Machine Learning Engineer Nanodegree
Capstone Project Report

Georgios Bouzias

March 16th, 2018

I. Definition

Project Overview

In the broader sense, as also described in the proposal, the project falls into the domain of bot detection by their activities/behaviour in a given platform. Historically, bots have been viewed as a problem for many reasons but mainly DDoSing or spreading malicious software. Another reason that might render bots unwanted, is the fact that they spoil the online experience of a user in for example a game or somewhere like a website where people are supposed to interact with each other.

The first part (the one of DDoSing etc) has seen considerable work through the years like in [1, 2] and many more cases where the aim is to deal with this problem by tracking down bandwidth, packet timing and taking into account traffic behaviour in general. The other problem that refers to the decay of user experience of an online product has to do with bots using superhuman abilities in for example a game or a bidding site which is the proposed case. They can achieve that in many ways, for example by having access to an API, the screen pixels, or they refresh in an optimal manner or something similar.

Research in that area leverages the users historical data, or other behavioral aspects like for example in cases [3] and [4]. In [3] a single behavioral sequence (as feature) is extracted by social media histories to detect spam bots, and in [4] a CAPTCHA test is proposed in an online game context. In the area there are also companies trying to deal with these kinds of issues in the spectrum they are discussed here [5].

The case this project is dealing with, is bot detection in an online auctions platform. The problem is viewed as a supervised learning problem and we leverage a labeled dataset from a kaggle competition [6]. The issuing authority of the competition was facebook, in 2015, and its purpose was for recruiting. The activity of each user is captured by a very large (around 900MB) bids dataset where every bid (more than 7.6m bids) is listed, along with the person who made it and additional information such as the ip it came from, the device it was made with, the bid's timestamp etc. Also, in addition to that, there is a labeled dataset of 2013 bidders and whether they are robots or not (1 and 0 respectively).

That accounts for two datasets, one with bidders and one with bids. The general idea is to populate the bidders dataset with features by the bids dataset and then use supervised learners to predict the label of each bidder. For every bidder we can have a "bidder table" with the bidder's bids, from the larger bids table, and then we can create meaningful features by that for every bidder separately. This report describes this process, along with the classification in detail. The key software that has been used to produce our solution is python v3.6.4 with the tools: numpy, pandas, matplotlib, sklearn, xgboost. The solution is implemented in two ipython notebooks, '*Feature Creation - Capstone Project.ipynb*' and '*Classifier - Capstone Project.ipynb*' that deal with feature construction and classification respectively.

On a final note, it has to be mentioned that despite the fact that we refer to two datasets here, that was not the case in the proposal and here lies a minor inconsistency between the report and the proposal. We had suggested that there are two bidder datasets, one for training and one for testing. But because the datasets were issued by kaggle, the test dataset was for kaggle's public leaderboard and it therefore wasn't labeled. It only had the bidder ids the predictions should refer to. The request for the participants was to file a csv with probability rates for each id and from the kaggle side, they would compute the AUC score for the given rates. To work around that in this project, instead of evaluating our approach with the bidders in the test set we got a 10-k cross validated score by the train set.

Problem Statement

The problem can be stated shortly and concisely as *detecting the bots among a user base of an online auction platform by their bidding histories.* As has already been said there is a labeled (robot/human) dataset of bidders, and a dataset with bids for all those users. The proposed solution is a two step solution, the first step being to use each bidder's history in order to create relevant features, and the second to apply supervised learners to the labeled dataset of bidders.

Regarding the first step, the way to find the features heavily relies on the bids dataset. While we discuss that more thoroughly later on, at a higher level in this point we can say that given the history of someone's bids in chronological order, we can have multiple distributions regarding the features of the bids someone is making. As far as these features go, there is the device listed for each bid, the url used, the ip, country, auction, merchandise and a timestamp - all being categorical except timestamp.

For each one of these columns we can have several stats and counts that describe it, such as the number of devices, urls etc. that someone is using, or the mean number of bids by one - let's say - device, or the lowest and most bids per device and other things. Similarly we can get features by the distribution of counts for all the features in the bids dataset - with the exception of timestamp where it makes little sense to count since every timestamp shouldn't happen more than once. In addition to that there might be a lot of information in the timeseries and the temporal patterns of someone's bidding activity in general, or by auction/country etc that we will discuss in the methodology section.

As far as the second step, which is the application of classification algorithms, that is also very straight forward as well. Given that we have a labeled dataset with many features that we created in the first step, we will approach the problem as a supervised classification one. We will be using two algorithms - more specifically a Random Forest and a Gradient Boosting one - and ultimately, given a bidder with a set of features they should be able to give a probabilistic prediction for whether he/she is a bot or a human. For the algorithms we will first tune their parameters and then evaluate them by the Area Under the ROC curve metric. The evaluation as we said earlier (absent the test set), will be on the train set, with a 10 fold cross-validation method.

Metrics

In this project the metric that will be using to measure the performance of the classifier will be the Area Under the ROC curve (AUROC or AUC for short). This is a metric that can be used for the non-deterministic classifiers, meaning the ones that can give a probabilistic output. Given the true labels of a set of instances (the test set), and a computed likelihood of belongingness to a class by the classifier, a ROC curve can be computed [7] based on True Positive Rate (TPR) and False Positive Rate (FPR).

$$True\ Positive\ Rate = \frac{Positives\ correctly\ classified}{All\ positives}$$

$$False\ Positive\ Rate = \frac{Negatives\ incorrectly\ classified}{All\ negatives}$$

The resulting ROC curve mainly captures the success of a model in different thresholds. Afterwards the area under this curve can be computed, which reflects the overall performance of the model across those thresholds. In addition to that, another benefit is that because this is an area of the unit square, it will always be within a range of $[0, 1]$ and that makes interpretation and comparison more intuitive. Except of the overall performance, given the ROC curve one can also understand the potential trade-off between True Positives and False Negatives, and the sacrifice that has to be made in one end to get a better score on the other by manipulating the threshold.

As far as using AUC instead of some other metric goes, it can be said that AUC is very robust against very skewed distributions which is also the case here. That is because it is comprised by a ratio of positive (TPR) and ratios of negative data (FPR), and not some ratio that includes count measurements of both the positive and negative distributions [7]. However, that is not the case with other metrics like accuracy, which might be very misleading since we could make a very naive classifier predicting only zeros and its accuracy would be at around 90%, or precision/recall metrics that heavily fluctuate. A reason of the fluctuation according to the distribution skewness is because as we can see at the equation below, precision is comprised by points of both classes.

$$Precision = \frac{Positives\ correctly\ classified}{Positives\ correctly\ classified + Positives\ incorrectly\ classified}$$

In addition to that, it is fair to say that Equal Error Rate (comprised by False Rejection Rate and False Acceptance Rate), could also probably be used here. However it might be that AUC is a little bit more nuanced since there is also the ROC curve which makes a relevant trade-off for this problem visible. Finally, we can also say that an additional incentive for AUC is that this is the metric that was used for the benchmark models - the issuing authority from kaggle ranked the solutions according to that - so with it we can more easily compare our model.

II. Analysis

Data Exploration

The first part of the project was to understand the data and make relevant features. All this process is documented in '*Feature Creation - Capstone Project.ipynb*', and it starts with exploring the data. The two datasets that were used for this projects as has already been said are the bidders one with the labels and the bids one which holds the information for all bids that the bidders in the first dataset have made. The first dataset, which holds the bidder ids and the labels, also has the payment account of the user and the address. Here is a view of the first three rows of the dataset:

```
                                 bidder_id  \
0  91a3c57b13234af24875c56fb7e2b2f4rb56a
1  624f258b49e77713fc34034560f93fb3hu3jo
2  1c5f4fc669099bfbfac515cd26997bd12ruaj

                           payment_account  \
0  a3d2de7675556553a5f08e4c88d2c228754av
1  a3d2de7675556553a5f08e4c88d2c228v1sga
2  a3d2de7675556553a5f08e4c88d2c2280cybl
```

```
                                 address  outcome
0  a3d2de7675556553a5f08e4c88d2c228vt0u4      0.0
1  ae87054e5a97a8f840a3991d12611fdcrfbq3      0.0
2  92520288b50f03907041887884ba49c0cl0pd      0.0
```

Of course all sensitive information is obfuscated for the sake of the project. Out of some basic statistics that we run it seems that:

```
The bidders dataset has 2013 rows.
The bidders dataset has 2013 unique bidders.
The bidders dataset has 2013 unique payment accounts.
The bidders dataset has 2013 unique address.

The dataset consists of 1910 humans and 103 robots, which accounts for 94.88%
and 5.12% respectively.
```

We can see from the get go how heavily skewed the dataset is towards humans. Also we can see that the payment accounts and the addresses are not really needed for this project as they are matched 1-to-1 with the ids. For that reason we drop those columns moving forward. That leaves the bidders data set pretty naked, and it is the one we will look to enhance with ids.

In addition to the bidders we shall do a similar inspection to the bids dataset. The kind of information it holds for every bid is its id, its bidder's id, the url auction it happened for, the merchandise the user was looking for when it came to the site (not the merchandise the auction was for, but more like the ad or the search term that led the user to the site), the device a user made the bid by, the timestamp it happened, the country and ip it came from, and the url that was used for it. A closer look at the first three rows seems like this:

```
   bid_id                            bidder_id auction     merchandise  \
0       0  8dac2b259fd1c6d1120e519fb1ac14fbqvax8   ewmzr         jewelry
1       1  668d393e858e8126275433046bbd35c6tywop   aeqok       furniture
2       2  aa5f360084278b35d746fa6af3a7a1a5ra3xe   wa00e      home goods

   device              time country              ip              url
0  phone0  9759243157894736      us    69.166.231.58  vasstdc27m7nks3
1  phone1  9759243157894736      in    50.201.125.84  jmqlhflrzwuay9c
2  phone2  9759243157894736      py   112.54.208.157  vasstdc27m7nks3
```

By the issuing authority, it is stated that the ips and urls are also obfuscated to protect privacy and in addition to that the timestamps have been transformed. Also, regarding the timestamps, they look like they are in Unix time format, and it looks like they are the only numerical value with all the others being categorical. Regarding some statistics about the categorical values it seems that:

```
The bids dataset has 7656334 rows.
The bids dataset has 7656334 unique bids.
The bids dataset has 6614 unique bidders.
The bids dataset has 15051 unique auctions.
The bids dataset has 10 unique merchandise categories.
The bids dataset has 7351 unique devices.
The bids dataset has 200 unique countries.
The bids dataset has 2303991 unique ips.
The bids dataset has 1786351 unique urls.
```

However there is a small problem with this, because the bids dataset includes bids for bidders that are not in our dataset, but only in the dataset kaggle used to do evaluations for its public leaderboard which as said earlier will not be used here. For that reason we removed those bids. Also, we go rid of the bidders for which there are very few bids, because for them there is no pattern to learn. From a check that we run, we found that the percentage of humans in the bidders (1034, which is more than half of the bidders) that did below 20 bids was 99.33%. By a close inspection of the bids of the small-ball bidders, we decided to drop the bidders with 2 bids or less as there is hardly any discernible behavioural pattern to train a classifier with.

In addition to that, after the removal, in the remaining rows that would be the ones that we would work with, we run a check for NaN values. It seemed that the only NaNs we had were 2700 values all in the countries column. What we did was that we replaced them with an empty string (") and we were cautious later on when it came to making features for countries when it was needed. In that sense after dropping those rows and taking care of the NaNs, the above statistics look like:

```
The bids dataset has 3070630 rows.
The bids dataset has 1536 unique bidders.
The bids dataset has 12740 unique auctions.
The bids dataset has 10 unique merchandise categories.
The bids dataset has 5728 unique devices.
The bids dataset has 199 unique countries (one is '' that replaced nan).
The bids dataset has 1030732 unique ips.
The bids dataset has 663561 unique urls.
```

Afterwards, to complete an initial pre-processing/exploring stage, we checked some statistics of the distributions of the unique counts for each bidder and finally we explored and transformed the timestamp (which is probably the most important feature). Regarding the distribution of counts, by grouping the bids dataset by the bidder id column we got the mean, the median and the standard deviation of the distribution of counts for each category:

```
The average auctions per bidder is 80.47786458333333.
The average merchandise per bidder is 1.0006510416666667.
The average devices per bidder is 101.17057291666667.
The average countries per bidder is 16.856119791666668.
The average ips per bidder is 871.560546875.
The average urls per bidder is 446.6451822916667.
------------------------------------
The median auctions per bidder is 19.0.
The median merchandise per bidder is 1.0.
The median devices per bidder is 20.0.
The median countries per bidder is 5.0.
The median ips per bidder is 31.0.
The median urls per bidder is 10.0.
------------------------------------
The standard dev. of auctions per bidder is 163.08370575548003.
The standard dev. of merchandise per bidder is 0.025507210968957799.
The standard dev. of devices per bidder is 207.95095530227817.
The standard dev. of countries per bidder is 25.556962553167057.
The standard dev. of ips per bidder is 5427.287629974064.
The standard dev. of urls per bidder is 3034.296323390458.
```

From a first glance we can see that there are very skewed distributions only by watching how far the mean is in some cases from the median, like the urls and ips. Also we can see something odd, which is that the mean merchandise is very close to 1, which means that for every bidder there is almost an 1-to-1 mapping with its merchandise. After checking it further, we found out that there is in fact only one bidder that has two merchandise categories. From that we can see that except the fact that it is very strange, and looks like an error might have happened here while collecting the data, there is not really patterns to be found here if that is the case.

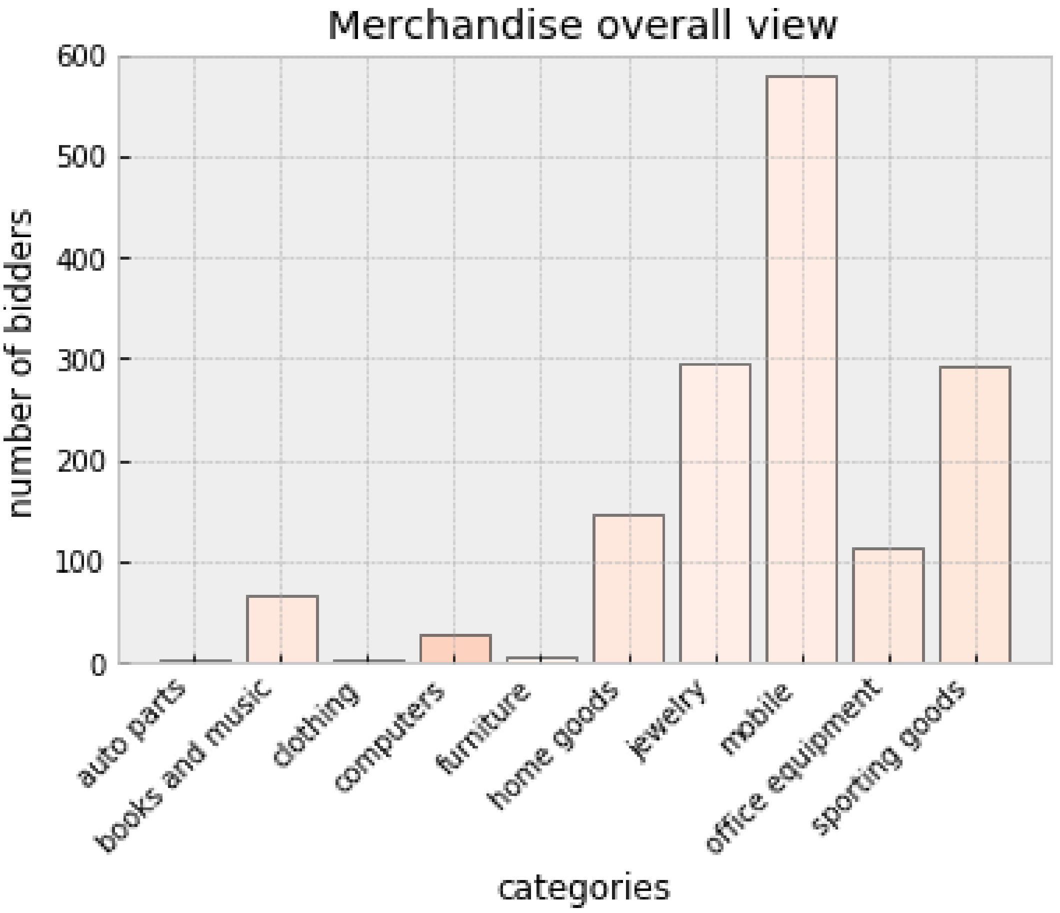


Figure 1: The counts for each merchandise category. We can see that some like 'auto parts' or 'clothing' have very few bidders. Also the intensity of the red color indicates how *contaminated* a category is with bots.

For that reason, we removed this column going further. Before removing though, we also looked at the percentages of bots in each of the merchandise categories, just in case we could use merchandise as a simple single categorical value but it didn't seem to matter as well (meaning that there was no concentration of bots in one merchandise category). In Figure 1, the counts for the bidders in every merchandise category can be seen, as well as how much "contaminated" a category is with bots, which is captured by the intensity of the red color. Only in the computers category there seemed a little more concentration but that was negligible.

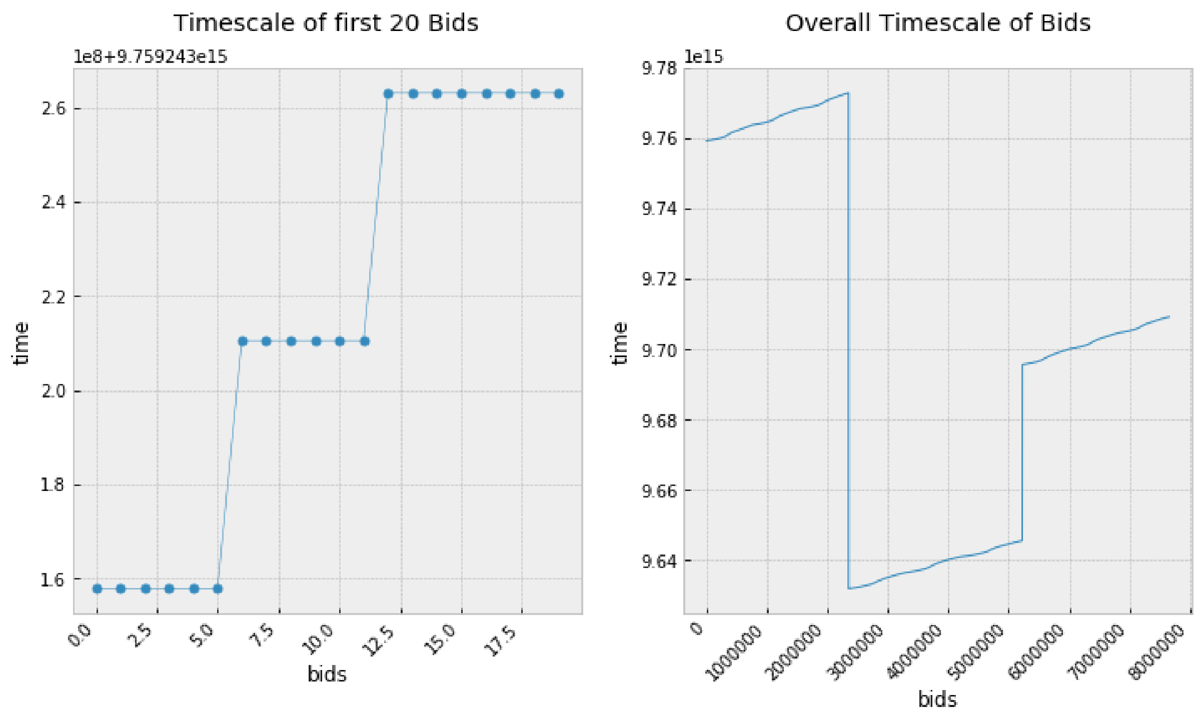


Figure 2: The timeseries of bids before any kind of processing, as sorted in the dataset. In the left we can see the first 20 bids, and in the right all 7.6m (in approximation).

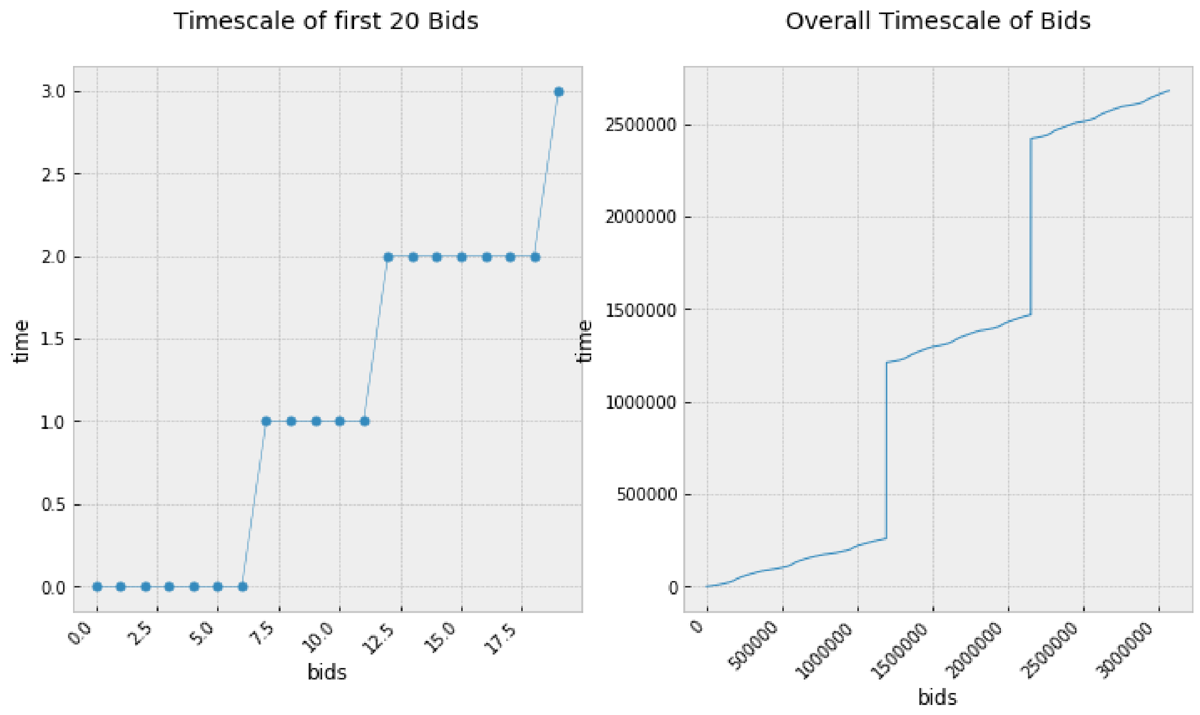


Figure 3: The timeseries of bids after all 4 processing steps. In the left we can see the first 20 bids, and in the right all 3m (in approximation).

Finally, when looking at the timestamp that we have so far only slightly discussed, we can see by a raw data quickplot in Figure 2 several things. First of all, it seems that the values are extremely high integers, that there are three main periods, and that the bids dataset is not really sorted, as the first period, is the latest and should be last. Also, by the left plot which only shows the first 20 values (from the 7m bids as this is the initial set), we can see that there are several bids made in the same timestamp.

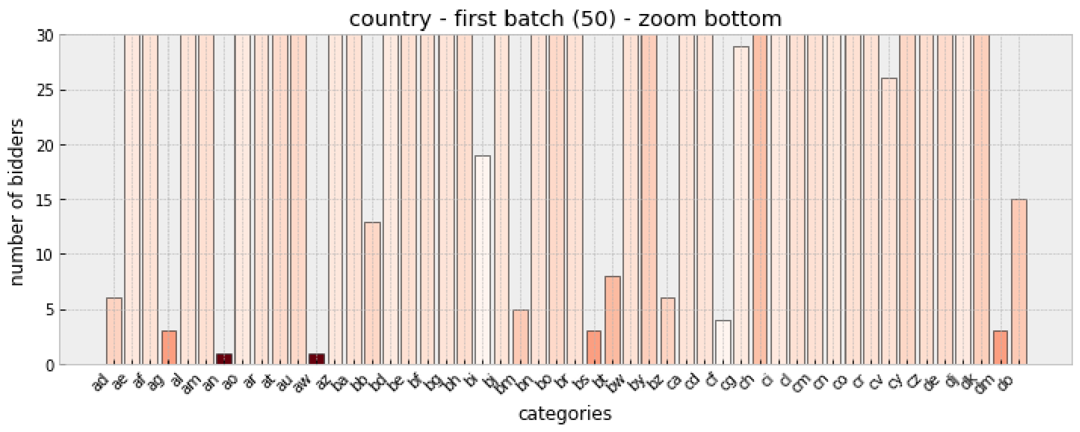
Also, by further exploring the intervals it seems that there is a somewhat stable sampling rate throughout the dataset. If we first sort the dataset according to the timestamp and then get the counts for all the appearing intervals, it seems that there is only a handful of values (only 16 different values in 7m bids), and one of the values - which appears two times - is extremely large. It obviously accounts for the two big gaps, that if we sort the dataset are exactly equal. That means that there are probably two big breaks between bidding activity. Also, since the smallest jump happens almost 90% of the time and that all the other jumps are almost exact multiples of that, we can safely assume that this is a kind of sampling rate that we can consider a unit of time.

Given all that, we decided to do a four-step processing according to the timeseries. Each step either refers to the entire dataset (or better the pandas dataframe in which it was held), or only the timeseries column, and the result can also be seen in Figure 3. The pre-processing steps that we took regarding the timeseries were:

1. Sorted the entire bids dataset according to the timestamp column
2. Normalized the timestamp column so that it starts from 0
3. Re-adjusted the index of the entire dataset so that it is a continuous sequence (referring to the pandas dataframe structure that it was being held)
4. Normalized according to the unit of time, so that the values are more comprehensible

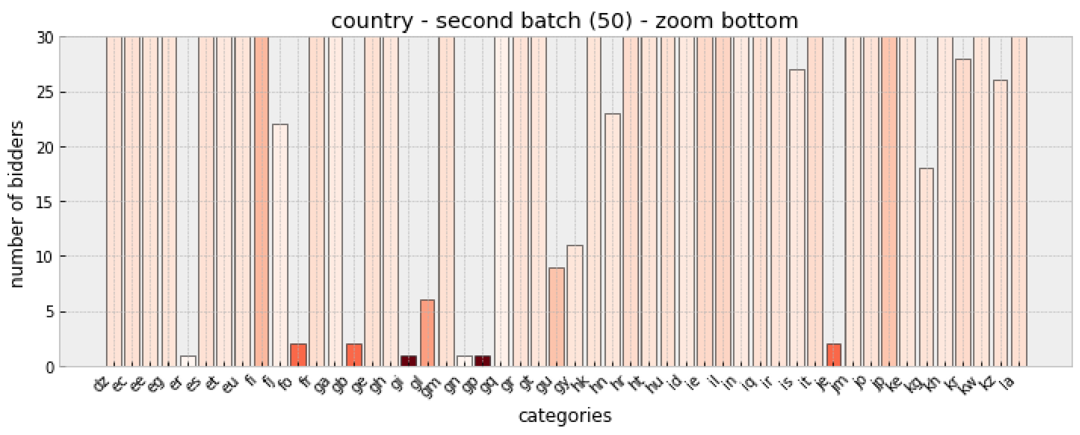
Exploratory Visualization

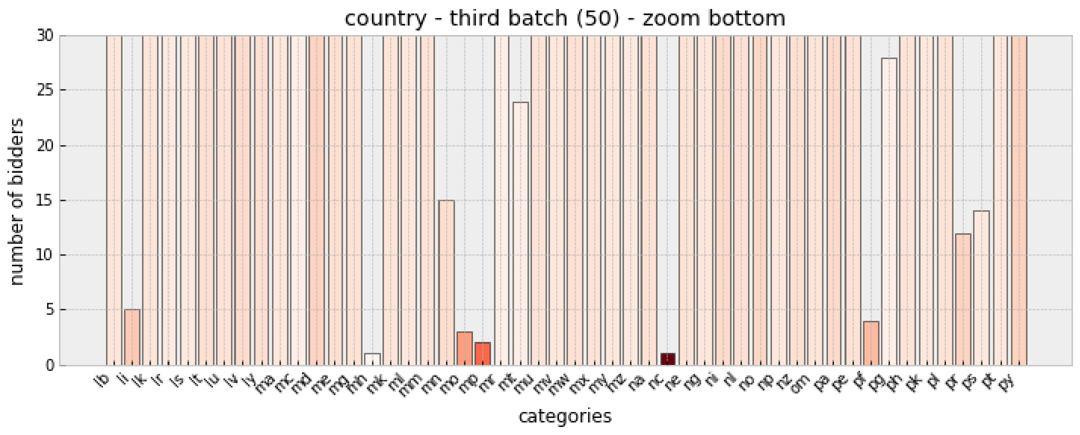
The remaining columns of the bids dataset from which we will extract features are the auction, device, country, timestamp, ip and url. As we saw earlier though, except the timestamp all these columns are categorical and have too many categories. That makes doing a bar plot like the one we did for merchandise a bit hard. However, as far as the country column goes, this is possible because there only are around 200 countries.



(a) First batch of 50 countries, their counts of bids (until 30), and ”contamination” (red color intensity).

In the plots of Figure 4 there is a ”zoom” in the lower 30 bidders, because due to the very big height for some countries, the scale grows very much and the small values which are the most interesting ones in the sense that they have the highest bot percentages are lost. The full


(b) First batch of 50 countries, their counts of bids (until 30), and "contamination" (red color intensity).


(c) Third batch of 50 countries, their counts of bids (until 30), and "contamination" (red color intensity).

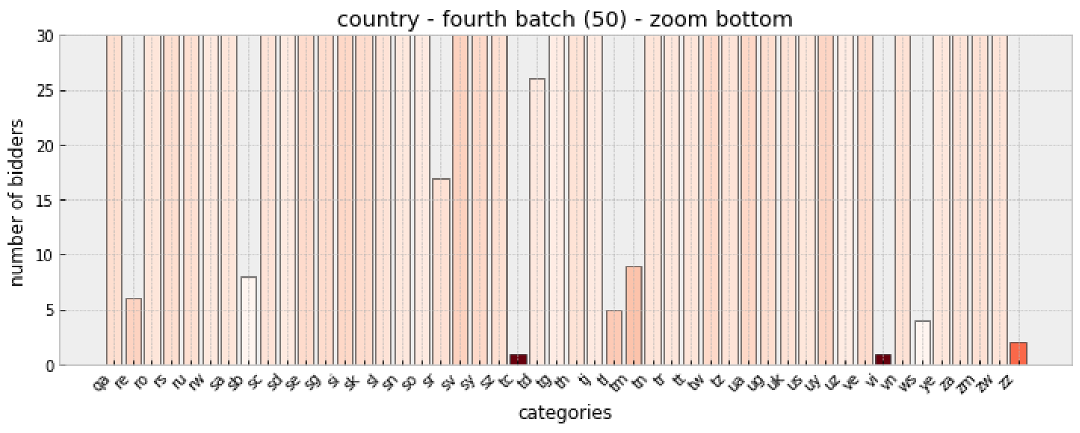
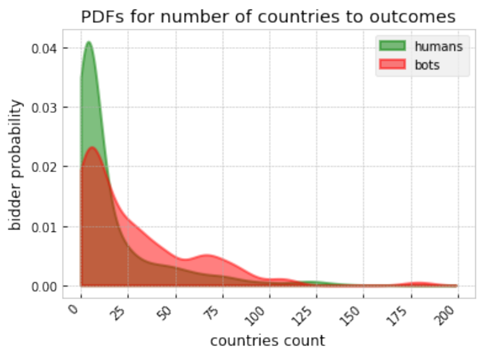
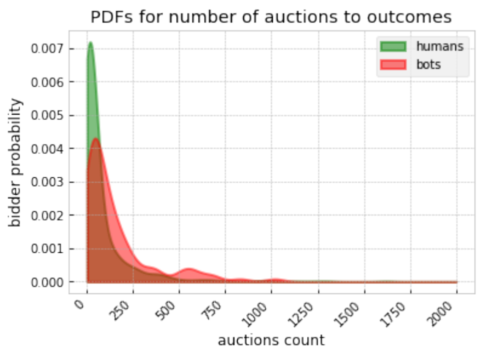

(d) Fourth batch of 48 (49 with NaNs) countries, their counts of bids (until 30), and "contamination" (red color intensity).

Figure 4: In this figure, a bar chart for every country of the 199 (one for NaN countries) in the dataset is shown split in four groups of 50 or 49. The length of the bar shows the number of bidders and its red color intensity shows the *contamination* of this country with bots. In order to highlight the small countries where the most interesting results lie, this plot zooms in the bottom of the chart, showing only heights up to the 30 first bidders.

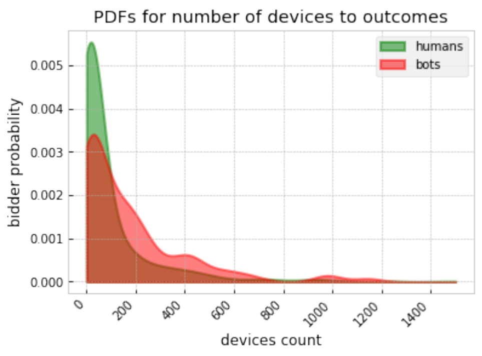
length plots that show better the comparison between the countries in the sense of their number of bidders can be found at Figure 8 at the appendix.



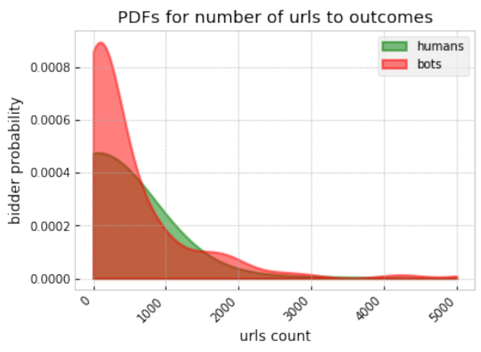
(a) Bot/Human PDFs for number of countries



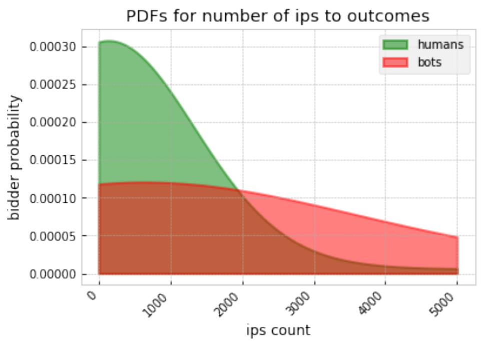
(b) Bot/Human PDFs for number of auctions



(c) Bot/Human PDFs for number of devices



(d) Bot/Human PDFs for number of urls



(e) Bot/Human PDFs for number of ips

Figure 5: In this figure, five Probability Density Functions as shown for the categories of countries, auctions, devices, urls and ips, and their likelihood to have a human or robot according to their counts.

As far as the remaining features go, and given that we were considering to make features for the distribution of counts as said earlier (e.g. the number of auctions or countries someone bids for or from) we tried to visualize the probability of a bidder to be a bot or human according to the absolute count of auctions someone is bidding for or the countries, urls, devices, ips he/she is bidding from. That can be seen in the following plots for the remaining categories. In addition to that, we tried to make a bar chart that shows the counts of any of those categories as a function of the number of bidders or the number of bids, but it is a bit confusing it it only shows that there are a few instances (a few auctions or a few countries etc) that gather most of the attention for every category, which of course makes sense (the plot can be seen in Figure 9 at the appendix). It was usually the case that the biggest count was for a few users and the revers (e.g. that a few users bid in very many auctions, and many users bid in few).

From the plots, which can be seen in Figure 5, we can see that there is not much difference across the first three plots; it seems that the bot bidders tend to have a little bid more devices, to bet in more auctions and from more countries and ips. However, they seem to use fewer urls thyan the human users. Another thing about these plots, is that in the x axis, the maximum number of possible counts would be all the counts of the dataset. For example, in the case of countries, the max number of countries one can have is 199, and therefore the plot's x axis was until that. However that is not the case for urls and ips, for them we selected an arbitrarily large enough stopping point to show the first counts because the number of unique elements was extremely big, and the plot would not make sense.

Algorithms and Techniques

Regarding the algorithms and techniques that have been used for this project, it is clear from what has been said so far that they need to be discussed in two different groups. First there are the techniques and algorithms that have been used for Feature Construction and then the ones that have been used for Classification.

Feature Construction

The Feature construction can also be grouped even further into the features for categorical values and the temporal features. First regarding the categorical ones, for every group of features we computed the counts for each one of its values, for example if device1 appears 3 times and device2, appears 10 times, and then we found the aggregate number of all counts, the max count, the min count, the mean of counts, as well as the median and the standard deviation of them. For the case of countries we also found out the few countries that have few bidders in general (we saw that most of them were very reddish in the visualization), and then we made a feature out of the percentage of "ghost" countries a bidder bids from. All and all we made 6 features from the counts method for 4 categories, and 7 for one category, which accounts for 31 features.

Now except the categorical features, there are the temporal ones. For them we can do even further grouping. There are the ones that refer to the timestamps themselves, and the ones that refer to the intervals. As far as the timestamps go, we took the ratio of simultaneous bids to all the users bids (the bids that happen in the same time stamp from the same user), the smallest and biggest timestamp value, and the median, mean timestamp along with the std of the timestamps, which is very similar to what we did for the counts.

In addition to timestamps we also looked at the intervals. For that we extracted the number of periods someone participated in (meaning in how many of the three large periods we identified earlier someone made bets), and in similar fashion as before we found the smallest interval, the biggest interval, the mean, median interval and the std of intervals. In addition to those here we extracted the number of bids someone made. All that accounts for 13 temporal features.

However that is not the entire story regarding the timeseries data. Given that we can group even further this sequence of bids for every user, we can get several timeseries across another group, like auctions, devices or countries. For urls and ips it is not really feasible because in many cases there is a lot of them which means that there would be a timeseries of 1 element in a great deal of cases.

Then given that we have the timeseries for that group we can have several distributions, for example, for a specific bidder we can get several bidding sequences for the different auctions he or she participated in. Then we can run the timeseries and intervals statistics on each sequence separately and then we have a distribution of means, medians, mins, maxes, stds, for both timeseries and intervals, as well as a distribution of num of periods, number of unique bids and ratio of simultaneous bids. That accounts for $13 \times 5 = 65$ features, if we find the min, max, mean, median and std of those distributions.

From the three categories that we could do the further grouping, in this project we used auctions. One could say that the results could be improved if we were to also use the features from grouping by devices and countries, but the feature space would increase dramatically by 130 features. Due to time concerns and computational power we decided to only do the grouping by auctions and if the results would be bad then we would consider making changes.

Finally regarding the feature creation, these are all the features that we computed along with the number of bids for each participant. All these account for $31 + 13 + 65 = 109$ features for each bidder. We produced a new dataset for this that essentially was an augmented bidders dataset, since it contained the bidder id, the label, and 110 extra columns and we used it to do the classification.

Classification

As far as classification goes the main idea was that we tried two supervised learners that are robust to large feature spaces, a Random Forest and an implementation of a Gradient Boosting algorithm called XGBooster (Extreme Gradient Boosting). We first optimized them with Random Search of the parameter space (that used AUROC score and 5-fold cross validation to evaluate each set of parameters), and then we computed the AUROC score, with the best set of parameters and a 10-fold cross validation for the final result and visualization. The data that we used on the classifiers was the 'bidders_full_features.csv' dataset that we produced from the previous phase.

The two basic techniques that we used for classification was cross validation and random search of the parameter space. For cross validation we used a stratified split because the sample was heavily skewed towards humans and a random split could very probably leave the robots class very underrepresented. Also, the cross-validation was used during both the optimization (5-fold) and the AUC final result. In addition to that, regarding the random search, it didn't happen the same way for both classifiers. In the random forest we searched the space in one go, but in the XGB we searched the space in batches, first for the tree parameters, then for the regularization and finally for the alpha. More details on how we implemented the techniques we will discuss in the following implementation section.

Benchmark

For this project the benchmark is quite clearly defined. Because as has been said earlier this is a project of a previous kaggle competition that has been finished a few years ago, the results along with discussions and sometimes code are available. However, they are not directly comparable with the results we will produce, because as has already been said these results are evaluated in a test set for which we do not have the labels. kaggle had the labels and produced the result after the users presented a csv with bidder id's and likelihoods.

As far as the results on the final, private leaderboard go, it seems that around 45% of the participants had an AUC score of 0.9 or better - the best was 0.94254. This project is challenging for us in many aspects like making plots, making efficient code (because the data is too large and if some aspects of pandas or numpy are not used well, some cells would run for ages), making good features etc. However, given the access at resources that we have, namely the discussions, the results and some times code and blog posts of solutions [8, 9, 10], we would expect this solution to be at around the 20-30% best. Also, even though the score is not directly comparable with the kaggle leaderboard ones since we are training, optimizing and predicting on the same data set we think that a cross-validated result is not far from the true description of the models generalization power.

III. Methodology

Data Preprocessing

Probably the most demanding part of this project was Feature Construction, which means that we created the features of the train dataset entirely. For that reason there was no preprocessing step with the conventional way of saying it. Instead, we were looking for NaNs, or mistakes, or outliers when producing the features, mostly to make sure that they were correctly produced. However, there was a point just before producing the final csv in the Feature Construction notebook and after making some last checks for mistakes or unwanted values, where we did a normalization procedure with the sklearn MinMax scalar object to bring all columns to a [0, 1] range, and we also filled for 17 bidders their 5 NaN values with 0. We expected that since we were making the features because in some cases there could be only one value in a timeseries and therefore making intervals would not be possible. These two things synopsize our pre-processing efforts for this project.

Implementation

In regards to the implementation of the learners, as said earlier we used two classifiers. These were the sklearn implementation of a Random Forest [11], and the XGBoost implementation of a Gradient Boosting algorithm [12]. The purpose of both classifiers is to output a likelihood (probability) score for a given bidder that he is a robot according to the bidder's features. To do that, we first load the data that we produced in the feature construction step, we give them a shuffling just in case they are put with some sort of hierarchy and then we are optimizing and evaluating the two learners separately. The implementation for all the classification process can be found in '*Classifier - Capstone Project.ipynb*'.

The two basic tools that we used for the optimization and evaluation of both the classifiers were Cross-Validation and Randomized Search for parameters. As far as Cross-Validation goes, we have always used it as Stratified, because the distribution of labels was heavily skewed and a Stratified split ensures that the ratio of both labels is maintained. In addition to the Stratified mode, we also used it unshuffled to make sure that all data points are shown during training. Random Search, which was the method for tuning the parameters by randomly sampling a set of parameters by a designated parameter space, also uses CV in that fashion. They way it works, is that for a specified number of iterations, it picks a set of parameters at random and outputs a score. Then the best scoring parameters are kept.

In terms of applying the algorithms, as far as the RF goes, the process was very straightforward, by simply using the sklearn API. We did always run the RF with 2000 estimators, both in tunning and predicting. Also for Random Search we used the RandomizedSearchCV [13] object. For that process we used the following three steps:

1. Define an instance of the classifier, a parameter space, and a RandomizedSearchCV instance from sklearn.model_selection, for 30 iterations, that uses the AUC metric and 3-fold CV

2. Fit the RandomizedSearchCV object, to find the best parameters
3. Re-initialize and fit the RF classifier with 10-fold CV to get a better approximation of the AUC and visualize it, along with the ROC curve [14]

Subsequently, as far as the XGB model goes, its case is a bit more complicated, mainly because of the fact that we wanted to use the early stopping parameter with the cross validation, which we could combine only with the native API [15, 16], and not by the RandomizedSearchCV object. Early stopping allows XGB to stop making extra trees if the performance stops improving for a number of consecutive iterations. We did always run XGB with 300 max estimators, and early stopping at 30 runs - by far most attempts stopped improving somewhere before 100 trees. Also, because it has different classes of parameters (not just tree parameters, but regularization ones, and an alpha), it seems that it would be best to be optimized in batches [17]. In that regard, following the instructions in [16] we followed an six step process to optimize XGB:

1. Change the input data into a DMatrix format which is the core XGB data structure
2. Define a parameter space and some initial values
3. Find the best **tree-related** sets of parameters, by getting a cross-validated AUC score for each set, with the Random Search method by the
4. Find the best **regularization-related** sets of parameters, by getting a cross-validated AUC score for each set
5. Find the best **learning-rate**, by getting a cross-validated AUC score for each set
6. Use the final best set of parameters to produce a 10-fold cross validated AUC result to get a better approximation of the AUC and visualize it, along with the ROC curve [14]

A key difference with the XGB Cross-Validation and the RF one is that in the case of XGB we did a 5-fold split (it was faster so we could afford getting an improved AUC score). In addition to that, it should be mentioned that regarding the final mean ROC-AUC computation, we are using most of the code from the sklearn example in [14], by which we are finding the mean True Positive Rates across the False Positive Rates of all folds that lead to a ROC plot and a final AUC score. Finally, besides what has already been mentioned in the introduction about the non-available test set - instead of which we got a cross-validated result on the train data - we could say that everything else regarding the implementation went as planned.

Refinement

In terms of refinement, for both classifiers we held the number of estimators (which is the number of weak tree learners they would use) constant throughout tuning. That is because in general for these cases more weak learners means more accuracy - or no improvement. Then, by using Random Search and Cross-Validation the way we discussed earlier we got the best set of parameters.

In the case of Random Forest, the number of estimators was set to 2000. Also, 6 parameters were looked at for 30 iterations by a RandomizedSearchCV object. That object handled CV which was set to 3 folds. For every one of the iterations, the search object picked a set of parameters at random from the space we confined them and was running a CV with them; then the best set of parameters was available to access it on the classifier. The parameters and the corresponding space they were looked at were:

- "max_depth": [None, 2, 5, 8, 11, 14, 17, 20]*, were the allowed values*
- "max_features": [1-100]*, was the allowed integer range*
- "min_samples_split": [2-100]*, was the allowed integer range*
- "min_samples_leaf": [1-100]*, was the allowed integer range*
- "bootstrap": [True, False]*, were the two possibilities*

- "criterion": ["gini", "entropy"], *were the two possibilities*

The resulting best set of parameters for the Random Forest was: max_depth 8; max_features 76; min_samples_split 30; min_samples_leaf 56; bootstrap True; criterion "entropy".

In the case of XGBooster, as has also been mentioned earlier, the number of max_iterations was always held set in 300, and the early stopping at 30. In very few cases the iterations were more than 100. Also as has been said earlier we could not use the RandomizedSearchCV object in combination with early stopping, so we hardcoded the Random Search and in addition to that, we mentioned that we wanted to look at the parameters in batches. Because of the batches we also used a different number of iterations according to what we were optimizing: we did run 50 iterations for the two first batches and 99 for the last.

The way this worked was by initializing a default set of parameters, and we were gradually changing that set with the parameters we are optimizing over time. In order to do the Random Search in this format we created the random distributions in a list and sequentially we went over them and found their AUC score. Then we simply kept the set with the best score as the RandomizedSearchCV object does. We computed the parameters in three batches as shown below - of course the parameters we computed later were optimized on the refined parameters of the previous batch. Also in the final batch which was the learning rate, instead of searching randomly we defined a sequence with small intervals and looked into that.

1. Tree-related parameters
   - "max_depth": [3-40), *was the allowed integer range*
   - "min_child_weight": [0-40), *was the allowed integer range*
   - "gamma": [3-50), *was the allowed integer range*
   - "max_delta_step": [0-10), *was the allowed integer range*
2. Regularization-related parameters
   - "subsample": [0.1-1), *was the allowed floating point range*
   - "colsample_bytree": [0.1-1), *was the allowed floating point range*
   - "lambda": [1-5), *was the allowed integer range*
3. Learning rate
   - "eta": [0.005, 0.5), *, the values were floats between that range in 0.005 intervals - this list was searched exhaustively and not randomly since there was no other parameters*

The initial set of parameters for the XGB was: max_depth _; "min_child_weight" _; gamma _; max_delta_step _; subsample 0.8; colsample_bytree 0.8; lambda 1; eta 0.3. Some of the values do not have parameters (they have _) because the were the first to be tuned. **Finally, the resulting best set of parameters for the XGB was: max_depth 18; "min_child_weight" 2; gamma 6; max_delta_step 9; subsample 0.611591; colsample_bytree 0.723225; lambda 1; eta 0.3.**

IV. Results

Model Evaluation and Validation

Given the two tuned models, the Random Forest one and the XGB one, we can now compare them in order to see which one is the best and choose it as our final solution. Because the tuning process already gave us some results about the performance of each one (since the results were already cross-validated), we know that the XGB model did a bit better in most cases (like 1-2% better on average). For a final comparison we will take the model and do a 10-fold CV. The AUC score for the two models was:

- **RF AUC score**: 0.9162
- **RF AUC std**: 0.04253
- **RF AUC time**: 178.5 sec
- **XGB AUC score**: 0.92347
- **XGB AUC std**: 0.05079
- **XGB AUC time**: 8.69 sec

In addition to that, regarding the results we can clearly see that the XGB has the edge. However, that edge is very small - a little bit more than 0.007 - and the RF on the other hand seems to be a little bit more consistent given that its std is a tiny bit better - only 0.008 better. Even with this std though we think that XGB performs better in general. Its worst CV round was around 0.88, and the RF's worst CV round was 0.86. Also in terms of the time it took to complete the 10-fold CV, XGB is also the clear winner since it finished in 8.69 seconds, while for the RF it took a bit less than 3 mins - which could maybe be improved by reducing trees since it maybe doesn't need all 2000 for a similar result.

In both cases the Cross-Validation was Stratified and unshuffled, meaning that the ratio of both classes has been kept across splits and that all the data have been used as test set once. Also, in both cases when we did a shuffled CV, the AUC score was a bit lower but the standard deviation of the scores was smaller meaning that the predictions were more consistent. However the results from the unshuffled CV, which are the ones we display above, are probably more meaningful regarding the model's generalization power since it is evaluated on all data. Due to the way we have phrased our problem and the fact that we do not have held out (unseen) test dataset to evaluate our model, we have as we already said relied on the cross-validated result for this project. Another option would be to split the dataset into two parts, but since it effectively has less than 1600 rows (initially 2013 but we dropped users with less than 2 bids) that would make the two resulting sets very small to either train or test on, and therefore our result would be even weaker than this one for which we used CV.

This XGB model which is proposed here seems reasonable according to the expectations from a solution. Moreover, given that the need for this classifier is to filter the bot bidders out, a high enough threshold would be suspicious of very very few human users (False Positives). Also, to attest for the robustness of our implementation further, we also did some rudimentary sensitivity analysis by re-shuffling the rows of the initial dataset and re-evaluating our model - it might just be that the initial shuffling was accidentally very well suited for the task. In five attempts that we permuted the rows of the dataset, all the results were in a (0.92 - 0.929) range, which is quite convincing.

Justification

In response to the question of how adequately does the discussed solution solve the problem at hand, we should turn to the benchmark model that we have earlier specified. From a first glance, by looking at the raw AUC scores, it seems that it would end up in the 189th (out of the 985) place of a very competitive competition. Probably with a little bit more work and more features we could even approach top 50. However, that is not the whole picture because as has been said earlier these models have been evaluated in a test set for which we do not have the labels and our AUC score is a cross-validated result of how our model scores on parts of the training set itself.

In spite of that, we would expect that our model would score similarly - maybe a bit less - on the leaderboard's test set. The clear weakness of course is that the model would be put to predict data it has not seen before. However, we do not think that it would drop more than the very basic XGB model that we started with, before tuning any parameters. The scores in what appeared to be the worst combinations were very rarely dropping below 88% even when tuning the first batch of parameters.

The purpose of this problem though is to find bots, and to some extend it is similar to finding spam e-mails. For such a problem we want to find a bot but we really don't want to ban a legitimate player. So probably the model would be used at some point to the bottom left of the ROC graph, with a very high threshold to reduce the False Positives as much as possible, and even finding a few bots but no real users it is a gain. In addition to that, we have to take into account that the data set was very noisy and that also lowers the expectations in general. In that sense we do think that our solution is a good one, even if it was gonna drop as much as 90% in the kaggle leaderboard test set.

V. Conclusion

Free-Form Visualization

Beginning to reflect on the project, maybe the most important thing about it, is whether it achieves its purpose which is to improve the user experience. For that purpose, it is very important, while detecting the bots, to avoid False Positives because that could potentially ruin the user experience for some other users. For that reason it is necessary to set the correct threshold when it comes to declaring a user as bot. A ROC curve for that matter helps us make that threshold decision since it describes the trade-off for a percentage of True Positives (where we correctly identify a bot) to False Positives (where we misinterpret a normal user as bot).

In the case of this project we computed an ROC curve to get the AUC score in many cases. In Figure 6 we can see the ROC visualization of the final 10-fold evaluation of the XGB approach. This curve is the mean curve of the curves for each fold. The curves from all other validation rounds can be seen as well, slightly dimmer in the back. We can see that even with the worse split, we can get an improvement out of the model. In the best case scenarios, we can have perfect score (find almost all True Positives), with only superficial losses. For example, in the worst cv split, for a 0.05% of FPs we would have 0.3% TPs and in the best case the TPS would be around 100%. According to the mean, for a 0.05% False Positives we can filter out around 0.55% of the bots. However given how skewed the dataset is, maybe 0.05% is still a lot, and we should lower our standards even further.

Besides finding the optimal threshold, another very important thing during the beginning of the project was to find the relevant features that could potentially be used by our model. Feature creation was also the hardest part. At the beginning we could only guess what could be the best features that describe and data and we decided to go with a bunch of temporal features and features made by counts of categorical variables.

The algorithms that we used for the project though can give us an insight regarding our earlier tinkering and intuition. More specifically, the XGB Python API allows us to get a score for each feature on a training round. These scores come either as the average gain of the feature across all trees, or more simply as the number of times the feature was used by a tree, across all trees. In this case we thought that the average gain would be more accurate than the counts because it also captures **when** the feature was selected by the tree. A feature that was selected first would probably have more gain than one that was selected last and that is a qualitative characteristic captured by the 'gain' mode.

In addition to that, due to the fact that we were running a 10-fold CV, we took the average gain of all folds. Every gain score as we just said is also the average gain of all boosting rounds so every score here is an average of averages. The scores can be seen in Figure 7, where it seems that the best features were the mean count of bids across a user's devices, the mean interval,

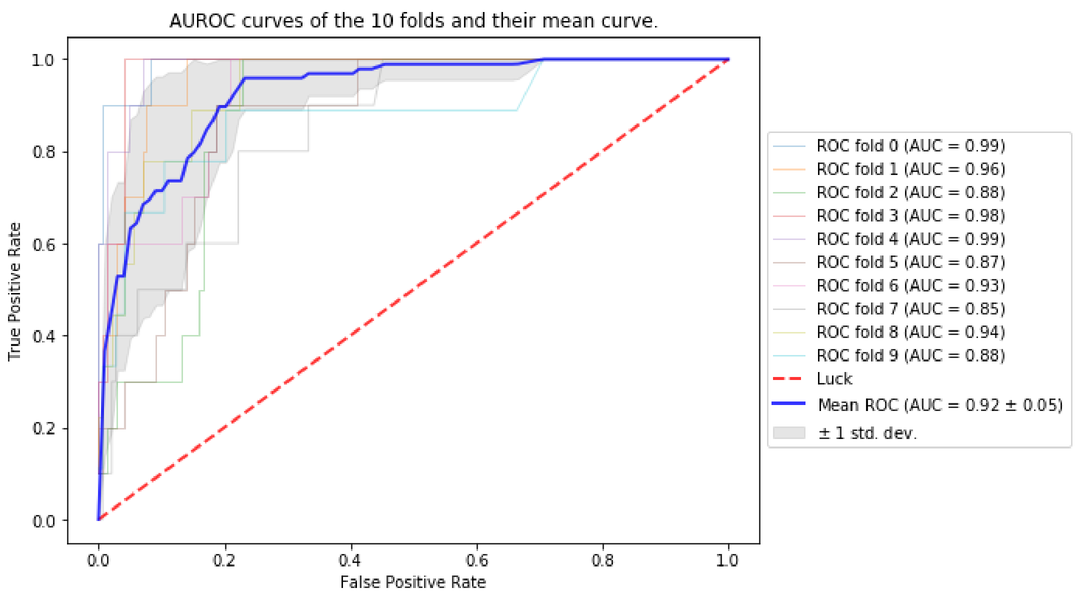


Figure 6: The ROC curves for each of the 10 folds that we used for the final evaluation of our XGB model, along with their respective AUC scores.

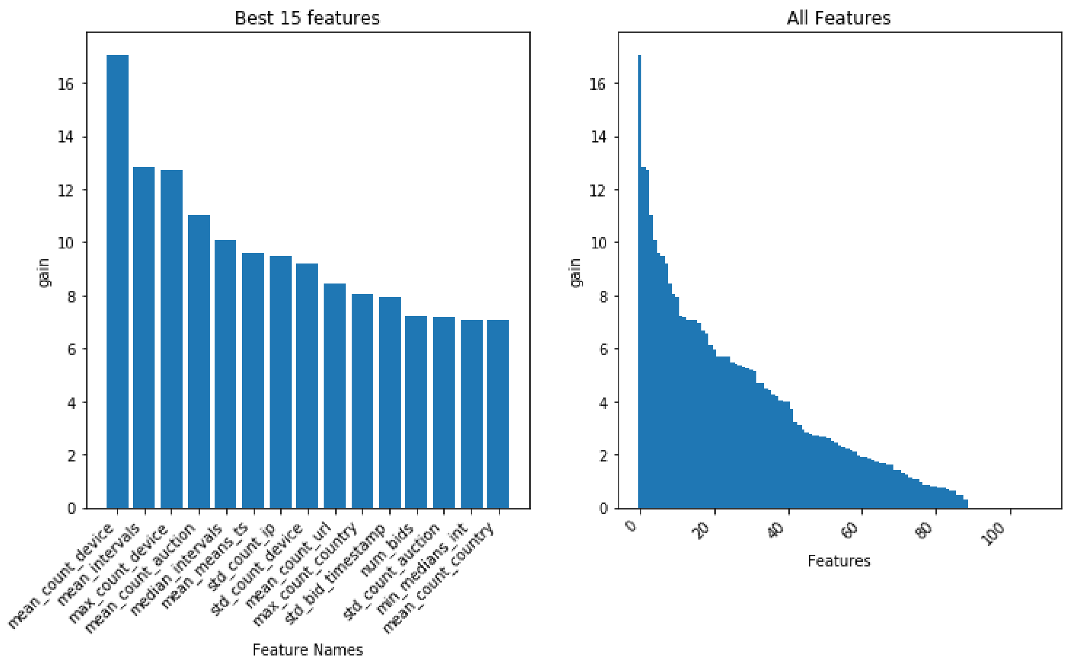


Figure 7: Plots according to how the features rank given their average 'gain' across all boosting rounds, and all folds. On the left we can see the 15 best with their feature names, and in the right all of the 109 features.

the count of bids of the device the user is mostly betting with and the mean count of bids across the bidder's auctions. mean_count_device, mean_intervals and mean_count_auction rank very high, no matter the ranking metric (instead of average gain we could also take the sum, or the average and sum of number of times a feature was used in the trees).

In general we see that good features come across all categories of the bids dataset. Maybe the most represented ones are 'device', 'auction' and the intervals that we got out of 'time'.

Also, it seems that from the 65 features of the timeseries per auction, only 2 are in the top 15 list, the mean of the timestamp means and the minimum of the interval medians - both timestamp means and interval medians being across auctions.

Reflection

In retrospective, i would say that the project deals in a comprehensive way with a very realistic and interesting problem, which is the problem of the deterioration of the user experience due to bot parties among users in an online auctions platform. In such a case where the bots interact or co-exist with users, historical data about the users' behaviour can be detrimental to finding them. In the case of the auctions site that we dealt with, the data they produced proved useful and enough to build a classification system that deals with the problem adequately. The project that built this system had two main parts, the Feature Construction and the Classification. In the first we made relevant features and in the latter we used them with some popular classification models.

The main part of the problem, which was also the most difficult and challenging, was to find the best features that adequately describe the data. Finding them was particularly challenging because of the timeseries nature of the data. There could be many threads that describe differently the behaviour of a bidder depending on the angle it could be viewed. For example one could see the bidding behaviour of an individual according by the countries he or she was bidding from, or by the auctions etc. In addition to the temporal characteristics there were also some more static ones, which in the end, in some cases they appear to be more robust.

With the present solution, we showed that getting the counts of the categorical values, and making temporal features by the crude timestamps, the intervals, or the sequences of timestamps and intervals that occur if a bidder is looked as someone with multiple timeseries grouped by the different auctions he or she participated in, can prove to be a very good way to solve such a problem. Of course the scope is a little bit narrow here since we are only referring to auction websites. However, a similar solution (meaning statistics from categorical and temporal values) could possibly be used for other similar datasets that include a timeseries history of someone's actions.

Improvement

The thing that could potentially improve the solution more than anything else would be to spend more time in Feature Creation. Creating the features was also the most difficult part of the project. One thing we could do, could be to extend the groupings by which we did the 65 timeseries features per auctions. We could do that for devices and countries as well. However, we already saw that in terms of performance these were the lowest performing features.

Moreover, the obvious thing to do is to flat out create more features, but in addition to that, there could also be more ways to create them. In terms of the simple creation, one possibility that we did not explore could be to not only think in terms of intervals between someone's betting but also to the bidder's activity in connection to what happened earlier, or what happens regularly in the same auction, or from the same country etc. Apart from that though it is hard to imagine what would be the best kind of features and the potential impact of the ones just mentioned. More field knowledge and exploration of the data would be necessary to give us insight. In addition to that, another possible and more exotic alternative, could be to try use some kind of a NN (like an CNN or LSTM or combine them) to create features given a bidder's dataset, but we did not really explore or gave that direction much though.

Other minor improvements that could be made would be to use more algorithms than the Random Forest and XGB, or tune more extensively. Regarding the first, we could probably have also used a SVM or an ensemble method that would combine the different learners. The reason we did not do it here is mainly because of time complexity, given the setting we were running the experiments on. Adding those could be interesting, however we do not think that the result would increase significantly, if there was any in the first place.

In addition to that, when tuning the XGB, we tuned the tree and regularization parameters with a learning rate of 0.3, and when tuning the learning rate we again found that 0.3 was the best rate. It would be maybe interesting to see whether a smaller initial learning rate could bring better results, and whether the optimal learning rate would be again the same after tuning the tree and regularization parameters. That would indicate that when optimizing the tree or regularization parameters, they are optimized for this specific learning rate and that it essentially does not make sense for the learning rate to be optimized in that way - late after the other two - since it remains the same when tuned last.

References

[1] Strayer, W. Timothy, et al. "Botnet detection based on network behavior." *Botnet Detection.* Springer, Boston, MA, 2008. 1-24.

[2] Cooke, Evan, Farnam Jahanian, and Danny McPherson. "The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets." *SRUTI* 5 (2005): 6-6.

[3] Cresci, Stefano, et al. "DNA-inspired online behavioral modeling and its application to spambot detection." *IEEE Intelligent Systems* 31.5 (2016): 58-64.

[4] Golle, Philippe, and Nicolas Ducheneaut. "Preventing bots from playing online games." *Computers in Entertainment (CIE)* 3.3 (2005): 3-3.

[5] https://datadome.co/how-to-detect-malicious-bots/

[6] https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot

[7] Fawcett, Tom. "ROC graphs: Notes and practical considerations for researchers." Machine learning 31.1 (2004): 1-38.

[8] https://www.kaggle.com/c/facebook-recruiting-iv-human-or-bot/discussion

[9] https://rinzewind.org/blog-en/2015/my-take-on-kaggles-facebook-recruiting-iv-human-or-robot.html

[10] http://www.thomas-robert.fr/en/kaggles-human-or-robot-competition-feedback/

[11] http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[12] http://xgboost.readthedocs.io/en/latest/

[13] http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html

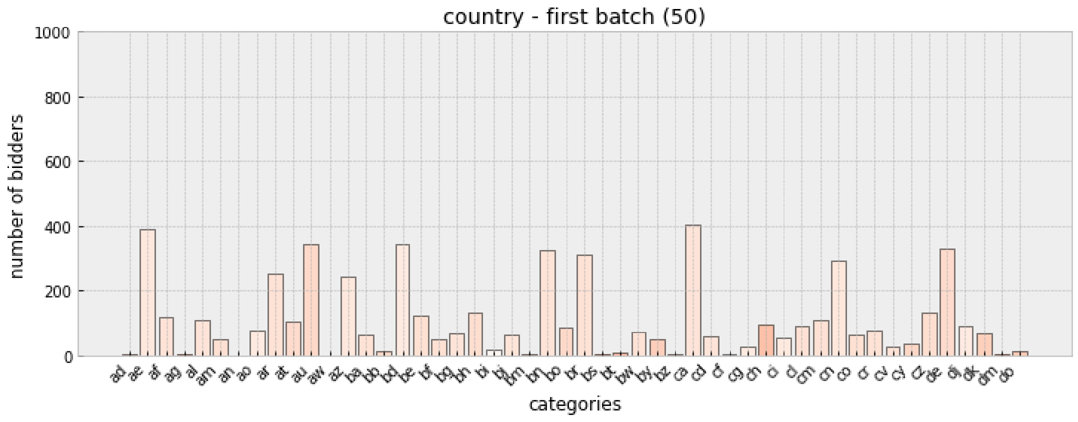
[14] http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc_crossval.html

[15] http://xgboost.readthedocs.io/en/latest/python/python_api.html

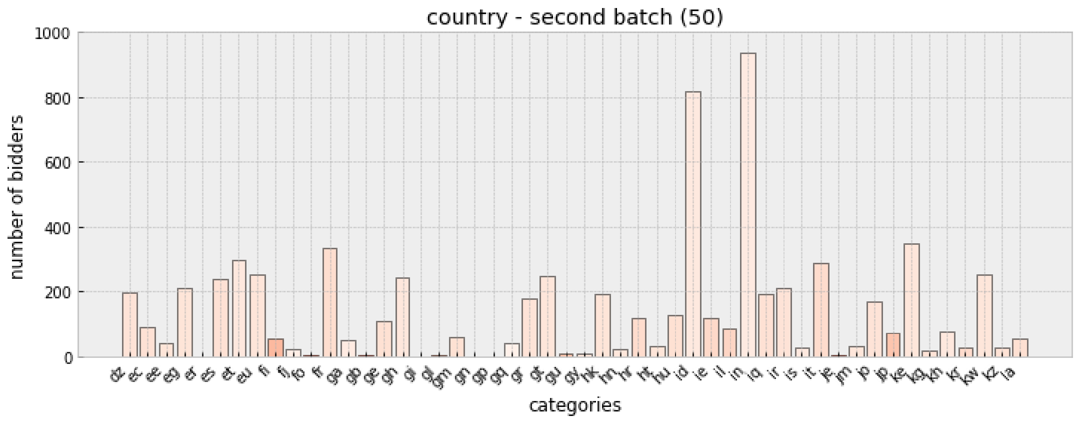
[16] https://cambridgespark.com/content/tutorials/hyperparameter-tuning-in-xgboost/index.html

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

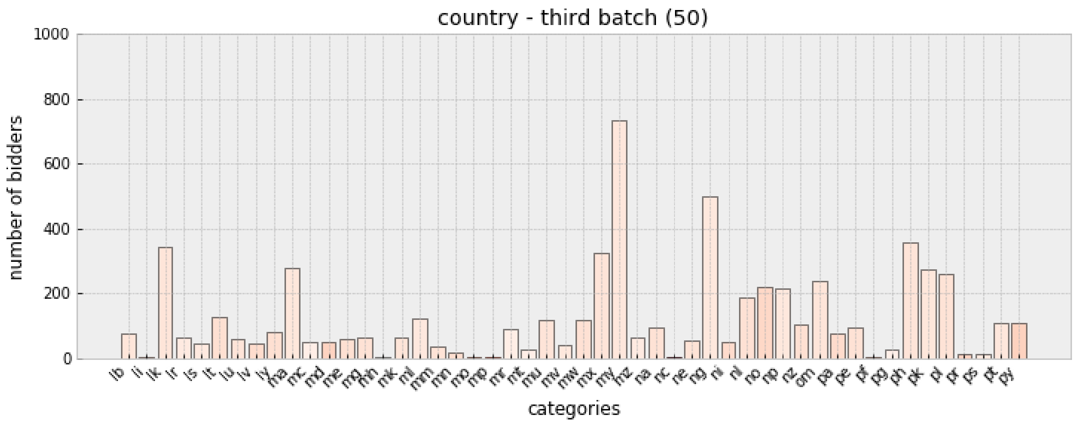
A. Visualizations Appendix



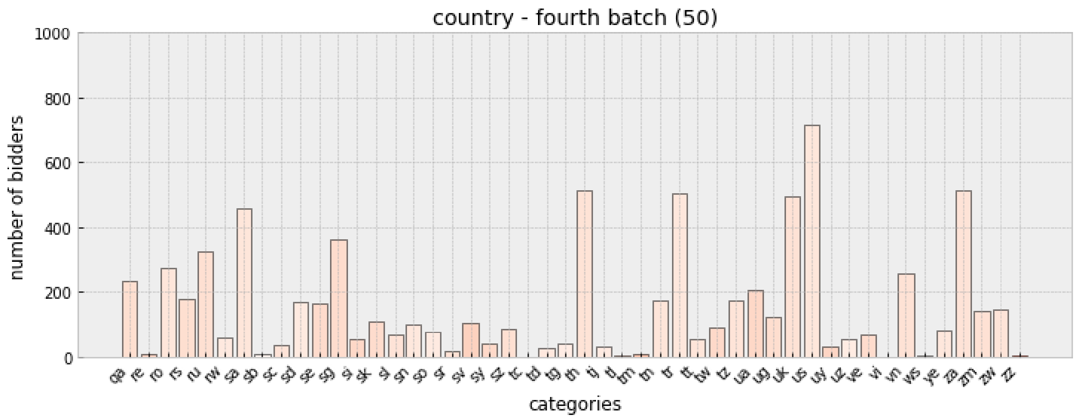
(a) First batch of 50 countries, their counts of bids, and ”contamination” (red color intensity).



(b) First batch of 50 countries, their counts of bids, and ”contamination” (red color intensity).

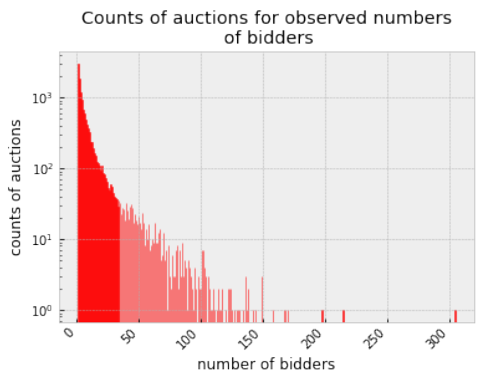


(c) Third batch of 50 countries, their counts of bids, and ”contamination” (red color intensity).

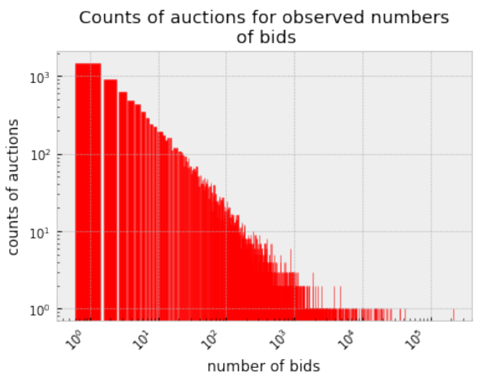


(d) Fourth batch of 48 (49 with NaNs) countries, their counts of bids, and "contamination" (red color intensity).

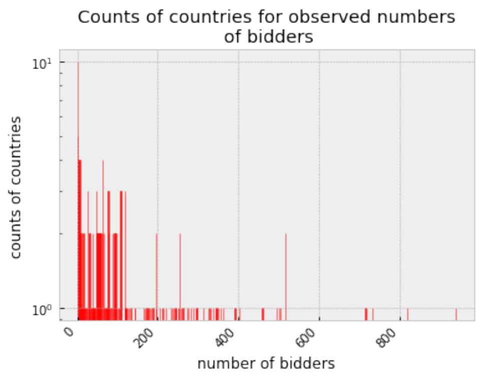
Figure 8: In this figure, a bar chart for every country of the 199 (one for NaN countries) in the dataset is shown split in four groups of 50 or 49. The length of the bar shows the number of bidders and its red color intensity shows the *contamination* of this country with bots. A drawback with this plot is that the small countries where the most interesting results lie are getting lost due to the hight of the countries with many bidders.



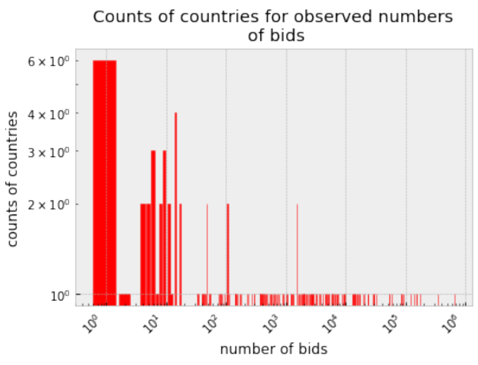
(a) Auction counts with the number of bidders



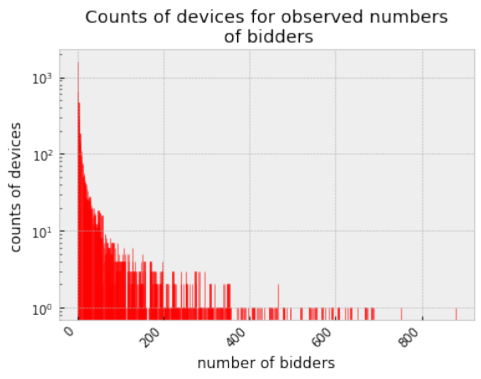
(b) Auction counts with the number of bids



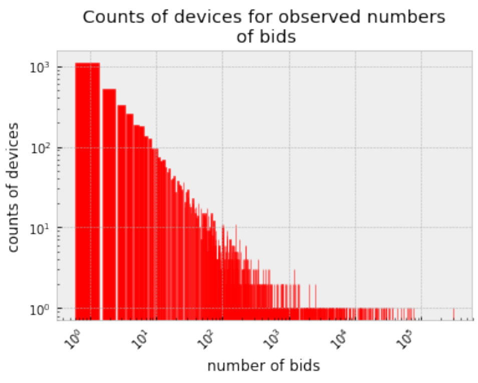
(c) Country counts with the number of bidders



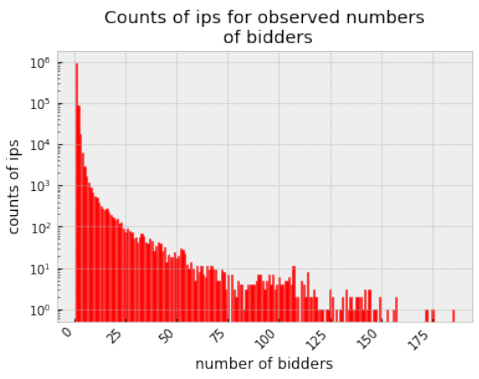
(d) Country counts with the number of bids



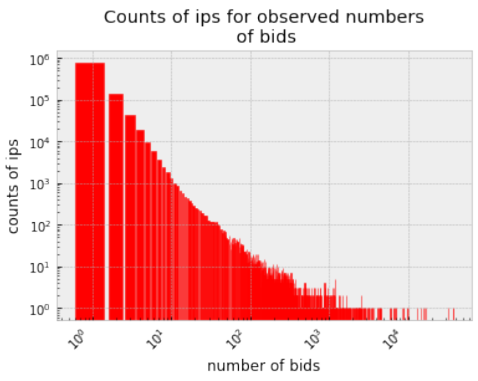
(e) Device counts with the number of bidders



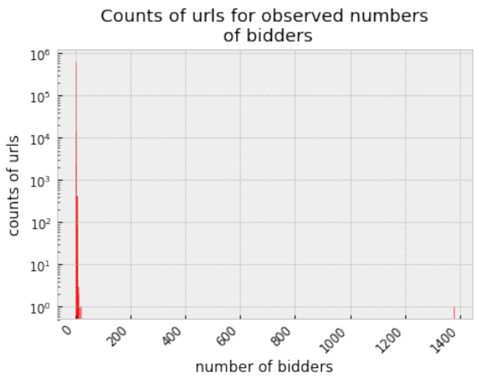
(f) Device counts with the number of bids



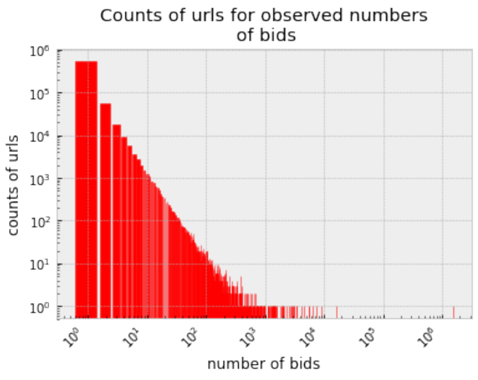
(g) IP counts with the number of bidders



(h) IP counts with the number of bids



(i) Url counts with the number of bidders



(j) Url counts with the number of bids

Figure 9: In this figure, the counts of 5 categorical variables, namely auctions, countries, devices, ips and urls, are plotted as a function of their number of bidders and number of bids.