Machine Learning Nanodegree

Capstone Project

Kaggle Competition: Human or Robots?

Jesús A. Martínez V.

December 31, 2017

1. Project Overview

1.1. Domain Background

Artificial Intelligence (and Machine Learning in particular), has aroused a great amount of interest in the population in the recent years due to its wide range of application and to the large number of successful products currently on the market that embrace machine learning at their core, such as Netflix's accurate recommender system [1], Google's amazing translation platform [2] and even chatbots that provide medical assistance, such as MedWhat [3].

But every coin has two sides. Even though the progress and comfort that technology and automation has brought to mankind is undeniable, with it, new problems such as DDoS attacks [4] and click fraud performed by botnets [5] have arose.

Our goal in this project is, precisely, to develop a classifier capable of detecting suspicious bot activity in a fictional auction website that has been experiencing a concerning exodus of users due to unfair robot traffic in their site. This problem corresponds to the following competition posted by Facebook at Kaggle: Human or Bot?.

1.2. Problem Statement

Goal

To identify online auction bids that are placed by "robots", helping the site owners easily flag these users for removal from their site to prevent unfair auction activity.

Type of machine learning problem

Given our aim is to categorize a user as a robot or human by its bidding activity, we're clearly working on an instance of supervised learning. Moreover, we will build a classifier that'll help us reach our goal and prevent unfair auction activity in the site.

Target function

$$C : Bids \times Bidders \rightarrow \{0,\ 1\}$$

Here $Bids$ is the bids dataset (described in the next section), $Bidders$ is the bidders (users) dataset (also described in the next section), and the expected output is a label, where 0.0 corresponds to a human user and 1.0 to a robot bidder.

Target function representation

We'll build a classifier to approximate our target function. The algorithms the we will explore are:

- Support Vector Machines.
- K-Nearest Neighbors.
- Gaussian Naïve Bayes.
- Gradient Boosting.
- Random Trees.
- Decision Tree.
- Perceptron.
- Multilayer Perceptron.
- Logistic Regression.
- Linear Discriminant Analysis.

1.3. Datasets and Inputs

To tackle this problem, we will use the data already available as part of the Kaggle Competition. There are two datasets to consider:

- **Bidders dataset:** Includes a list of bidder information, including information such as their id, payment account and mailing address. Several of these fields are obfuscated to protect privacy.
- **Bids dataset:** Includes 7.600.000 bids spread across different auctions. The bids in this dataset are all made from mobile devices.

Given that our problem comes from a Kaggle Competition, we didn't need to perform any action to obtain the data; it was given as part of the competition's initial resources.

File descriptions

- **train.csv**: Training set from the bidder dataset.
- **test.csv**: Test set from the bidder dataset.
- **bids.csv**: Bids dataset.
- **sampleSubmission.csv**: Sample submission in the correct format.

Datasets fields

For the **bidder dataset**:

- **bidder_id**: Unique identifier of a bidder.
- **payment_account**: Payment account associated with a bidder. Obfuscated to protect privacy.
- **address**: Mailing address of a bidder. Obfuscated to protect privacy.
- **outcome**: Label of a bidder indicating if it is a robot or not. A value of 1.0 translates into a bidder being a robot, whereas a value of 0.0 means the user's a human. The outcome was half hand labeled and half stats-based. There are two types of "bots" with distinct levels of proof:
    - Bidders who are identified as bots/fraudulent with strong proof. Their accounts have been already banned from the auction site.
    - Bidders who may have just started their business/clicks or their stats deviate from system wide average. There is no clear proof that they are bots.

For the **bids dataset**:

- **bid_id:** Unique id for the bid.
- **bidder_id:** Unique identifier of a bidder. Corresponds to a bidder in the bidder dataset.

- **auction:** Unique identifier of an auction.
- **merchandise:** The category of the auction site campaign, which means the bidder might has come to this site by way of searching for a product category. For instance, if the bidder arrived at the site by a “home goods” search, but ended up bidding for “sporting goods”, this field will contain “home goods”. This categorical field could be a search term or an online ad.
- **device**: Phone model of a visitor.
- **time:** Time when the bid was made (obfuscated to protect privacy).
- **country:** The country that the IP belongs to.
- **ip:** IP address of a bidder (obfuscated to protect privacy).
- **url:** URL where the bidder was referred from (obfuscated to protect privacy).

1.4. Solution Statement

To tackle the problem described in Section 1.2, we will use Supervised Learning. We will build a binary classifier, given that our task is to build a model capable of successfully labeling a user as a robot or human based on their auction activity behavior.

We will make a train-validation split on our training set (train.csv; see Section 1.3 for more details) so we can hold out a portion of it to measure the performance of the model on unseen data and tune accordingly.

The metric we will use is Area Under Receiver Operating Characteristic Curve [6], which is the metric accepted in the Kaggle Challenge this problem is based on. More details about the metric on the next section.

1.5. Evaluation Metrics

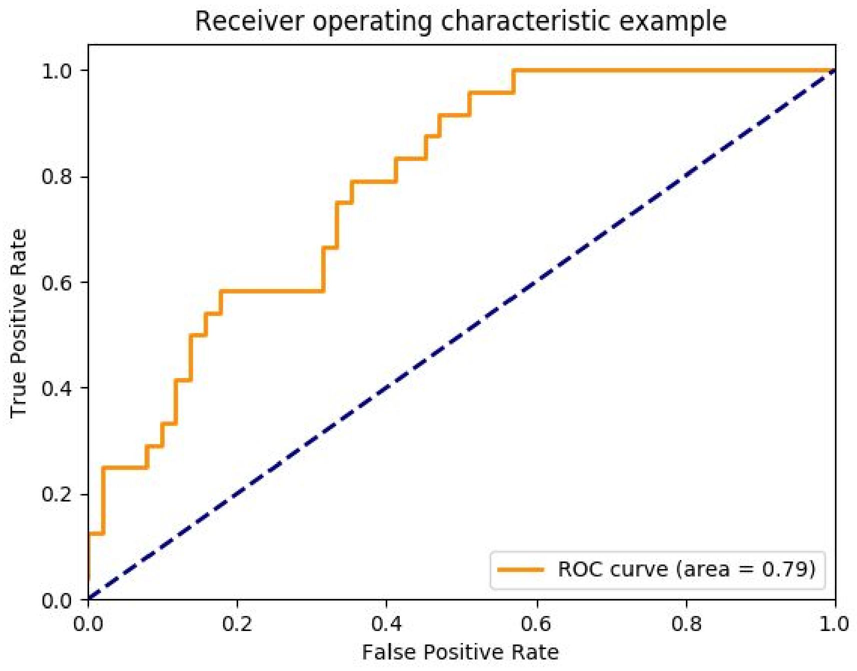
For this project we will use the Area Under Receiver Operating Characteristic Curve as a performance metric, given that’s the one used in the Competition and, hence, the benchmark we want to compare our model with.

Receiver Operating Characteristic Curve is a commonly used tool for measuring the performance of a binary classifier. It is a plot that shows the relation between the True Positive Rate and False Positive Rate for every possible decision threshold. These rates are defined as follows:

$$True\ Positive\ Rate = Sensitivity = \frac{True\ Positives}{All\ Positives}$$

$$False\ Positive\ Rate = 1 - Specificity = 1 - \frac{True\ Negatives}{All\ Negatives} = \frac{False\ Positives}{All\ Negatives}$$

A typical ROC plot could look like this:



The Area Under Receiver Operating Characteristic Curve is the proportion of the square that falls below and left of the ROC curve (shown in orange). A score of 0.5 means our classifier is no better than random guessing, and 1.0 is the score of a perfect classifier.

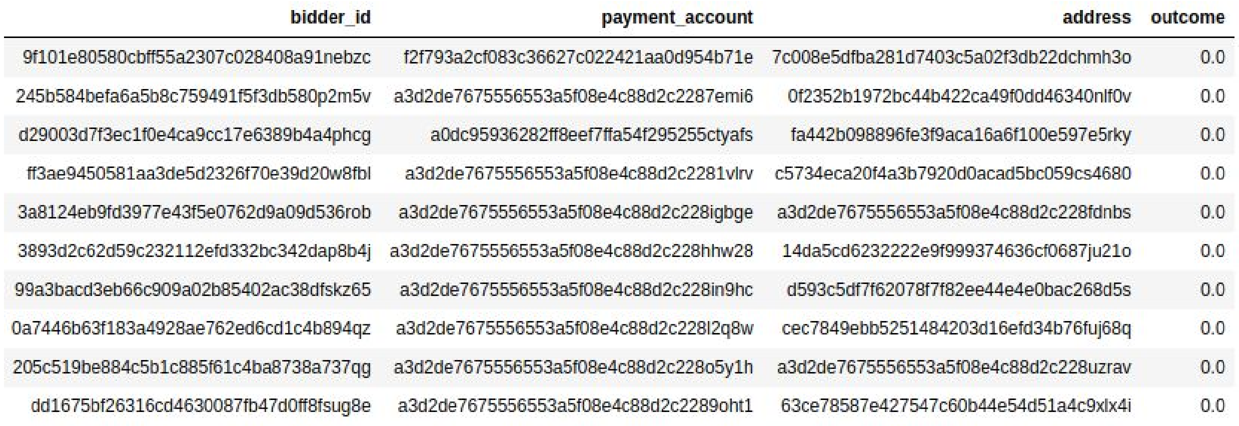
One very appealing advantage of Receiver Operating Characteristic Curve is that they are insensitive to poorly predicted "probabilities". This means that if what our classifier outputs is an actual probability between 0.0 and 1.0, or simply a number between, say, 0.9 and 1.0 it won't make any difference in the resulting curve, because it focuses on how well the model separates the two classes [7]. Thus, we can think of AUC as the probability of the classifier ranking higher a randomly chosen positive observation than a randomly chosen negative observation, which makes it a great metric for datasets with highly unbalanced classes.

2. Analysis

2.1. Data Exploration

Given we are solving a problem coming from a Kaggle competition, our data was already collected and handed to us in its main page. For a more thorough description, please refer to Section 1.3.

The data is splitted in both bidders and bids dataset. First, let's explore **bidders** data in **train.csv**. Following are some samples of it:



bidder_id	payment_account	address	outcome
9f101e80580cbff55a2307c028408a91nebzc	f2f793a2cf083c36627c022421aa0d954b71e	7c008e5dfba281d7403c5a02f3db22dchmh3o	0.0
245b584befa6a5b8c759491f5f3db580p2m5v	a3d2de7675556553a5f08e4c88d2c2287emi6	0f2352b1972bc44b422ca49f0dd46340nlf0v	0.0
d29003d7f3ec1f0e4ca9cc17e6389b4a4phcg	a0dc95936282ff8eef7ffa54f295255ctyafs	fa442b098896fe3f9aca16a6f100e597e5rky	0.0
ff3ae9450581aa3de5d2326f70e39d20w8fbl	a3d2de7675556553a5f08e4c88d2c2281vlrv	c5734eca20f4a3b7920d0acad5bc059cs4680	0.0
3a8124eb9fd3977e43f5e0762d9a09d536rob	a3d2de7675556553a5f08e4c88d2c228igbge	a3d2de7675556553a5f08e4c88d2c228fdnbs	0.0
3893d2c62d59c232112efd332bc342dap8b4j	a3d2de7675556553a5f08e4c88d2c228hhw28	14da5cd6232222e9f999374636cf0687ju21o	0.0
99a3bacd3eb66c909a02b85402ac38dfskz65	a3d2de7675556553a5f08e4c88d2c228in9hc	d593c5df7f62078f7f82ee44e4e0bac268d5s	0.0
0a7446b63f183a4928ae762ed6cd1c4b894qz	a3d2de7675556553a5f08e4c88d2c228l2q8w	cec7849ebb5251484203d16efd34b76fuj68q	0.0
205c519be884c5b1c885f61c4ba8738a737qg	a3d2de7675556553a5f08e4c88d2c228o5y1h	a3d2de7675556553a5f08e4c88d2c228uzrav	0.0
dd1675bf26316cd4630087fb47d0ff8fsug8e	a3d2de7675556553a5f08e4c88d2c2289oht1	63ce78587e427547c60b44e54d51a4c9xlx4i	0.0

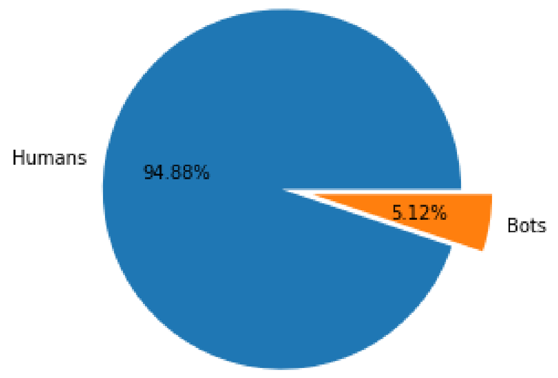
As we can see above, all the relevant fields (besides outcome) are obfuscated.

Following is a summary of the descriptive statistics of the bidders dataset:

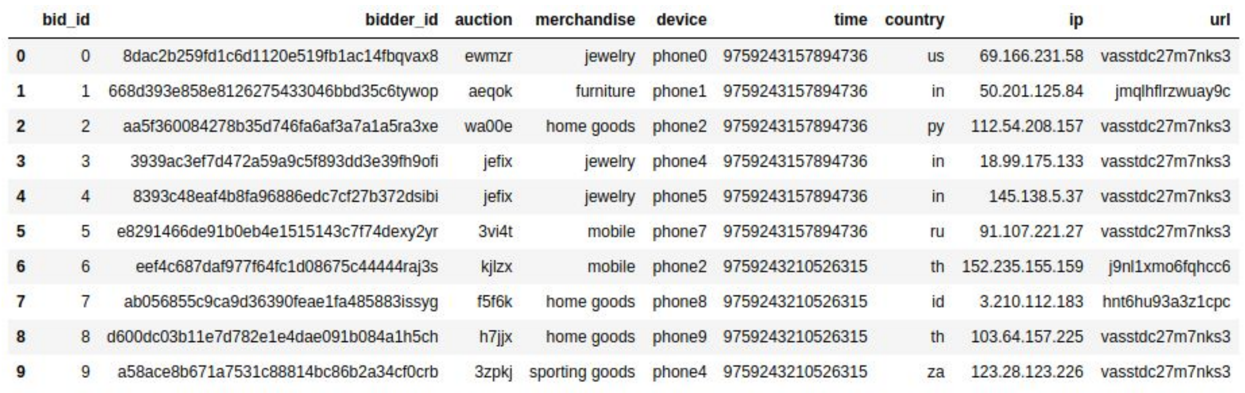
CATEGORY	**COUNT**	**PROPORTION**
HUMAN	1910	94.88%
ROBOTS	103	5.12%
TOTAL	2013	100%

The first thing we notice is that our dataset is highly unbalanced, where 1910 out of 2013 records represent human bidders (94.88%), whereas only a small percentage (5.12%) of the bidders were flagged as bots (103 out of 2013).

For a better understanding of this situation, let's visualize it:



Let’s now explore **bids** dataset. Following are some samples extracted from it:



	bid_id	bidder_id	auction	merchandise	device	time	country	ip	url
0	0	8dac2b259fd1c6d1120e519fb1ac14fbqvax8	ewmzr	jewelry	phone0	9759243157894736	us	69.166.231.58	vasstdc27m7nks3
1	1	668d393e858e8126275433046bbd35c6tywop	aeqok	furniture	phone1	9759243157894736	in	50.201.125.84	jmqlhflrzwuay9c
2	2	aa5f360084278b35d746fa6af3a7a1a5ra3xe	wa00e	home goods	phone2	9759243157894736	py	112.54.208.157	vasstdc27m7nks3
3	3	3939ac3ef7d472a59a9c5f893dd3e39fh9ofi	jefix	jewelry	phone4	9759243157894736	in	18.99.175.133	vasstdc27m7nks3
4	4	8393c48eaf4b8fa96886edc7cf27b372dsibi	jefix	jewelry	phone5	9759243157894736	in	145.138.5.37	vasstdc27m7nks3
5	5	e8291466de91b0eb4e1515143c7f74dexy2yr	3vi4t	mobile	phone7	9759243157894736	ru	91.107.221.27	vasstdc27m7nks3
6	6	eef4c687daf977f64fc1d08675c44444raj3s	kjlzx	mobile	phone2	9759243210526315	th	152.235.155.159	j9nl1xmo6fqhcc6
7	7	ab056855c9ca9d36390feae1fa485883issyg	f5f6k	home goods	phone8	9759243210526315	id	3.210.112.183	hnt6hu93a3z1cpc
8	8	d600dc03b11e7d782e1e4dae091b084a1h5ch	h7jjx	home goods	phone9	9759243210526315	th	103.64.157.225	vasstdc27m7nks3
9	9	a58ace8b671a7531c88814bc86b2a34cf0crb	3zpkj	sporting goods	phone4	9759243210526315	za	123.28.123.226	vasstdc27m7nks3

As we can see above, fields such as time, bidder_id, auction, device and url are obfuscated.

This doesn't mean they won't be useful for our model, but surely make things a bit harder to interpret for us.

Following is a summary of the descriptive statistics of the bids dataset:

CATEGORY	**COUNT**
BIDS	7,656,334
BIDDERS	6,614
AUCTIONS	15,051
DEVICES	7,351

COUNTRIES	200
IPs	2,303,991
URLs	1,786,351
MERCHANDISE CATEGORIES	10

As we can see, there are a vast amount of bids (7,656,334) distributed along 15,051 auctions.

Also there are 6,614 distinct bidders in this dataset, which contrasts with the 2,013 in the training set.

These bids were made from 7,351 distinct mobile devices models, from 200 countries and from 2,303,991 different IP addresses.

All the bids fall in one of ten categories.

Finally, these came from 1,786,351 distinct URLs.

The following table shows statistics grouped by bidder for each category seen above:

CATEGORY	**MEAN**	**MEDIAN**	**MODE**	**MIN**	**MAX**
BIDS	1157,595	18	1	1	515,033
AUCTIONS	57,808	10	1	1	1,726
DEVICES	73,438	8	1	1	2,618
COUNTRIES	12,724	3	1	0	178
IPs	544,096	12	1	1	111,918
URLs	290,745	5	1	1	81,376

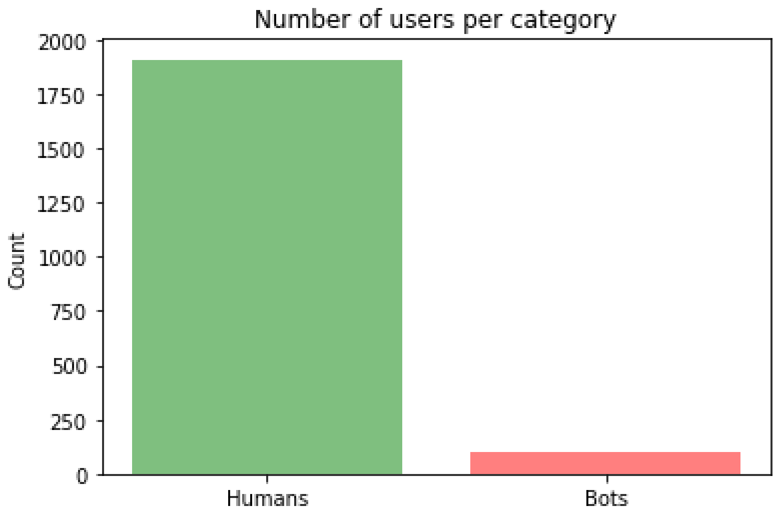
One key aspect that we can notice is the huge distance between the mean and the median in the six features (auctions, bids, countries, ips, urls and devices) selected above. Also, the range of values for each feature is widely spread. Take bids for instance: The highest number of bids a user made is 515,033, while the most repeated number of bids per user is 1 (also the minimum). Clearly, the highest values have a stronger influence on the mean, which is around 1,157 bids per user. The fact that the value that sits at the center of the distribution of bids per user is 18 (i.e., the median) hints us about the skewness of it.

The same analysis applies for the remaining features.

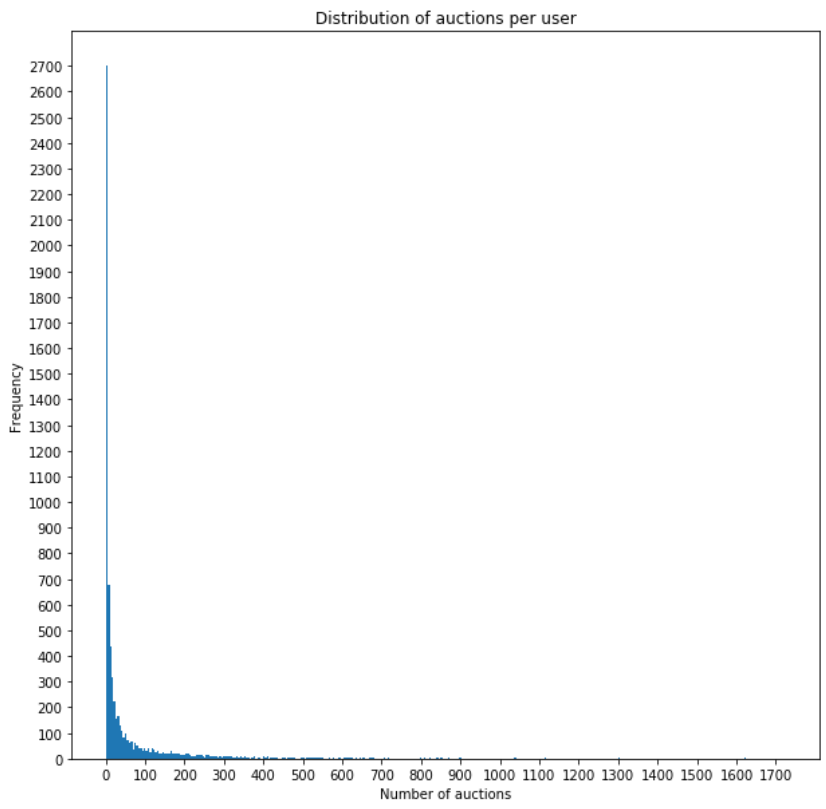
2.2. Exploratory Visualization

Given our aim is to determine which information or bidding behavior is characteristic of bot users, we are going to focus on visualizing only data related to bidders.

Let's start by seeing again the distribution between bidders labeled as bots and as humans:

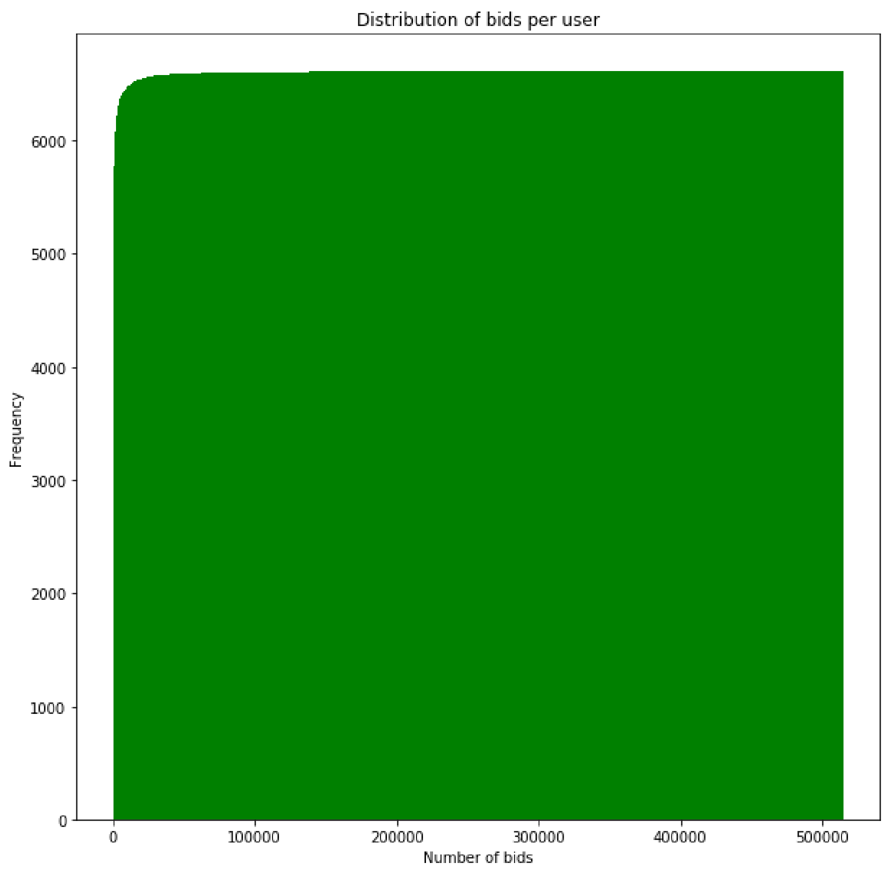


Let's now visualize auctions per user:



The distribution of auctions per user is skewed to the right. This means that most users participate in less than 10 auctions. This is noticeable in the highest peak at the far left in the graph.

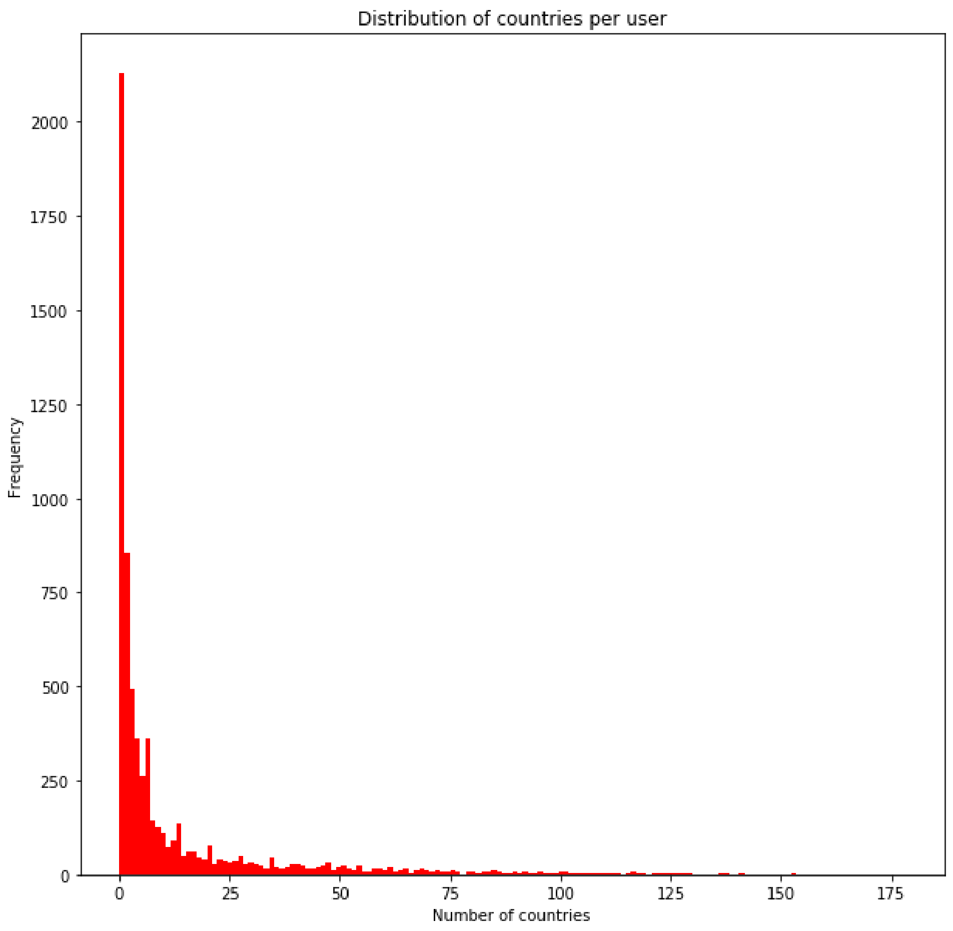
Less than 100 users participate in 100 auctions or more. As we approach to the tail of the distribution, we can notice tiny bins which are most likely comprised of less than 10 points, which means 10 or less users participate in a high number of auctions.



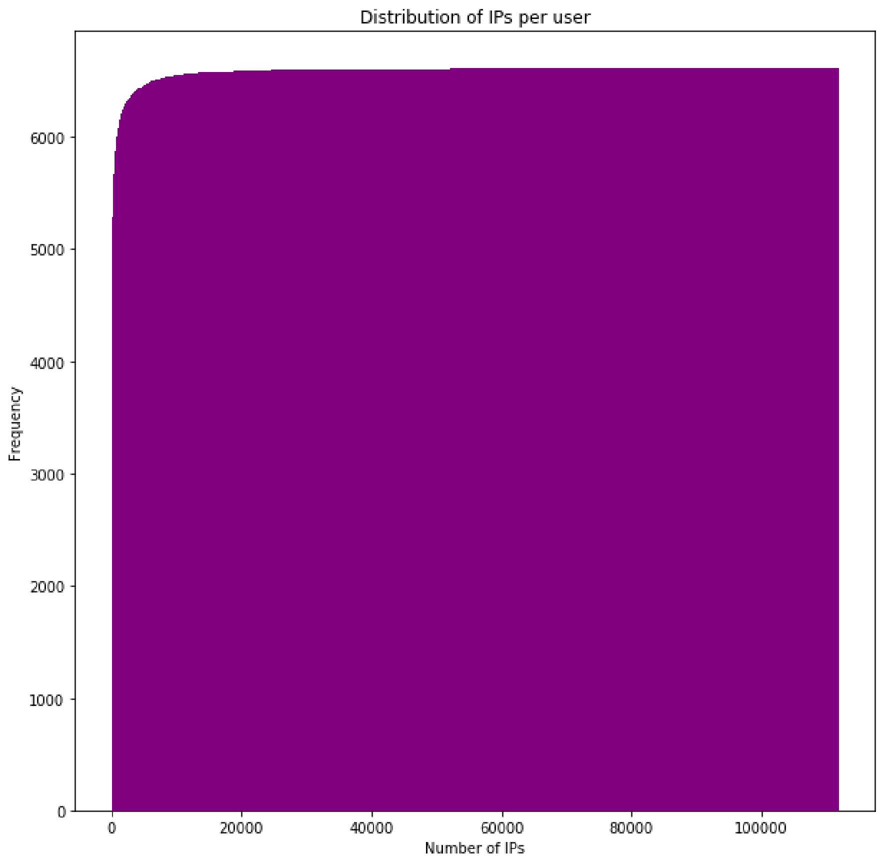
An alternative to the classic histograms, is to visualize the cumulative effect of each class in the overall distribution of data.

As we can see in the above plot, the distribution peaks or gains most of its width in the far left. This means that most of the data is comprised of users that bidded only once. This makes sense, given the fact that we know the mode (i.e., the most repeated value) is 1, and the median is 18.

As we move to the right side of the graph, we can the slope is almost zero, which translates in a negligible contribution to the density of the distribution from the other classes (users that bidded hundreds of thousands of times).



This graph is very similar to the histogram of auctions per user. The same pattern applies here: Most of the data is located at the left side of the graph, which means that most users bidded from less than two countries. Then, we have a skewed distribution to the right.



This graph is very similar to the cumulative histogram of bids per user. The same pattern applies here: Most of the data is located at the left side of the graph, which means that most users bidded from less than a couple of different IP addresses.

Yet again, the contribution of users with thousands of distinct bidding IPs is negligible.

2.3. Algorithms and Techniques

The classifier used was a Random Forest. After trying several other algorithms, Random Forest showed the best performance among all of them. For more details about the algorithm selection process, please refer to Section 3.3.

Random Forest is an ensemble learning method for both classification and regression. At its core, it trains a set of decision trees, outputting the class that repeats the most (mode) among the trees that compose the forest.

One of the great advantages of Random Forest is its robustness against overfitting.

The most relevant parameters that can be tuned to optimize the model are:

- Number of estimators (the number of trees in the forest).

- Maximum depth (Maximum depth of each tree).
- Minimum of samples per split (Minimum number of samples required to split an internal node of the trees).
- Minimum number of samples per leaf node.
- Criterion (The function to measure the quality of a split. It can be either Gini impurity or entropy for information gain).

Although the selected final model is a Random Forest, Decision Tree and Gradient Boosting models achieved similar score and ranked among the top three classifiers.

A Decision Tree is a non-parametric algorithm that uses an underlying graph-like structure that represents decision or branching points that lead to a particular outcome. This graph-like structure makes the algorithm a more relatable and understandable tool for humans because the inner working of it is clearly examinable.

The most relevant parameters that can be tuned to optimize a Decision Tree are:

- Maximum depth (Maximum depth of each tree).
- Minimum of samples per split (Minimum number of samples required to split an internal node of the trees).
- Minimum number of samples per leaf node.
- Criterion (The function to measure the quality of a split. It can be either Gini impurity or entropy for information gain).

Gradient Boosting, like Random Forest, is an ensemble learning method for both classification and regression. While Random Forest, relies on fully grown trees (high variance, low bias), Gradient Boosting opts for many weak learners (high bias, low variance) as a mean to reduce the error by reducing bias, as opposite of Random Forest that reduces error by reducing variance. Gradient Boosting uses decision trees as default weak learners, but it can work with other models too. In our case, we use Gradient Boosting with decision trees.

The most relevant parameters that can be tuned to optimize a Gradient Boosting classifier are:

- Number of estimators (the number of weak learners).
- Maximum depth (Maximum depth of each tree).
- Minimum of samples per split (Minimum number of samples required to split an internal node of the trees).
- Minimum number of samples per leaf node.
- Criterion (The function to measure the quality of a split. It can be either Gini impurity or entropy for information gain).
- Loss (loss function to be optimized).

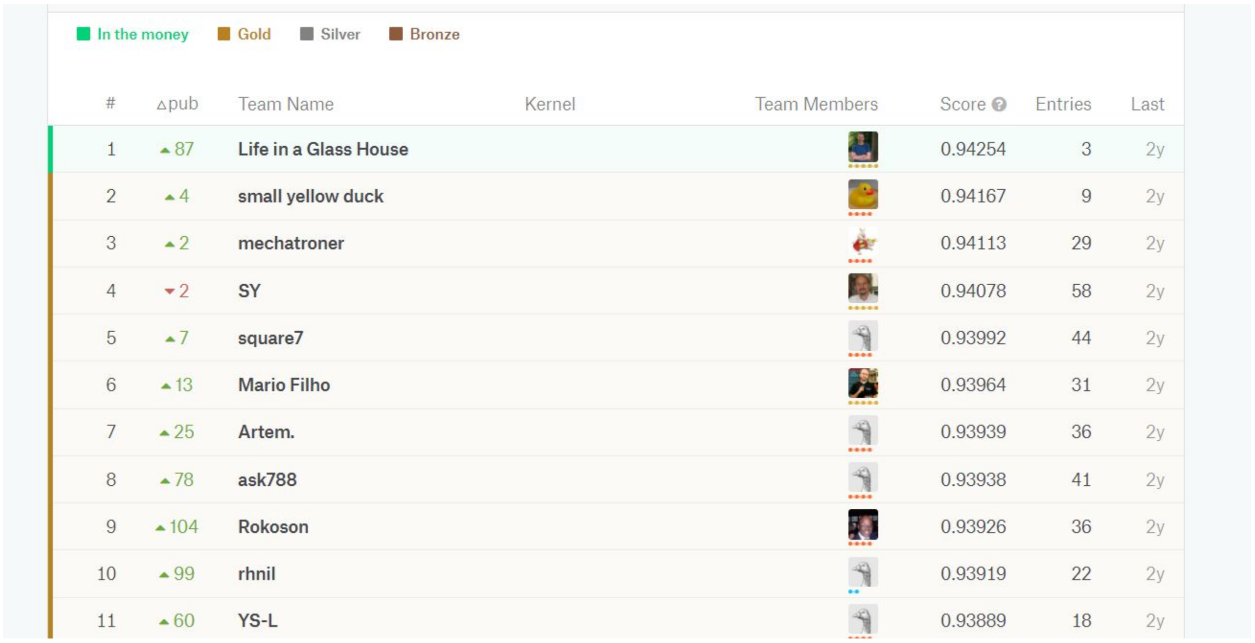
- Learning rate (this parameter shrinks the contribution of each tree by the specified amount. There is a trade-off between the learning rate and the number of estimators).

2.4. Benchmark

Given the nature of our problem, our benchmark will simply be the score achieved by the winner team of the Kaggle Competition on the Private Leaderboard, which is calculated with approximately 70% of the test data:

$$Benchmark\ Score\ =\ Benchmark\ AUC\ =\ 0.94254$$

The picture below shows the final standings (Winner and Gold – Top 11 teams) for the competition (now closed):



🟩 In the money 🟧 Gold ⬜ Silver 🟫 Bronze

#	△pub	Team Name	Kernel	Team Members	Score	Entries	Last
1	▲87	Life in a Glass House			0.94254	3	2y
2	▲4	small yellow duck			0.94167	9	2y
3	▲2	mechatroner			0.94113	29	2y
4	▼2	SY			0.94078	58	2y
5	▲7	square7			0.93992	44	2y
6	▲13	Mario Filho			0.93964	31	2y
7	▲25	Artem.			0.93939	36	2y
8	▲78	ask788			0.93938	41	2y
9	▲104	Rokoson			0.93926	36	2y
10	▲99	rhnil			0.93919	22	2y
11	▲60	YS-L			0.93889	18	2y

3. Methodology

3.1. Feature Engineering

The features extracted from the **bids** dataset are the following:

- Number of bids per user.
- Number of auctions per user.
- Number of countries per user.
- Number of IPs per user.
- Bids to auction ratio: $\frac{Number\ of\ bids\ per\ user}{Number\ of\ auctions\ per\ user}$
- Average response time: $\frac{\sum_{i=0}^{N-1} time(bid_{i+1}) - time(bid_i)}{N-1}$

More details can be found in the **summarize_data.py** script.

3.2. Data Preprocessing

The processing done in the **prepare_train_and_test_sets.py** script consist of the following steps:

- Oversampling the training set using the SMOTE algorithm [8] in order to balance the classes and ease the training process of the classifier. This oversampling technique wasn't needed for the test set and, hence, it wasn't applied.
- Normalized features in both the training and test sets to the range [0, 1], using **scikit-learn** MinMaxScaler.
- 80% of the training data was used to train the classifier, while the remaining 20% was hold-out for validation purposes. For reproducibility, the selected random seed was 42.

3.3. Implementation

The implementation process consisted on spot-checking base instances of the algorithms listed in Section 1.2. Each instance is known as a **candidate**. Each candidate is trained using the default parameters or good initial ones. Our objective in this stage is not to optimize but to select among all candidates the top three performers.

For automatization purposes, each candidate is accompanied by a parameter grid that will only be used if that candidate obtains a spot in the podium.

Among the main challenges that arose during the implementation phase are:

- Highly imbalanced data that makes considerably difficult to learn the nuances and edge cases of the dataset, thus hurting the performance of each candidate. At first, we decided to tackle this situation by implementing oversampling and data augmentation techniques from scratch. However, this task proved to be daunting and very time consuming, for which we finally relied on **imbalanced-learn** implementation of the SMOTE oversampling algorithm.
- Preparing both the training and testing data, due to the information being spread among two datasets (**bidders** and **bids**). It took several investigation-experimentation-testing loops to come up with the final features that were extracted from the **bids** dataset and incorporated to both the training and testing sets. For more information, please refer to **notebooks/data_exploration.ipynb**.
- Choosing the right visualizations posed a great challenge as well due to the widely spread of values that each feature can take. For instance, most of the bidders participate in less than five auctions, but there are a couple of them that are present in more than 1500. This produces an histogram both very tall and wide.

Here is a table that describes the candidates, their initial parameters, the score they achieved and their parameter grid (the best three performers are highlighted in yellow) (more details can be found in **train_candidates.py** script):

CANDIDATE	**INITIAL PARAMETERS**	**GRID PARAMETERS**	**ACHIEVED SCORE**
Decision Tree	Default	**criterion**: ('gini', 'entropy') **splitter**: ('best', 'random') **min_samples_split**: (2, 3, 5, 10) **min_samples_leaf**: (1, 2, 5)	0.941940412529
Random Forest	Default	**n_estimators**: (10, 20, 25, 50, 75, 100) **criterion**: ('gini', 'entropy') **min_samples_split**: (2, 3, 5, 10) **min_samples_leaf**: (1, 2, 5)	0.944373326726
Gradient Boosting	Default	**loss**: ('deviance', 'exponential') **learning_rate**: (0.1, 0.01, 0.005) **n_estimators**: (10, 25, 50, 75, 100) **criterion**: ('friedman_mse', 'mse', 'mae') **min_samples_split'**: (2, 3, 5, 10) **min_samples_leaf**: (1, 2, 5)	0.927174309527

SVC	Default	**kernel**: ('rbf', 'poly', 'linear', 'sigmoid') **degree**: (3, 4, 5) **C**: (1.0, 0.5, 0.001, 1.5)	0.636349871644
Gaussian Naïve Bayes	Default		0.771694918754
Logistic Regression	Default	**C**: (1.0, 0.5, 0.001, 1.5)	0.737290001996
k-Nearest Neighbors	Default	**n_neighbors**: (2, 3, 5, 10, 50) **weights**: ('uniform', 'distance') **algorithm**: ('auto', 'ball_tree', 'kd_tree', 'brute') **p**: (1, 2, 3)	0.888402535461
Perceptron	Default	**tol**: (1e-4, 1e-3) **max_iter**: (1000, 5000) **penalty**: (None, 'l1', 'l2')	0.710073710074
Multilayer Perceptron	**solver**='lbfgs' **hidden_layer_sizes**=(512, 128)	**solver**: ('lbfgs', 'sgd', 'adam') **hidden_layer_sizes**: [(128, 64), (512, 128, 64)] **activation**: ('identity', 'logistic', 'tanh', 'relu') **alpha**: (0.002, 0.005, 0.0001) **learning_rate**: ('constant', 'invscaling', 'adaptive')	0.861203449439
Linear Discriminant Analysis	Default	**solver**: ('svd', 'lsqr', 'eigen')	0.769379004673

3.4. Refinement

The next stage in our implementation process is to fine-tune the three best candidates in order to select a winner. For that purpose we make use of **scikit-learn GridCV** cross-validator to pick the combination that provides the best results based on the relevant parameters specified in the grid that accompanies each podium candidate. Here is a table that describes the candidates, their grid parameters, their score before fine-tuning, their score after fine-tuning and the best parameters found by grid search. The winner is highlighted in yellow (more details can be found in **train_best.py** script):

CANDIDATE	**GRID PARAMETERS**	**BEST PARAMETERS**	**ACHIEVED SCORE BEFORE FINE-TUNING**	**ACHIEVED SCORE AFTER FINE-TUNING**
Decision Tree	**criterion**: ('gini', 'entropy') **splitter**: ('best', 'random') **min_samples_split**: (2, 3, 5, 10) **min_samples_leaf**: (1, 2, 5)	**criterion:** 'entropy' **splitter:** 'best' **min_samples_split:** 2 **min_samples_leaf:** 1	0.941940412529	0.942480677775

Random Forest	**n_estimators**: (10, 20, 25, 50, 75, 100) **criterion**: ('gini', 'entropy') **min_samples_split**: (2, 3, 5, 10) **min_samples_leaf**: (1, 2, 5)	**n_estimators:** 50 **criterion:** 'entropy' **min_samples_split:** 2 **min_samples_leaf:** 1	0.944373326726	0.948599095658
Gradient Boosting	**loss**: ('deviance', 'exponential') **learning_rate**: (0.1, 0.01, 0.005) **n_estimators**: (10, 25, 50, 75, 100) **criterion**: ('friedman_mse', 'mse', 'mae') **min_samples_split'**: (2, 3, 5, 10) **min_samples_leaf**: (1, 2, 5)	**loss:** 'deviance' **learning_rate':** 0.1 **n_estimators':** 100 **criterion:** 'friedman_mse' **min_samples_split:** 2 **min_samples_leaf:** 2	0.927174309527	0.933144756674

4. Results

4.1. Model Evaluation and Validation

In order to validate the performance of each candidate, we selected 20% of the training set as a validation set. For reproducibility purposes, the random seed that was used is 42.

The final model is a Random Forest. We chose this classifier because it achieves the best score in the validation set after fine-tuning, 0.948599095658, versus the 0.933144756674 that Gradient Boosting achieved and the 0.942480677775 scored by a fine-tuned decision tree. The best hyperparameters combination for it were selected using grid search with cross-validation. They are:

- Number of estimators: 50.
- Criterion: Entropy.
- Maximum depth (Maximum depth of each tree).
- Minimum of samples per split: 2
- Minimum number of samples per leaf node: 1

Our Random Forest classifier scored 0.997402918996 in the training set and 0.948599095658 in the validation set. This gap is most likely a sign of overfitting to the training set. This situation can be alleviated by fine-tuning the generated trees in the forest, preventing them to achieve unnecessary depths, instead having a bigger forest of shallower trees (a similar approach to the one that Gradient Boosting takes).

The Random Forest was tested against the prepared testing set (for more details about the preprocessing and preparation steps, refer to Section 3.2. The code is located in the **prepare_train_and_test_sets.py** script).

Best model's predictions were saved in a file called **submission.csv** and uploaded to the competition submission page. Following are some sample predictions:

BIDDER ID	**PREDICTION**
bef56983ba78b2ee064443ae95972877jfkyd	1.0
7ade70030d559a6c255be2f6feca17acnrqs0	0.0
4981c32c54dde65b79dbc48fd9ab6457caqze	0.0
9d4849aaa68d39cda082ce116d62e92dutahi	1.0
6d999f5aea4bbf279e50cba9050500cfxu26e	0.0

The achieved score was 0.70278 in the public leaderboard, which consists of 30% of the test data, and 0.72328 in the private leaderboard which is comprised of the remaining 70% of the test data.

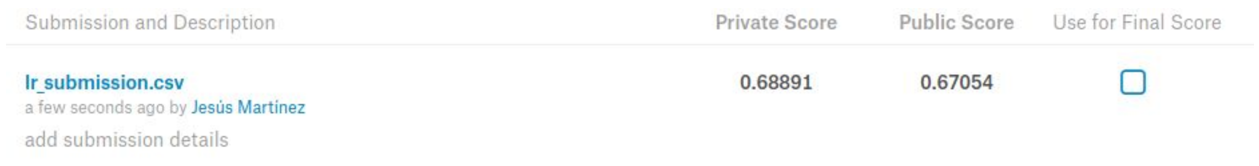
submission.csv
5 days ago by Jesús Martínez
add submission details
0.72328 0.70278 ✓

Compared our model's score (0.72328) with the benchmark score of 0.942543 we established in previous section, our model is still far from the highest ranks of the final standings in the private leaderboard. This might be explained by several of the following reasons:

- Our model is not robust enough to the fluctuations, nuances and corner cases in the data.
- Our engineered features lack sufficient explanatory power.
- Our model has overfit to the training data, thus it fails to generalize on unseen data.
- We engineered insufficient features.

- The selected algorithm might not be the best option for the problem at hand. Exploring other classifiers such as Gradient Boosting or Neural Networks could yield better results.

Although our Random Forest classifier still needs tuning to compete with the highest ranking models, when compared with a simpler algorithm, such as Logistic Regression, it performs better in both private and public leaderboards. Logistic Regression achieves 0.67054 in the public leaderboard, versus the 0.70278 our Random Forest scores, while Logistic Regression has an AUC of 0.68891 in the private leaderboard against the 0.72328 of our best model.



Submission and Description	Private Score	Public Score	Use for Final Score
lr_submission.csv a few seconds ago by Jesús Martínez add submission details	0.68891	0.67054	☐

5. Conclusion

5.1. Free-Form Visualization

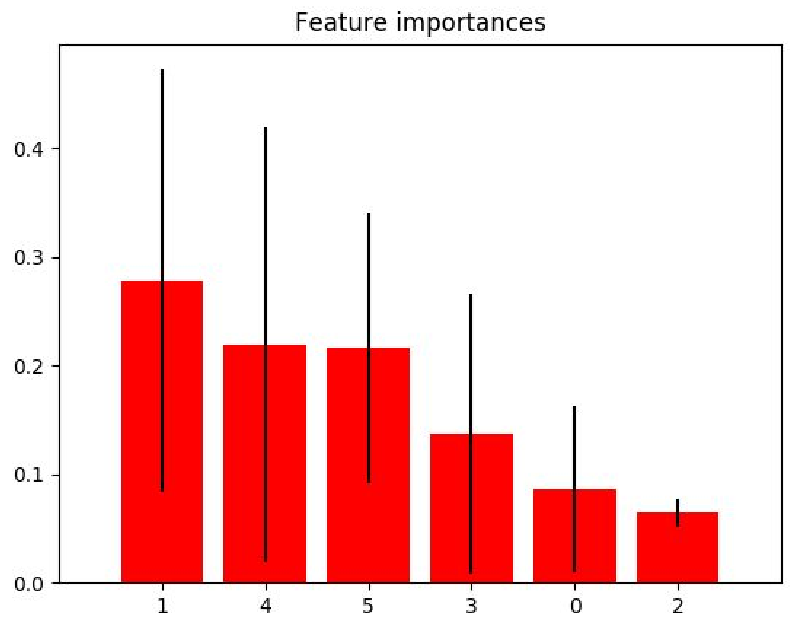
Following is a plot that shows the importance of each feature in the classification task of discerning if a user is a robot or a human. The mapping between feature number and names are:

- Feature 0: Number of auctions per bidder.
- Feature 1: Number of bids per bidder.
- Feature 2: Number of countries per bidder.
- Feature 3: Number of IP addresses per bidder.
- Feature 4: Bids to auction ratio.
- Feature 5: Average response time between two consecutive bids.

Here are the ranking of the features in terms of importance for the Random Forest:

1. Feature 1 (0.277743).
2. Feature 4 (0.218954).
3. Feature 5 (0.215998).
4. Feature 3 (0.136870).

5. Feature 0 (0.085904).
6. Feature 2 (0.064533)



The plot and the ranking suggest that the number of bids per bidder (feature 1), the bids to auction ratio (feature 4) and the average response time between two consecutive bids (feature 5) are the most informative features and, hence, drive the behavior of the final model. The number of IP addresses per bidder (feature 3) is less important than the first three features, but still contributes to the output. Finally, the number of auctions per bidder (feature 0) and the number of countries per bidder (feature 2) are not very informative, being their predictive power negligible and could perfectably be dropped with a very low risk of hurting the model's performance.

5.2. Reflection

I found this project very challenging because it wasn't only focused on the algorithmic side of machine learning (comparing different models, selecting best hyperparameters, splitting data for training and testing, etcetera), but on the data science aspect as well.

Due to the obfuscation of several fields and the lack of important information such as time units (what does the **time** field represent for a bid? Days? Months? Seconds?), monetary units (what is the amount of each **bid**?) and knowledge of the internal behavior of the auction site (among other), interpretation plays a major role in filling the gaps. Moreover, feature engineering is an indispensable step before the training phase, because the data is spread among two datasets: **bidders** and **bids**.

Another characteristic of the data that increases the difficulty is that is highly imbalanced, where only 5.12% of the bidders in the training set has been flagged as bots. For that reason, one of the most popular classification metrics, accuracy, loses explanatory power, because it only focuses on the correct predictions. Unfortunately, in this case a model that always predict 0.0 would achieve a decent 94.88% of accuracy, but it wouldn't be helpful solving the task at hand. Hence, we use AUROC as our score function because it performs really well with imbalanced datasets. Please, refer to Section 1.5 for a better explanation of Area Under Receiver Operating Characteristic Curve.

Imbalance also posed a challenge at training time, with our candidate models being unable to learn as a consequence of insufficient positive examples. In order to combat this situation [9] we relied on **imbalanced-learn** implementation of the Synthetic Minority Over-sampling Technique algorithm, which increased the performance of the majority of candidate models in 30% to 40%, approximately.

Given the great gap between the score achieved by the model in the validation set and its performance in the final test set, it seems evident that we overfit to both the training and validation set. It is very likely that we engineered insufficient features, and that their explanatory power isn't enough to capture all nuances and corner cases of the data.

Overall, this was a very fun and challenging project that forced me to think out of the box, consult external resources and further the boundaries of my comfort zone, thus expanding my knowledge and providing me with new tools and techniques to tackle future and more complex problems in the realm of machine learning and data science in general.

5.3. Improvements

There are many improvements that can be done to achieve much better results. In particular, Kaggle competitor **small yellow duck** identified the following features as the most important for the Random Forest Classifier he trained and that achieved 2nd place in the final leaderboard:

- The median time between a user's bid and that user's previous bid.
- The mean number of bids a user made per auction.
- The entropy for how many bids a user placed on each day of the week.
- The means of the per-auction URL entropy and IP entropy for each user.
- The maximum number of bids in a 20 min span.
- The total number of bids placed by the user.
- The average number of bids a user placed per referring URL.
- The number of bids placed by the user on each of the three weekdays in the data.
- The minimum and median times between a user's bid and the previous bid by another user in the same auction.
- The fraction of IPs used by a bidder which were also used by another user which was a bot.

References

[1] https://dl.acm.org/citation.cfm?id=2843948

[2] https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

[3] https://medwhat.com/

[4] https://en.wikipedia.org/wiki/Denial-of-service_attack

[5] https://dl.acm.org/citation.cfm?id=1323139

[6] https://en.wikipedia.org/wiki/Receiver_operating_characteristic

[7] https://www.youtube.com/watch?v=OAl6eAyP-yo

[8] https://www.jair.org/media/953/live-953-2037-jair.pdf

[9]https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset/