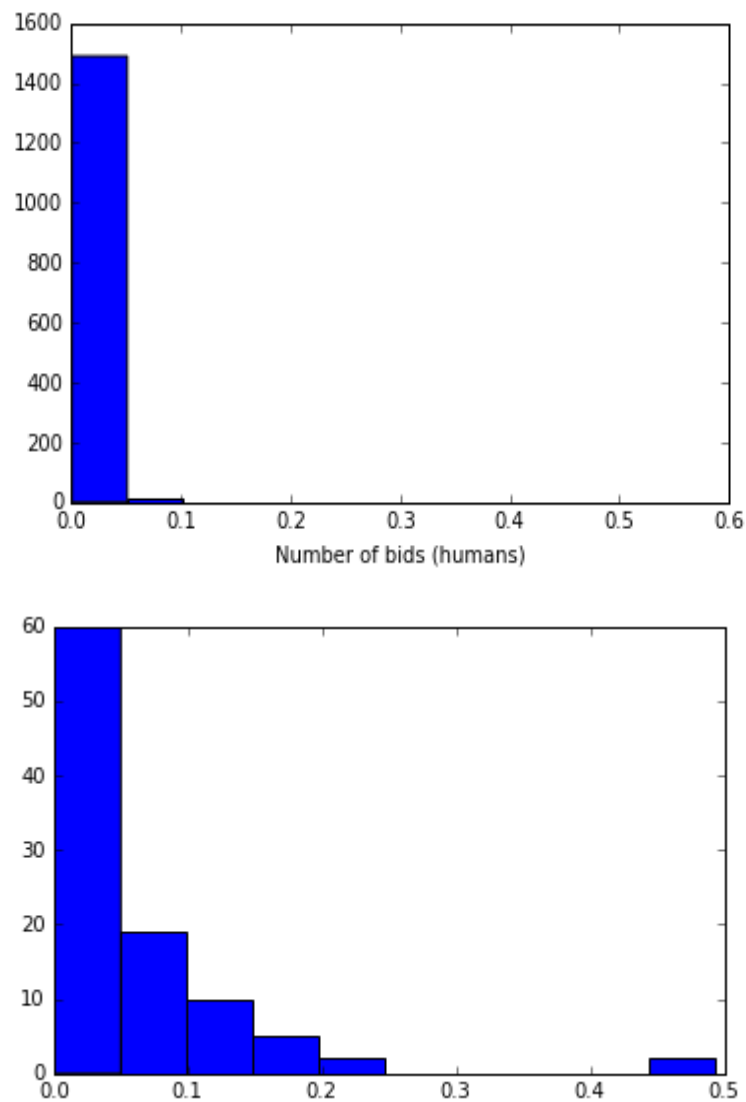


Figure 7: Number of bids for humans and bots

Country count is another interesting feature (in Figure 8) which highlights the number of countries that a single bidder belonged to, i.e., the number of countries that the bidder made the bid from. Majority of the human bidders placed a bid from less than 10 countries, while the bots bid from more number of countries, which implies that the IP address (from which the country was identified) was manipulated.

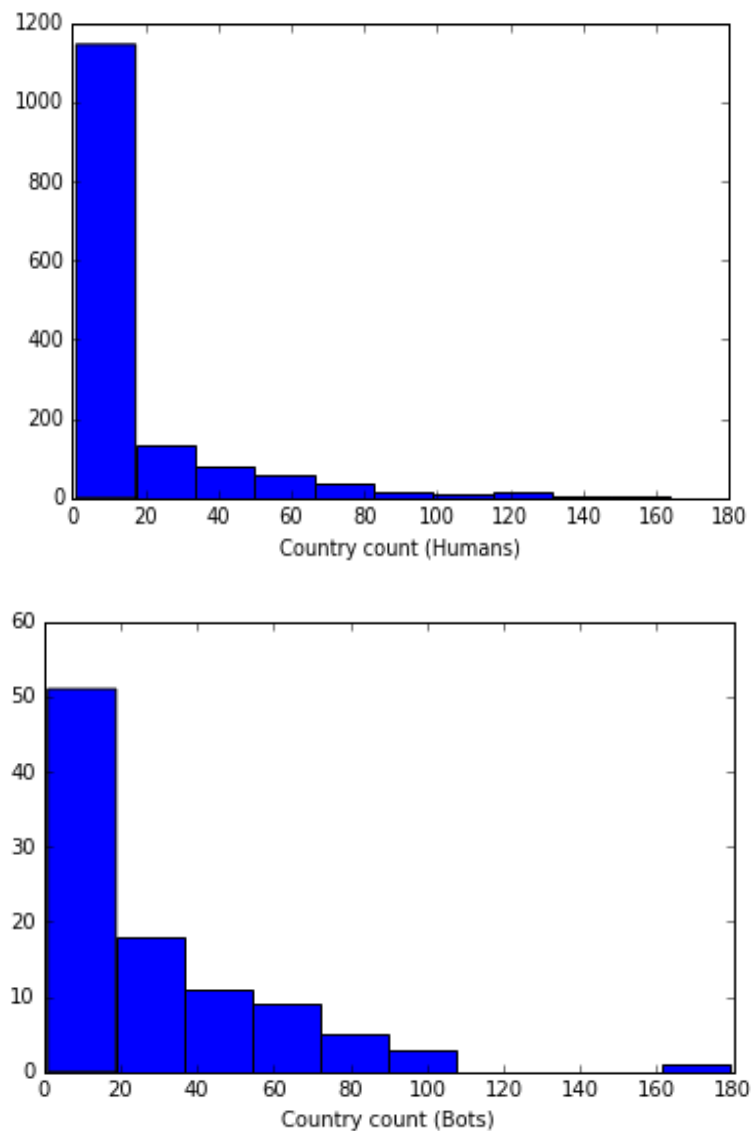


Figure 8: Country count for humans and bots

The auction participation time was calculated for human bidders and bots using normalization, as shown in Figure 9. Most of the human bidders participated in less than 50% of the total auction time and many of them participated in less than 10% of the auction time. This is indicative of human behavior where he/she places a bid and then exits. The bots, however, participate in varied portions of time with a considerable number of bots participating in over 50% of the auction time.

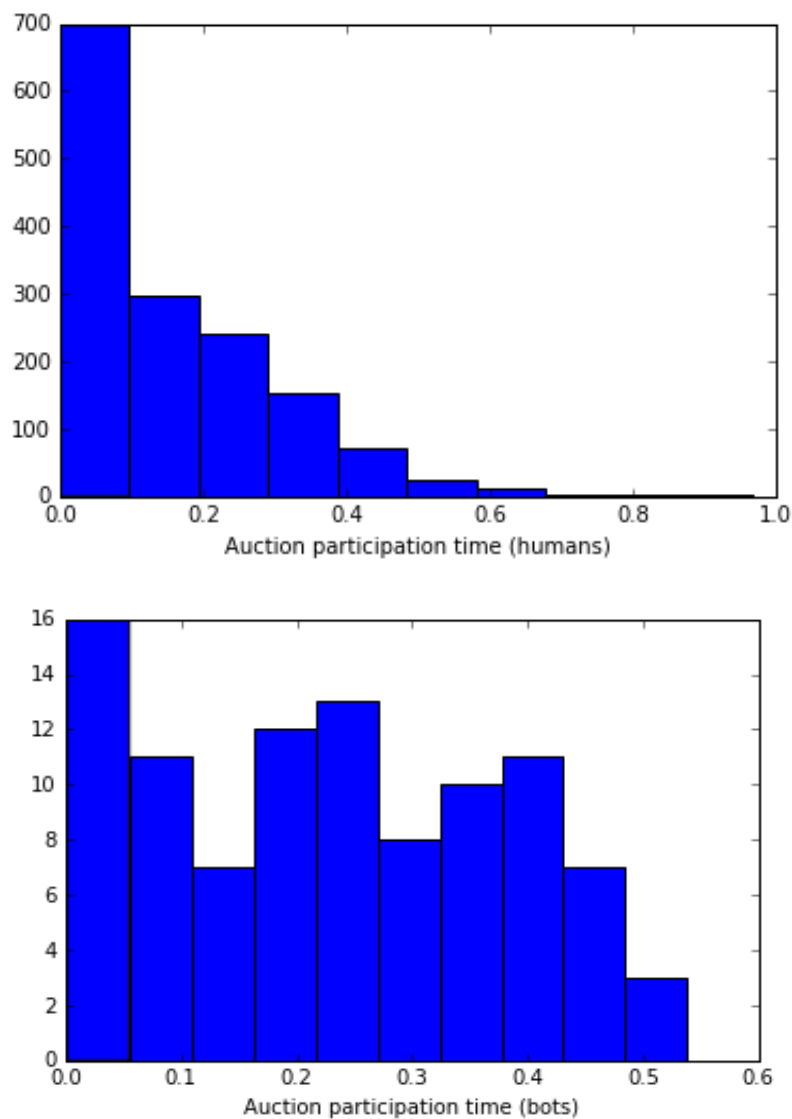


Figure 9: Auction participation for humans and bots

Bots also participate in more auctions than a human bidder as seen from the below histogram (Figure 10). Some bots are seen to bid in more than 800 auctions.

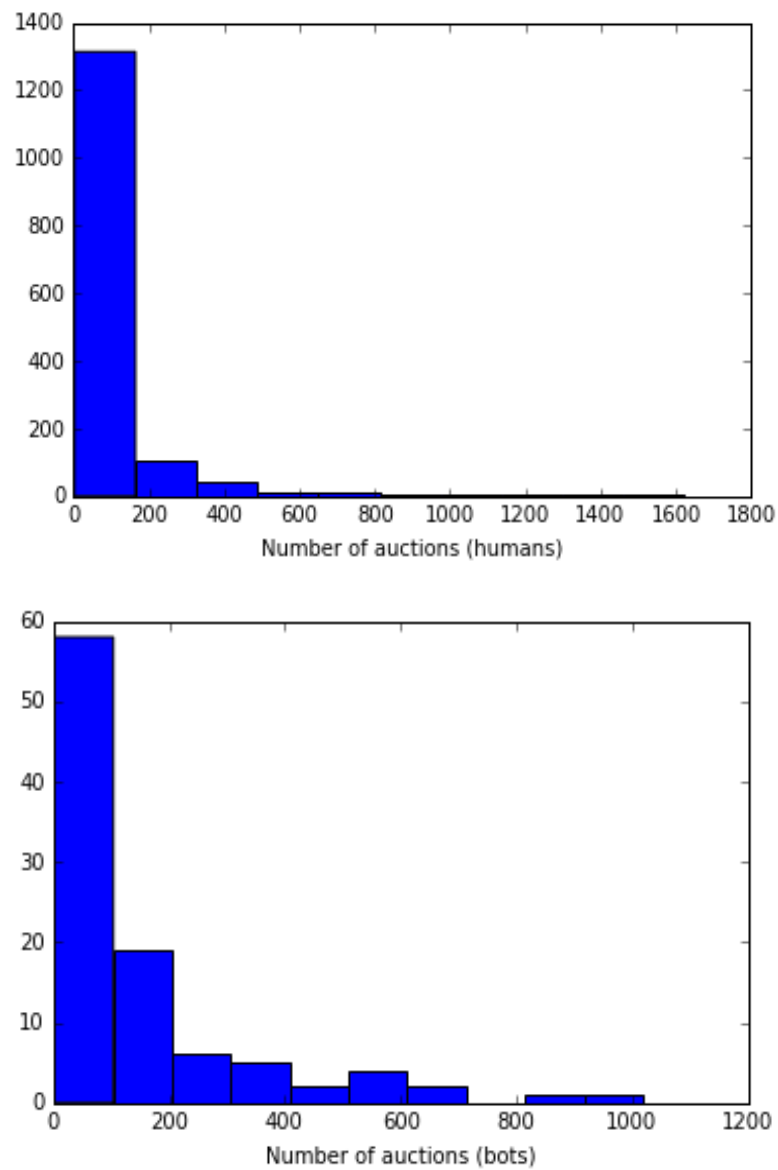


Figure 10: Number of auctions for humans and bots

Item count is an interesting quantity that we studied (as shown Figure 11) in our feature extraction process which grouped the bids placed by the human bidders and the bots by the merchandise that they bade for. Both types of bidders are interested in sporting goods, mobiles and jewelry, however, human bidders prefer mobiles over jewelry whereas bots bid for both almost equally.

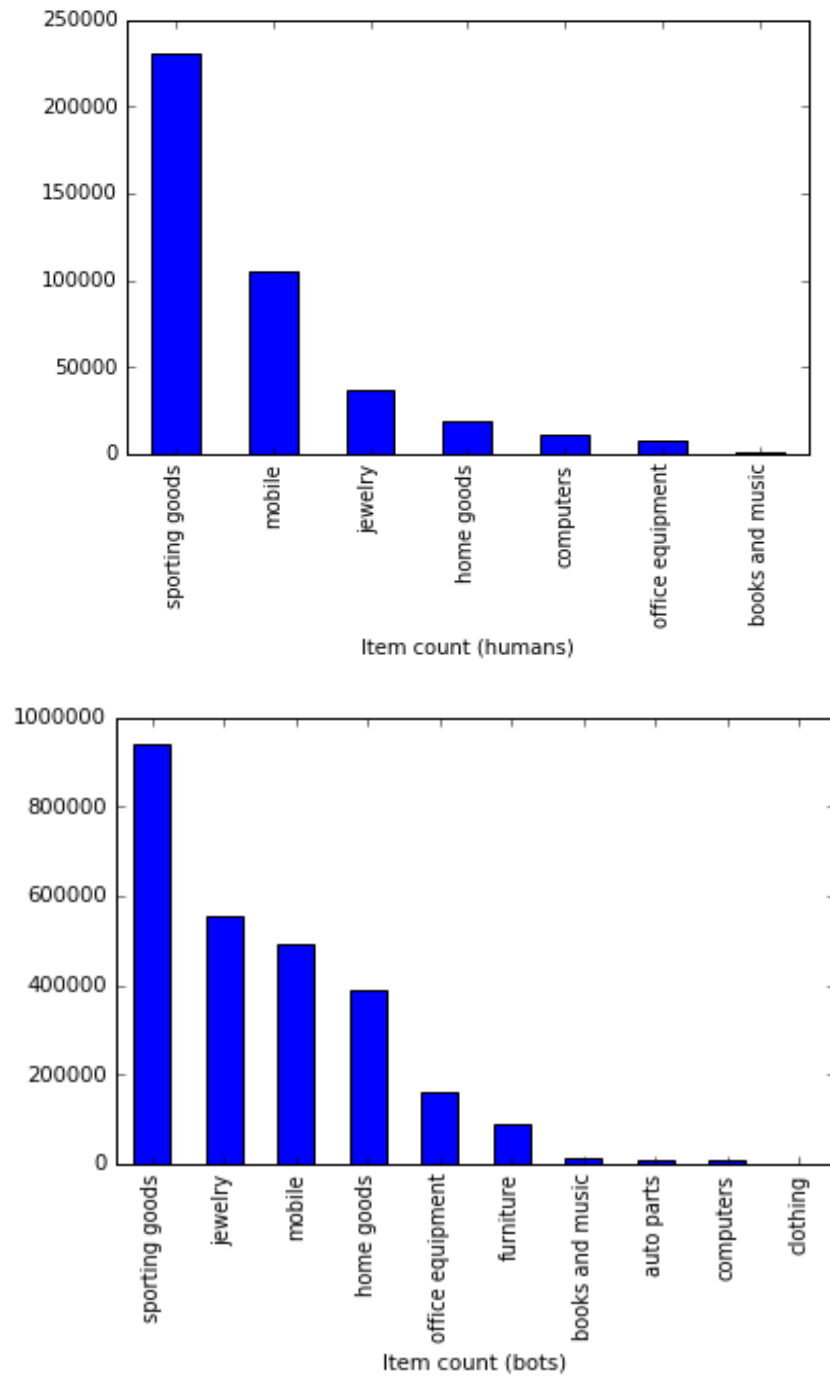


Figure 11: Item count for humans and bots

Bids placed by bots were also grouped by the country and plotted using a histogram as seen in Figure 12. It was seen that a certain country indicated by 'mo' had the highest number of bot bids, considerably higher than the other countries' bot bids.

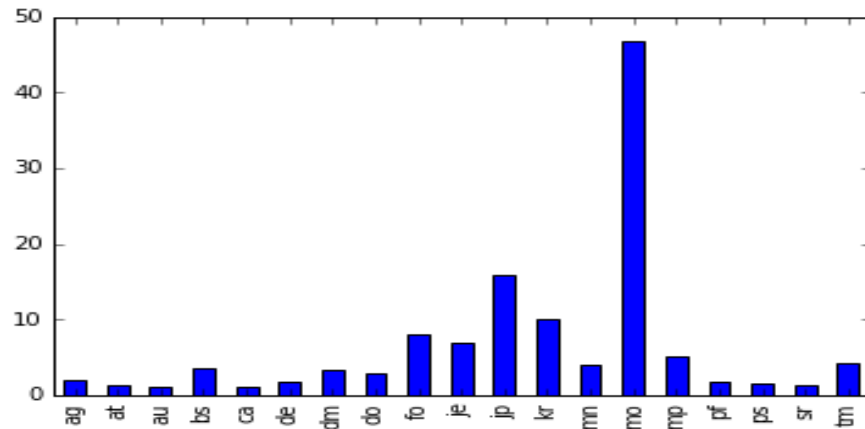


Figure 12: Bot bids aggregated by country

As we can expect, bots are automated software and hence can be more vigilant and persistent in outbidding the current highest bid. Hence, we included features such as response time (tfast) that capture this behavior.

Finally, the features used were as follows:

1. Bid count
2. Country count
3. IP – mean, median
4. Link (url) – mean, median
5. Device count
6. Merchandise (converted to numerical identifier)
7. Time difference between consecutive bids of a user– mean, standard deviation, median, minimum
8. Response time for a user, i.e., time difference between a user's bid and the bid placed just before it
9. Maximum number of bids
10. Crime statistics of country – mean, median, maximum – extracted from a separate database from SQL

The predictors used for further modeling are shown in Figure 13.

```
> head(myPredictors)
      bid_count country_count ip_mean ip_median link_mean link_median device_count merchandise tmean   tsd
[1,] 0.6931472           1 0.6931472 0.6931472 0.6931472 0.6931472           1      6 32.65626 32.30835
[2,] 0.6931472           1 0.6931472 0.6931472 0.6931472 0.6931472           1      8 32.65626 32.30835
[3,] 4.9558271          16 0.9842019 0.6931472 1.0140549 0.6931472          67      1 27.03079 29.07804
[4,] 1.3862944           2 0.6931472 0.6931472 0.6931472 0.6931472           3      9 30.94463 31.28923
[5,] 6.2989492          72 3.2046576 0.6931472 0.7503056 0.6931472         165      6 23.94505 24.69975
[6,] 3.1780538          10 0.7444405 0.6931472 0.7444405 0.6931472          16      8 27.05518 27.24176

      tmedian   tmin   tfast max_bids max_country_count crime_mean_p crime_median_t crime_max_p crime_max_t
[1,] 32.65626 32.65626 0.0000000 0.6931472           1 0.05039277           2 0.05039277           2
[2,] 32.65626 32.65626 0.0000000 0.6931472           1 0.05407473           2 0.05407473           2
[3,] 24.75143 20.08141 0.9214286 1.0665729          16 0.18141547           3 0.54118408           5
[4,] 30.94463 24.71322 0.5000000 0.6931472           2 0.29682565           4 0.47601808           5
[5,] 23.36608 17.77883 0.6845018 3.4947234          72 0.14953025           3 0.80239521           5
[6,] 26.88719 18.47197 0.6363636 0.7932306          10 0.09710505           3 0.20400406           4
> |
```

Figure 13: Final predictors

Classification Models and Results

Initial Model Development

Once the features are extracted, we can feed them as the predictor variables to our classification algorithms. Various classification algorithms were trained on the data and 10-fold cross validation was used for tuning their respective parameters and estimating their test error performance.

We trained a Logistic Regression model using forward, backward and step wise variable selection. Even the best logistic model performs only slightly better than random guessing, possibly due to its inability to capture non-linearity of the decision boundary. Support Vector Machines with radial kernel also performs poorly, possibly due to the existing class imbalance. Ensemble methods like Random Forest and AdaBoost exhibited the best performance on both training and testing data. Below results for various tuning parameters and their performance characteristic curves; the quantitative results have been tabulated and compared at the end of this section. The AUC values are visualized using “corrplot” in R and using ellipsoid method. Area of the ellipses in the following grids is inversely proportional to the AUC. This implies that the optimal tuning parameters are the values corresponding to the straight line in the grid plot.

Logistic Regression

In our logistic regression model, we find that the backward selection with k=5 yields the best performance as seen in Figure 14.

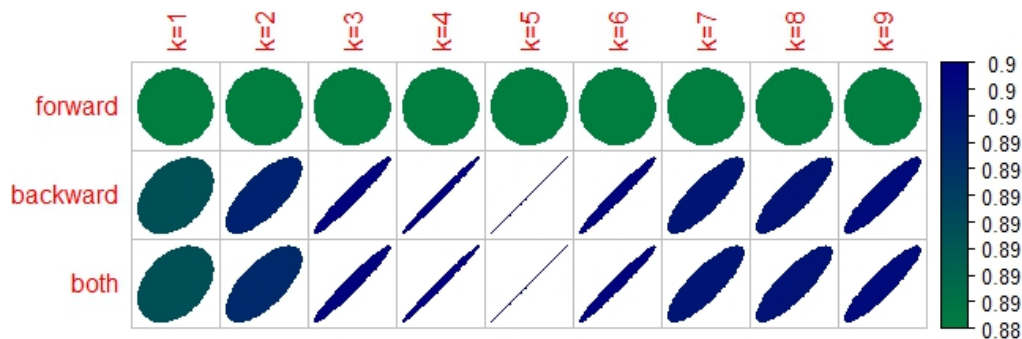


Figure 14: Feature selection for logistic regression

Support Vector Machines

The ROC curve shown in Figure 15 is an indication of the poor performance of the SVM model for this classification problem.

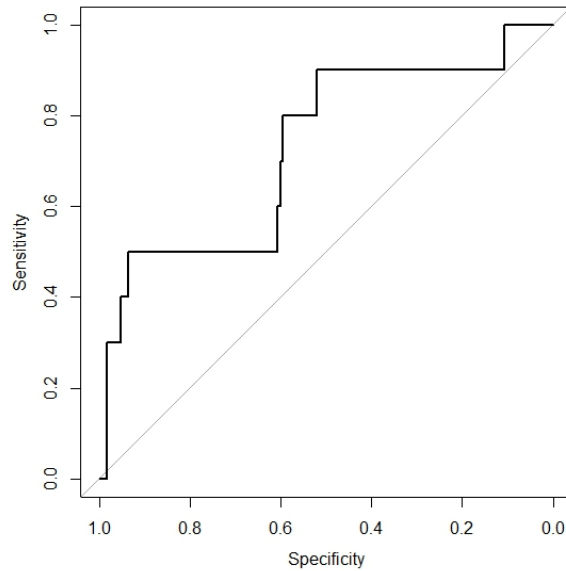


Figure 15: ROC curve for SVM

The selection of tuning parameters (gamma, cost) is made using the results from Figure 16 as shown. Gamma is the parameter for non-linear kernels and cost is the cost associated with constraint violation in the Lagrange formulation. We see that a gamma value of 0.01 and cost value of 0.05 is the best combination for minimizing misclassification.

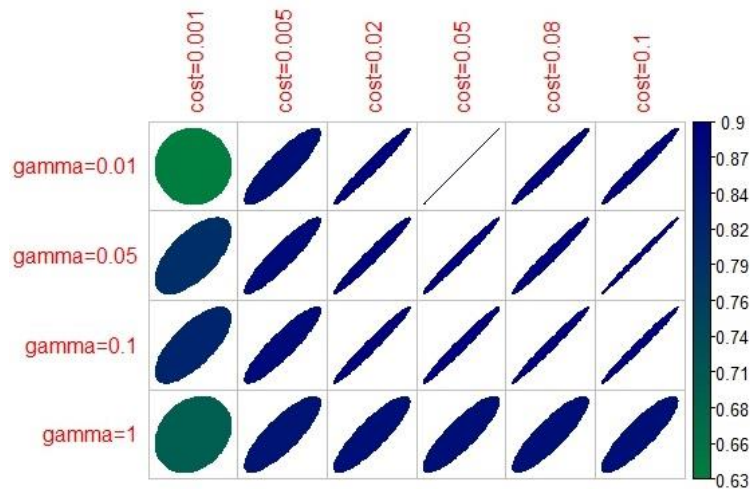


Figure 16: Tuning the gamma for RBF Kernel and cost parameter for a binary class SVM

Neural Networks

We also trained neural networks in our study and we found that the performance was poor. Relatively, a size of 2 and a decay factor of 0.00005 when used as tuning parameters for the neural network perform slightly better than any other combination of input parameters, as seen in Figure 17.

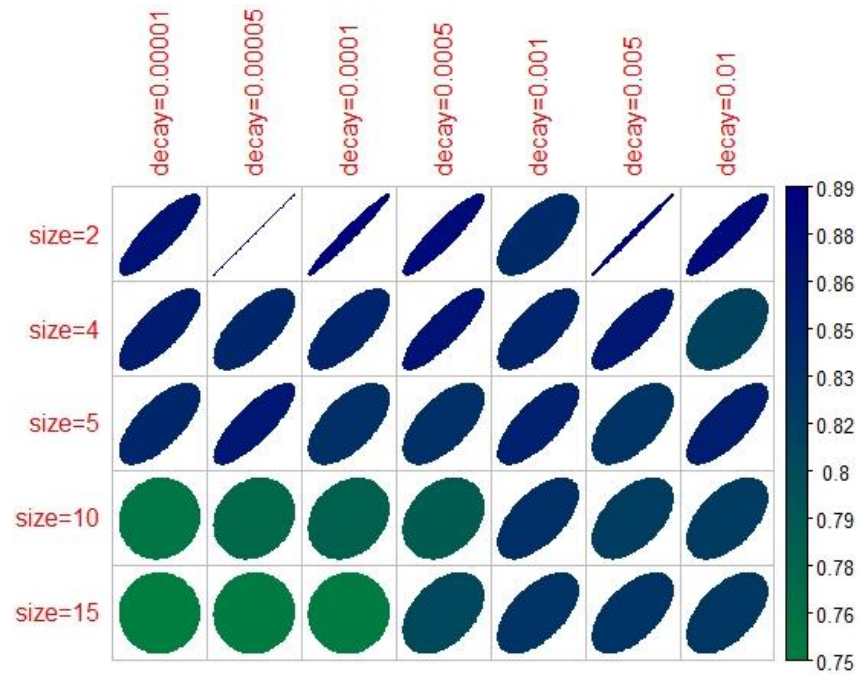


Figure 17: Size and decay parameter tuning for Neural Network Classifier

Random Forests

When random forest method was applied to the data set, the results obtained seemed promising. The performance was slightly better than that exhibited by the previous methods that we experimented with. The graph in Figure 18 shows the optimal number of trees that should be chosen. We decided to explore this method in further detail, which will be explained in the next section.

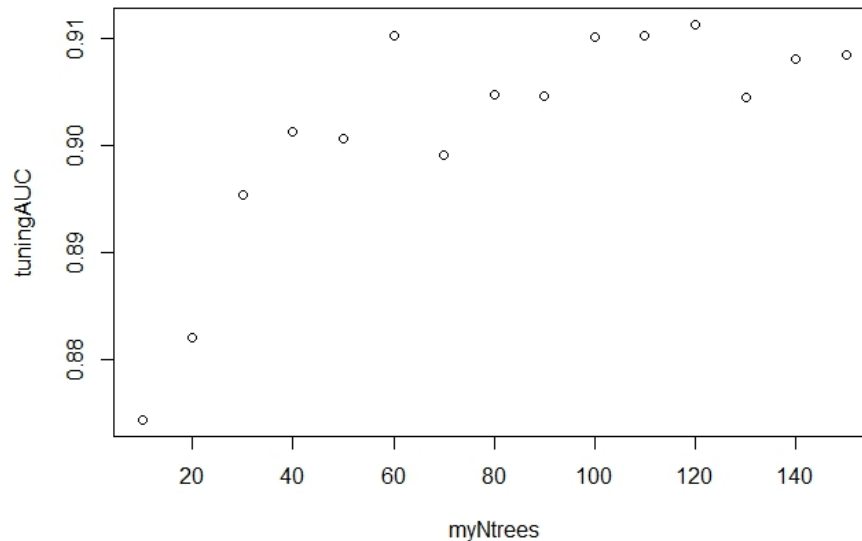


Figure 18: Tuning the number of trees in Random Forest

AdaBoost

AdaBoost also displayed favorable results with suitably-selected tuning parameters as seen in Figure 19. Because of the potential of this method to perform better, we decided to include this method in our ensemble approaches.

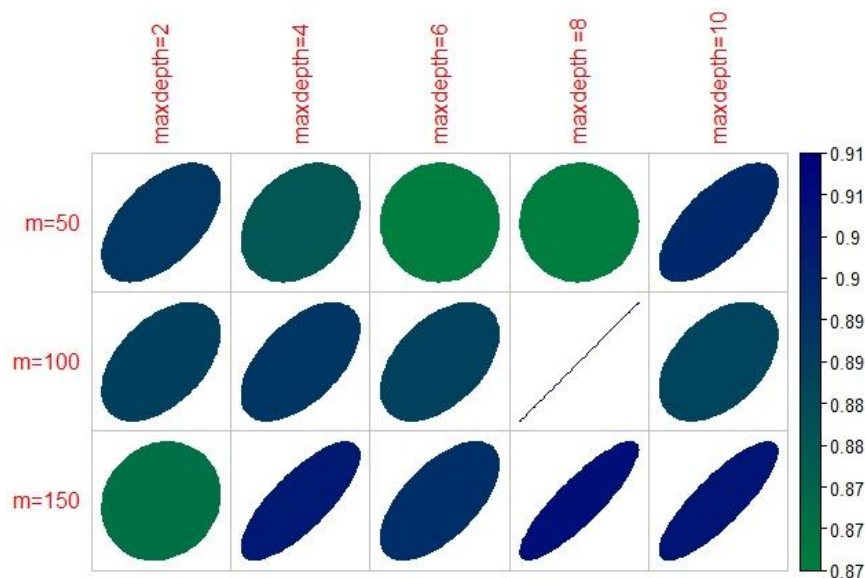


Figure 19: Tuning parameters for AdaBoost

Model Comparison

Final optimal parameters and model performance are shown in Table 1.

Table 1: Optimal tuning parameters and Model Performance in AUC

Base Models	Parameter 1	Optimal	Parameter 2	Optimal	Training AUC	Submission AUC
SVM	Gamma	0.01	Cost	0.05	0.7881597	0.6019000
Neural Nets	Decay factor	0.00005	Units in hidden layers	2	0.8795238	0.7864244
Random Forests	Number of trees	120	--	--	0.9046143	0.8606039
Logistic Regression	K	5	Type of Regression	Both	0.7971601	0.6658645
Adaptive Boosting	Number of trees used	100	Maximum Depth of tree	8	0.9122576	0.8706247

From our initial analysis of modeling the classification problem, we infer that the predominant class imbalance in the data affects the performance of some of the models above. We examined some methods to mitigate the over-generalization of classifying most of the points as belonging to the majority class i.e., human bidders.

Handling Class Imbalance

Most existing classification approaches assume the underlying training set is evenly distributed. In our case, the training dataset after feature extraction is highly imbalanced. The ratio of number of zeros to ones (humans to bots) in our training dataset is 19:1. This is perhaps the reason for poor classification results from SVM and Logistic Regression (LR) models. Another evidence that suggests looking at class imbalance is the better performance of SVM and LR on small sample size during 10-fold cross-validation. SVM and Logistic Regression contributed over 90% accuracy for 193 samples. This seems to be very good but it is observed that the actual prediction is slightly better than random guessing. This is further justified by the fact that in Neural Nets and Random Forests look at the data and cleverly decide that the best thing to do is to always predict majority class and achieve high accuracy. Thus, handling class imbalance is very crucial for our data and in this sub-section, we investigate how class imbalance can be taken care off and if it can produce better results.

There are several tactics to handle class imbalanced data. Some of them are listed as follows

- I. Collecting more data – A larger dataset might expose a different and perhaps more balanced perspective on the classes. More examples of minor classes may be useful later when we look at resampling your dataset. However, collecting data is not a feasible option in our case.
- II. Changing the performance metric – Metrics like Cohen’s Kappa or ROC curves seem to be effective when addressing accuracy in the case of class imbalanced datasets. In this Kaggle competition, the accuracy measure used for evaluating submissions is AUC and hence we cannot shift to a new analysis in our model.

- III. Resampling dataset and generating synthetic samples – You could sample empirically within your dataset or you could use a method like Naive Bayes that can sample each attribute independently when run in reverse. You will have more and different data, but the non-linear relationships between the attributes may not be preserved. These approaches are often very easy to implement and fast to run. Synthetically generating new samples from the minority class is also very effective.
- IV. Penalized models – Using penalization is desirable if you are locked into a specific algorithm and are unable to resample or you are getting poor results. However, setting the penalties for the classes is a difficult task. We would have to try a variety of penalty schemes to see what works best for our problem.

Considering the advantages, the best technique that we can use to handle class imbalance in our case is to generate synthetic samples. In this regard, we implement a technique called SMOTE – Synthetic Minority Over-Sampling Technique to see if our results can be improved. The essence of SMOTE is that a combination of over-sampling the minority class and under-sampling the majority class can achieve better classifier performance than only under-sampling the majority class. Chawla et al. [2], showed that this technique is better than varying the loss ratios in Ripper method or class priors in Naïve Bayes. The pseudo-code for the algorithm is as shown in Figure 20.

```

Input:  $T$ : Training set;  $N$  :  $N\%$  amount of synthetic samples ;  $k$ : Number of nearest
        neighbors.
Output:  $S$ : Set of synthetic samples
begin
     $X \leftarrow \text{MinorityInstances}(T)$ ;
     $n \leftarrow \text{NumberOfInstances}(X)$ ;
     $p \leftarrow \text{NumberOfVariables}(X)$ ;
     $S \leftarrow \emptyset$ ;
    if  $N < 100$  then
         $n \leftarrow (N/100) \times n$  ;                               /*  $N\%$  will be SMOTEd */
         $X \leftarrow \text{RandomSample}(X, p)$ ;
         $N \leftarrow 100$ ;
    end
     $N \leftarrow N/100$ ;
    for  $i \leftarrow 1$  to  $n$  do
         $\widehat{X}_i \leftarrow \text{KNN}(i, X)$  ;                               /* Compute  $k$  nearest neighbor for  $i$  */
        while  $N \neq 0$  do
             $\beta \leftarrow \text{RandomNumber}(1, k)$  ;                 /* chose one neighbor of  $i$  */
            for  $j \leftarrow 1$  to  $p$  do
                 $\alpha \leftarrow \text{RandomNumber}(0, 1)$ ;
                 $S_{ij} \leftarrow X_{ij} + (\widehat{X}_{i\beta j} - X_{ij}) \times \alpha$ ;
            end
             $N \leftarrow N - 1$ ;
        end
    end
    return  $S$ 
end

```

Figure 20: Pseudo-code for the SMOTE algorithm

The above algorithm is just an over-sampling approach in which the minority class is over sampled by creating “synthetic” examples rather than by over-sampling with replacement. This approach is inspired by a technique that proved successful in handwritten character recognition [3]. The minority class is over-sampled by taking each minority class sample and introducing synthetic examples along the line segments joining any or all of the k minority class nearest neighbors. The amount of over-sampling is a parameter to the algorithm. Depending upon the amount of over-sampling required, neighbors from the k nearest neighbors are randomly chosen. The synthetic examples cause the classifier to create larger and less specific decision regions. Chawla et al. discussed that it more effective to use under-sampling with SMOTE combination. The majority class is under-sampled by randomly removing samples from the majority class population until the minority class becomes some specified percentage of the majority class. This forces the learner to experience varying degrees of under-sampling and at higher degrees of under-sampling, the minority class has a larger presence in the training set.

We implement SMOTE technique combined with under-sampling of majority class to our data. We used the DMWR library in R to implement SMOTE. The data set was balanced to have equal proportions of zeros and ones. This sampled data was used to train SVM and LR and tested on the test dataset from Kaggle. **A submission AUC of 0.48 is obtained and the results showed that this technique is no better than random guessing.**

Basic models still fail but this is justified because considering an equal proportion of both the classes is not representative of the entire population. Our original dataset suggests that ones are rare events but now we are considering that both classes are equally likely. In the next section, we discuss the ensemble approach of stacking to further improve the prediction accuracy from the base models.

Ensemble Approach – Stacking

Failure of the base models and lack of significant improvement from the implementation of sampling technique of SMOTE resulted in a very poor prediction accuracy. We moved towards an ensemble learning approach to assess if improvements in prediction accuracy can be achieved. AdaBoost, with the highest AUC of 0.87 among all the base models, is used in this ensemble approach. In this sub-section, we briefly introduce stacking as an ensemble approach and explain how our stacking approach can help increase the prediction accuracy.

Before formally discussing model-averaging using stacking, we will highlight how stacking is effective. A popular solution to model averaging is linear regression of Y on $\hat{F}(x)^T = [\hat{f}_1(x), \hat{f}_2(x) \dots \hat{f}_M(x)]$, where $\hat{f}_m(x)$ are given predictions. However, there are simple examples where this does not work well. Let $\hat{f}_m(x), m = 1, 2, \dots, M$ represent the prediction from the best subset of inputs of size m among M total inputs, and then linear regression would ration the highest proportion of weight on the largest model, that is, $\hat{w}_M = 1, \hat{w}_m = 0, m < M$. The problem is that we have not considered all the models on the same footing by accounting for their complexity (the number of inputs m in this example). Stacking is a technique of doing the above without any potential issues. Let $\hat{f}_m^{-i}(x)$ be the prediction of x , using model m , applied to the dataset with the i^{th} training observation removed.

The stacking estimate of the weights is obtained from the least squares linear regression of y_i on $\hat{f}_m(x_i)$, $m = 1, 2, \dots, M$. In detail the stacking weights are given by

$$\hat{w}^{st} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^N \left[y_i - \sum_{m=1}^M w_m \hat{f}_m(x_i) \right]^2$$

The final prediction is $\sum_m \hat{w}_m^{st} \hat{f}_m(x)$. By using the cross-validated prediction $\hat{f}_m(x_i)$, stacking avoids giving unfairly high weight to models with higher complexity. Better results can be obtained by restricting the weights to be nonnegative and to sum to one. This seems like a reasonable restriction if we interpret the weights as probabilities that lead to a tractable quadratic programming problem. However, one can use any learning method, not just linear regression to combine the models.

We implement the above approach to our problem to see whether stacking can improve the model. Sum Squared Error was minimized to find the optimal weights. Weighted sum of the predictions is then used to get aggregated predictions for the training dataset. The unconstrained and constrained weights for each of the five models are as shown in Table 2.

Table 2: Unconstrained and constrained weight obtained from stacking approach

Base Model	Unconstrained Weight	Constrained Weight
SVM	0.2422	0.0947
Neural Network	-0.0098	0.0227
Random Forests	0.6582	0.6334
AdaBoost	0.1642	0.1919
Logistic Regression	0.1100	0.0571

Using the constrained weights as obtained above; predictions were made on the final dataset. **The AUC metric on the submission file improved from 0.87 to 0.89.** This is not a significant improvement considering our ranking on the public leaderboard for this completion. In view of further improving the accuracy metric, we will implement a **new two-stage AdaBoost** in next section.

Two-Stage AdaBoost (TSAdaBoost)

Boosting is a class of machine learning methods based on the idea that a combination of simple classifiers (obtained by a weak learner) can perform better than any of the simple classifiers alone. A weak learner (WL) is a learning algorithm capable of producing classifiers with probability of error strictly (but only slightly) less than that of random guessing (0.5, in the binary case). On the other hand, a strong learner (SL) is able (given enough training data) to yield classifiers with arbitrarily small error probability. An ensemble (or committee) of classifiers is a classifier build upon some combination of weak learners. The strategy of boosting, and ensembles of classifiers, is to learn many weak classifiers and combine them in some way, instead of trying to learn a single strong classifier. This idea of building ensembles of classifiers has gained interest in the last decade; the rationale is that it may be easier to train several simple classifiers and combine them into a more complex classifier than to learn a single

complex classifier. In this section, we will first introduce boosting as an algorithm, its shortcoming and then discuss the new two-stage AdaBoost algorithm implemented by us to improve the prediction accuracy.

The AdaBoost algorithm, introduced in 1995 by Freund and Schapire, solved many of the practical difficulties of the earlier boosting algorithms. The pseudocode of the algorithm is shown in Figure 21.

Given: $(x_1, y_1), \dots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \dots, T$:

- Train weak learner using distribution D_t .
- Get weak hypothesis $h_t : X \rightarrow \{-1, +1\}$ with error
$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i].$$
- Choose $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$.
- Update:
$$\begin{aligned} D_{t+1}(i) &= \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases} \\ &= \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \end{aligned}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t h_t(x) \right).$$

Figure 21: Pseudocode of AdaBoost

The algorithm takes as input a training set $(X_1, y_1), (X_2, y_2) \dots (X_m, y_m)$ where each X_i belongs to some instance space X and each label y_i is in some label set Y . AdaBoost calls a given weak or base learning algorithm repeatedly in a series of rounds $t = 1, \dots, T$. One of the main ideas of the algorithm is to maintain a distribution or set of weights over the training set. The weight of this distribution on training example i on round t is denoted $D_t(i)$. Initially, all weights are set equally, but on each round, the weights of incorrectly classified examples are increased so that the weak learner is forced to focus on the hard examples in the training set. The weak learner's job is to find a weak hypothesis $h_t : X \rightarrow \{-1, +1\}$ appropriate for the distribution D_t . The goodness of a weak hypothesis is measured by its error

$$\epsilon_t = \Pr_{i \sim D_t} [h_t(x_i) \neq y_i]$$

Notice that the error is measured with respect to the distribution D_t on which the weak learner was trained. In practice, the weak learner may be an algorithm that can use the weights D_t on the training examples. Alternatively, when this is not possible, a subset of the training examples can be sampled according to D_t , and these (unweighted) resampled examples can be used to train the weak learner.

Once the weak hypothesis h_t has been received, AdaBoost chooses a parameter α_t as shown in the pseudocode. Intuitively, α_t measures the importance that is assigned to h_t . Note that $\alpha_t \geq 0$, if $\epsilon_t \leq 0.5$ and that α_t gets larger as ϵ_t gets smaller. The distribution D_t is next updated using the rule shown in the pseudocode in Figure 21. The effect of this rule is to increase the weight of examples misclassified by h_t , and to decrease the weight of correctly

classified examples. Thus, the weight tends to concentrate on “hard” examples. The final hypothesis H is a weighted majority vote of the T weak hypotheses where α_t is the weight assigned to h_t .

It is well known for AdaBoost to select out the optimal weak classifier with the least sample-weighted error rate, which might be suboptimal for minimizing the naïve error rate. To overcome this drawback, reduce the variance in our estimation and increase the prediction accuracy we implemented a new ‘Two-stage AdaBoost’ (TSAdaBoost). The flowchart of our implementation is shown in Figure 22.

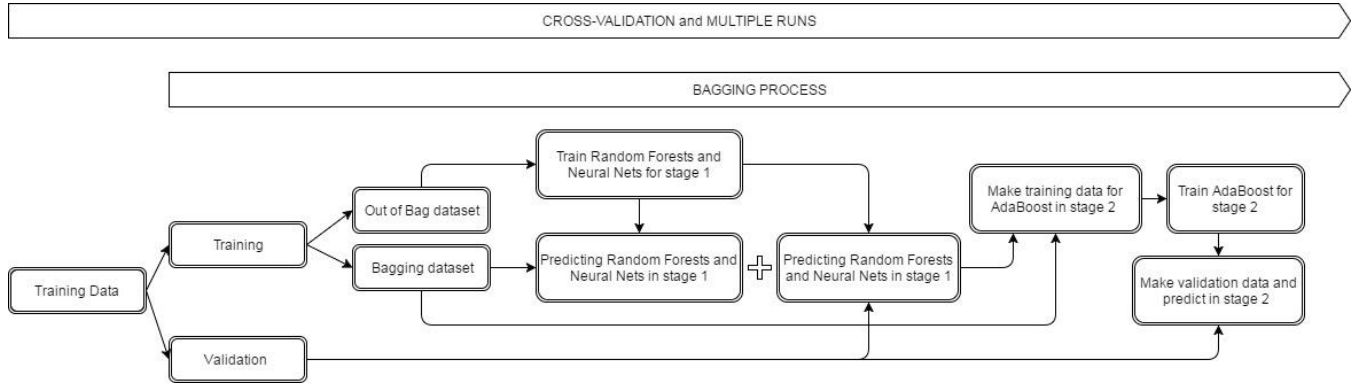


Figure 22: Flowchart of TSAdaBoost

As shown in the flowchart, in the first stage of the TSAdaBoost algorithm we implement two of our best base models i.e. Random Forests and Neural Nets and use the predictions from these to train the AdaBoost in the second stage. In the first stage, training data from our Kaggle competition is divided into training and validation data (to check the stability and accuracy of the predictions). We then make a 1:3 split on the training dataset into bagging and out-of bag dataset. Bagging helps decrease the misclassification rate of the classifier. It also decreases the variance of our prediction by generating additional data for training from your original dataset using combinations with repetitions to produce multisets of the same size as our original data.

We then fit Random Forest and Neural Nets on the out of bag dataset. These models are used to make predictions on the bagging dataset and 10-fold cross validation is performed on the initial validation data. However, only increasing the size of the training set does not improve the model prediction accuracy; it just decreases the variance, narrowly tuning the prediction to expected outcome. With this motivation to increase the prediction accuracy we use AdaBoost on the predictions obtained from the first stage. The prediction from the first stage of our model are included as features along in the training dataset for AdaBoost. This inclusion is made in such a way that similar predictions from both base models from the first stage are added as they are and predictions that contradict negate each other. This can be easily implemented if we add the features as

$$\text{new feature for stage 2} = \sqrt{\text{prediction from random forests} \times \text{prediction from neural nets}}$$

Thus, this will take care of negating contradictory predictions and adding similar predictions. The new training dataset is thus formed and we use it to train AdaBoost in second stage of TSAdaBoost. 10-fold cross validation is again performed to check variance of our estimation and stability of the algorithm.

Two-Stage AdaBoost Results

The algorithm took approximately 7 hours to run for 100 runs that were used to assess the stability. Figure 23 shows the mean accuracy and standard deviation obtained for the multiple runs of the TSAdaBoost algorithm.

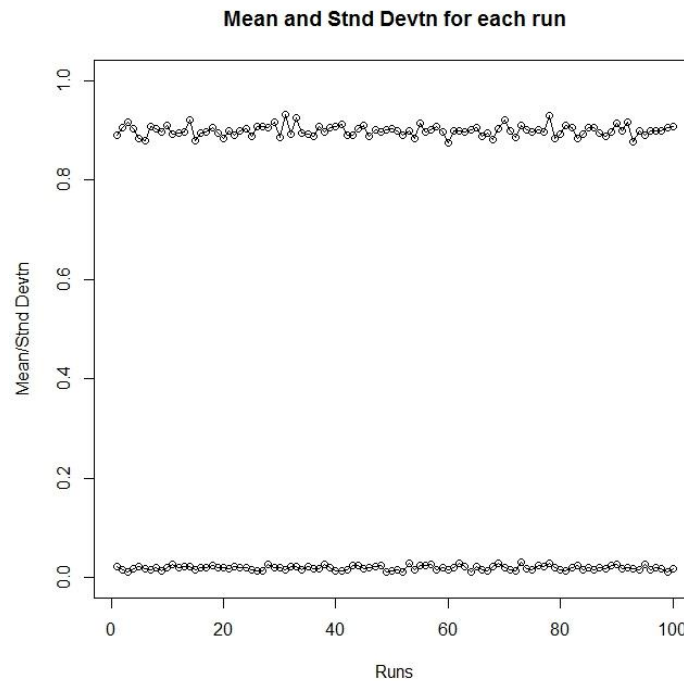


Figure 23: Plot for stability analysis of our new TSAdaBoost algorithm

The results show that the mean AUC is stable and the standard deviation does not fluctuate significantly. The average AUC on the train dataset was 0.9088. Final predictions were made on the test dataset and we obtained a submission AUC of 0.91 that placed us in top 20 of the private leaderboard for this competition.

Conclusions and Future Scope

With our Two-Stage AdaBoost model, we could achieve a very high classification accuracy and AUC score that led us to conclude that the model's ability to distinguish between a human bidder and a bot bidder, using the characteristics given, is satisfactory.

Further exploration to understand the trend of bot bids can be done by attempting to model the data using time series methods. In this case, we can study the pattern of bids placed by each bidder to identify bots who, generally, bid faster than humans, as we have seen from the results of our study.

Another point to be addressed is the fact that an unknown algorithm was responsible for labeling bids as placed by humans or placed by bots in the raw data. There is distinct indication in the data set that the algorithm classified bidders who placed a single bet as robots, while this is not necessarily the case. These 'outliers' need to be managed before our classification method can be used in general applications. Furthermore, this model can be generalized to accept more features and then applied to all online auctions. We believe that this could mitigate fraudulent activities in online auctions, hence, improving customer satisfaction.

Workload Allocation

Brainstorming process in feature extraction and report writing was done as a team. Codes for feature engineering using SQL and SQLite and handling class imbalance was done by Himadeep Reddy. Himadeep also came up with the Two-Stage Adaptive Boosting algorithm and coded the algorithm in R. Alekhya and Kirthana have worked on building the base models and implementing ensemble approach for stacking.

References

[1] <http://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/>

[2] Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16, 321-357.

[3] Ha, T. M., & Bunke, H. (1997). Off-line, Handwritten Numeral Recognition by Perturbation Method. *Pattern Analysis and Machine Intelligence*, 19/5, 535–539.

[4] Hastie, T., Tibshirani, R., Friedman, J. (2001). *The Elements of Statistical Learning*. New York, NY, USA: Springer New York Inc.

[5] Wen, J. B., Xiong, Y. S., & Wang, S. L. (2013). A novel two-stage weak classifier selection approach for adaptive boosting for cascade face detector. *Neurocomputing*, 116, 122-135.

[6] Freund, Y., Schapire, R., & Abe, N. (1999). A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780), 1612.