How to decide as soon as possible: application to web bot recognition

Aziza Zhanabatyrova
University of Genoa
Genoa, Italy
aziza91@inbox.ru

Clayton Leite
University of Genoa
Genoa, Italy
clayton.fk06@gmail.com

***Abstract*—In this work, we study a problem of detecting malicious ad fraud bots as soon as possible. First, we look at the motivations behind this project. Then the problem is approached using supervised learning with neural networks to distinguish between the behaviour of bots and humans; also the choice of this approach is explained. Initially, the network is trained using K-fold cross validation technique, to find optimal parameters, with which the network is later trained with the complete dataset. The performance is improved with sequential classification, which provides more confident results. Finally, we show the results in favour of the techniques used. At last, some final notes are discussed about the nature of the problem, the solution and its limitations.**

***Index Terms*—Ad fraud bots, machine learning, neural networks, multi-layer perceptron, k-fold cross validation, sequential classification**

I. Introduction

Deciding online, in the shortest possible time, is a problem relevant in many fields including computer science and robotics. For example, in speech recognition applications, when recognising a given spoken word, the software might not be able to make an accurate decision based on the first few letters, but only after collecting a sufficient amount of streaming information. One more example of this type would be the motion planning of a robot based on its goal location and the trajectory trend of moving obstacles in the environment. This paper addresses another particular instance of the same problem, web bot recognition.

Bots are software intended to perform automated tasks. For example, bots can perform searches for a certain product in a significant number of shopping websites and display the results ordered by price to the user, fetch weather information online and inform about chances of raining in the location of the user or simply have conversations with humans using their natural language.

Bots are becoming increasingly common online and, as a matter of fact, according to the results of bot tracking activity addressed on the 2016 Bot Traffic Report [3] issued by the web security firm Imperva, they generate more than a half of the internet traffic (Fig. 1).

The types of bots mentioned above are commonly termed as good bots; however, the study also reveals that the majority of bots seen online have malicious purposes, usually to carry out DDoS (Distributed Denial of Service) attacks, but not only.

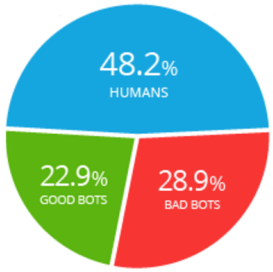

Fig. 1. Online bot activity. Source: Imperva Incapsula. Bot Traffic Report 2016. *https://www.incapsula.com/blog/bot-traffic-report-2016.html*. Accessed on: July 28th 2017.

As mentioned in [4], bots intended to abuse advertisement systems are also among the most common malicious ones. Often advertisers pay for including ads in online content using a click-based revenue model, that is, publishers earn money from ad networks proportionally to the number of clicks users performed on the advertisements they expose. Also, the advertisers set a daily limit of money they will pay for the publisher, and when this limit is exceeded, the publisher will stop exposing ads of the advertiser for that day.

A bot issuing numerous HTTP requests for ads URLs in a short time could just mimic user clicks and exploit this revenue model by inflating the earnings of the publisher while causing loss to the advertisers. This kind of exploitation is referred as *publisher click inflation* [4]. Another abuse of advertisements is the *advertiser competitor clicking attack* [4] and happens when an advertiser creates bots to fraud clicks on ads of its competitors. The competitors will be charged for the fraudulent clicks and then their daily budget for ads will rapidly exceed causing their ads not to be exposed by publishers for the day.

Therefore, it is of great importance to detect ad fraud bots and stop them before their actions result in losses for the advertisers attacked. This work will study a way to detect bots based on their behaviour.

II. Description of the problem

In internet terms, a session is a period during which two computers - usually a server and a client - communicate with each other. An HTTP session is a common type of client/server session and consists of a sequence of HTTP requests issued by

the client. Normal access requests patterns, i.e., those requests generated by human users, are substantially different from bots access patterns [6] and this distinction can be used to detect bots.

To facilitate the detection of bots, the HTTP requests can be encoded in a different way such that they contain only relevant information for the purposes of distinguishing between human and bot access patterns. Then these preprocessed HTTP requests can be sent as input to a system that will output whether or not the user is a bot. Naturally, one request is not sufficient to reveal decisive information. Thus, such a bot detection system should be capable of accumulating requests up to the moment when it judges it has enough information to determine if the requests were generated by a bot.

This bot detection structure can be further explained as follows:

1) An HTTP request is received from the client, and it is preprocessed into a relevant way for the bot detection system.
2) The bot detection system receives the preprocessed request and, based on it and other previously preprocessed requests received, it decides between three possible choices: the user is a bot, the user is not a bot, or the information is insufficient to conclude.
3) If the bot detection system determines that the user is a bot or a human, then the request sampling is interrupted for the session. And if it determines that it does not have enough information to perform the decision, then it waits for another preprocessed HTTP request. When the client sends another HTTP request, the loop restarts.

In this work, we are provided with 2,267,504 HTTP requests (previously preprocessed [1]) obtained from 13,587 sessions, and for every request, there is also the information on whether or not it was generated by a bot. Our goal is to create a bot detection system as defined above.

III. Methodology

A. Machine learning

As mentioned by Jarvis [5], the processes of learning constitute a fundamental stimulus of life and are essential for the humanity. Hence, if we could program machines to exhibit our same behaviour of learning, the impact would be dramatic [7]. Machine learning, i.e. the study of implementing the ability to learn in machines, has been of keen interest and a common area of research in science for the past decades.

We approach the problem of detecting ad fraud bots with artificial neural networks, which is one of the primary methods in machine learning. Our choice is justified by the fact that neural networks are very powerful with their ability to derive meaning from complicated or imprecise data, model extremely complex functions [12] and learn complex conditional probability distributions in multidimensional spaces, as it is our case.

[1]One of the features included in the preprocessed data is the time interval from the last request of the same session. As total, every preprocessed request has 25 features.

B. Artificial neural networks

Artificial neural networks (ANNs) were inspired by the architecture of a human brain, which consists of nerve cells or neurons that communicate via electrical signals [2].

In ANN, each input to a neuron is multiplied by some weight, and weighted signals are summed in the activation function to activate the node (neuron). If the output of the activation function exceeds a defined threshold, we predict a high valued output, for example, 1, otherwise 0 [8] (Fig. 2).

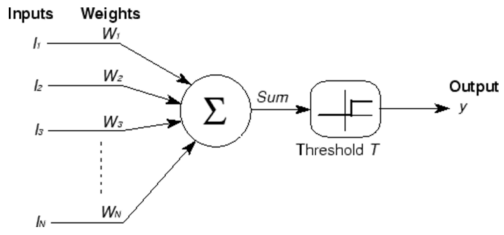

Fig. 2. Model of neuron. Source: http://slideplayer.com/slide/9376233/. Accesed on: 31st July 2017.

In Multi-Layer Perceptron (Feedforward ANN), the nodes are arranged in a layered structure, in which inputs are passed through each node in the first nonlinear layer, called hidden layer, producing some outputs which serve as inputs to every node in the next layer and so on until we reach the final output layer, without loops (Fig. 3).

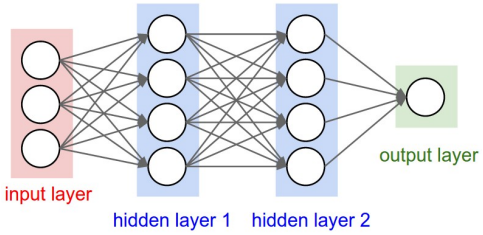

Fig. 3. Neural network with two hidden layers. Source: http://cs231n.github.io/neural-networks-1/. Accessed on: 31st July 2017.

These outputs are then compared to the target outputs, and until the error between them is minimized, we iteratively keep updating weights, which were initialized randomly [2]. This method is called supervised learning, where the main goal is to learn and optimize a model from labelled training data that allows us to make predictions about unseen data [8].

Each classification method has its pros and cons and performance depends heavily on the data. There is no single classifier that works best in all cases. Therefore the best practice would be to compare the performance of several algorithms on a particular dataset [8].

We use the MLP algorithm from the scikit-learn library in our code, which offers a user-friendly interface for efficiently using a large variety of machine learning algorithms as well as functions for data preprocessing, fine-tuning and evaluating models, whose implementation was highly optimized [8]. The neural network is trained using preprocessed data, where

preprocessing means shaping raw data in the form that is necessary for the optimal performance of a learning algorithm [8].

C. K-fold cross validation

Cross-validation techniques are used to avoid overfitting and to provide the model with a good generalization, thus finding a balance between the bias and variance dilemma [10]. They can also be used for the comparison between multiple models to find optimal model parameters [9], i.e., to assess and fine-tune the models' performances on unseen data.

K-fold is one of the cross-validation techniques. It consists of partitioning the dataset into k different folds, where (k−1) folds are used for the training, and the remaining one is used for testing [8]. This procedure is repeated k times so that we obtain k models and k performance estimates.

The next step is calculating the weighted average of the performance estimates. The K-fold cross-validation can be repeated for different parameters of the neural network, and once we have found satisfactory parameter values, we can retrain the neural network using those parameters on the complete training set; finally, we use the same dataset as the test set [8]. The procedure is depicted in Fig. 4.

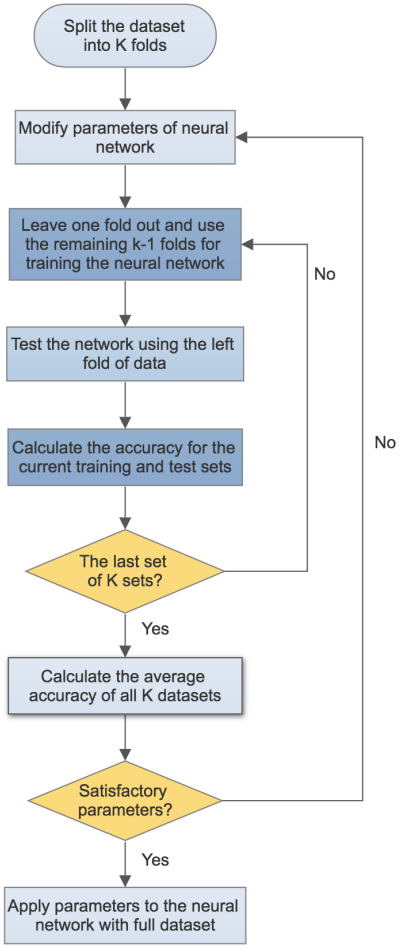

Fig. 4. K-fold cross validation method.

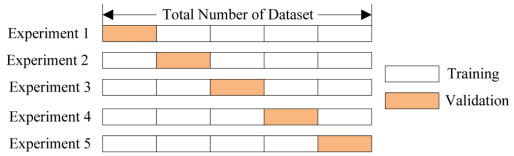

Fig. 5. K-fold cross validation for k = 5. Source: http://sdsawtelle.github.io/blog/output/week6-andrew-ng-machine-learning-with-python.html. Accessed on: 31st July 2017.

The reason for which we use K-Fold cross validation instead of the conventional validation method - where the dataset is simply split into two partitions: training set and test set - lies on the fact that the conventional validation leaves us with a high probability that difficult cases are not going to be contained in one of the two sets, which, as a consequence, result in a loss of modelling and testing capability.

Training neural networks with a large dataset is a hefty task from the computational viewpoint, however, since all the k trainings in the k-fold cross validation technique are independent on the rest, we opted to decrease computational time by writing a multi-core code, which resulted in a decrease of the order of 30% in time with comparison to single-core code. Also, in the tests, the parameter k was chosen to be 10.

D. Sequential classification

As mentioned earlier, one request is not sufficient to permit a distinction between bot or human. We will use the sequential probability ratio test described in [11] and [1] to deal with a sequence of requests.

In our case, the sequential analysis is a statistical hypothesis testing procedure [1] which consists of taking successive requests (sampling) and calculating, after each observation is taken, a likelihood ratio defined as:

$$R = \sum_{i=1}^{n} \log(P_1(y_i)) - \sum_{i=1}^{n} \log(P_0(y_i)), \qquad (1)$$

where $P_1(y_i)$ denotes the probability of request y_i have been generated by a bot and, similarly, n is the number of requests, $P_0(y_i)$ the probability of request y_i have been generated by a human. $P_1(y_i)$ is estimated as the output of the neural network for y_i, and $P_0(y_i) = 1 - P_1(y_i)$.

It is important to mention that we are assuming that the probabilities don't affect each other, i.e., the occurrence of a request y_i and a request $y_j \neq y_i$ have been generated by a bot are independent. This is probably not the case; however, this assumption does not affect the bot recognition system in a negative way, since it predicts a worst case scenario.

The likelihood ratio represents a measure of certainty. Positive (negative) values of R gauge how certain we can be about the hypothesis of a bot (a human) have generated the requests upon which it was calculated.

Then we can define two thresholds T_0 and T_1 such that:

- If $R > T_1$, we stop sampling and accept the hypothesis that a bot is sending the requests.

- If $R < T_0$, we stop sampling and reject the hypothesis that a bot is sending the requests.
- If $T_0 < R < T_1$, we continue the sampling without either accepting or rejecting the hypothesis.

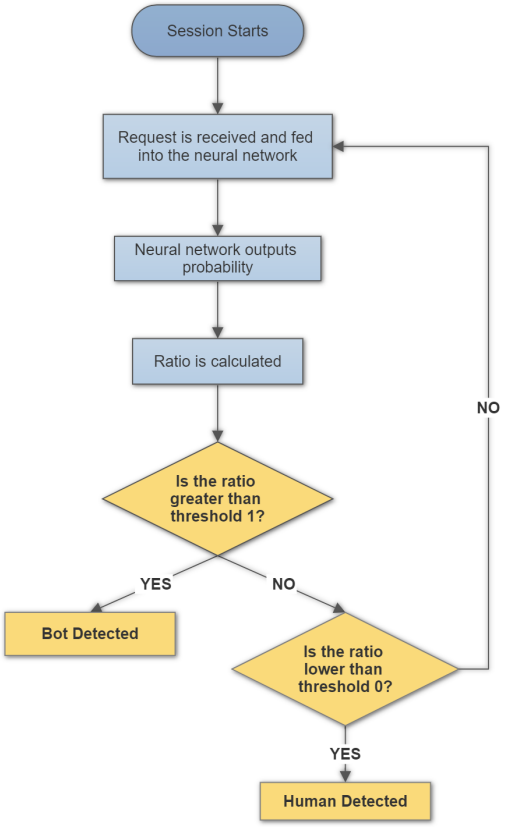

Fig. 6. Sequential classification process.

Figure 6 depicts the process of sequential classification.

Naturally, if T_1 is chosen to be a substantial number, then we might need too many requests to have $R > T_1$ and decide that the requests came from a bot. This poses a problem because if we have a bot sending requests, we will want to detect it as soon as possible. Also choosing T_1 to be a small number poses a problem because we might commit many false positives, i.e., concluding it is a bot when it is not.

Alternatively, if T_0 is chosen to be very small, we might spend too much time before concluding that a human is connected to the session. And a larger value of T_0 might result in many false positives, i.e., concluding it is a human when a bot possesses the session.

We shall choose the values of the thresholds that guarantee a balance between errors (false positives and false negatives) and the time taken to accept or reject the hypothesis.

IV. Results

A. K-fold Cross Validation

Among the neural network models tested and shown in table I, we chose the one with two hidden layers containing ten neurons each, since it leads to a good accuracy at the same time as keeping a reasonable light computational effort.

The models with increased number of neurons, however, were discarded, because they provide only a negligible increase in accuracy - possibly, of the order of the uncertainty associated with the measurement - which is not sufficient to explain its use since the computational effort associated with them is severely high.

Configuration	Average accuracy
(5,5)	97.85%
(10,5)	98.02%
(10,10)	98.08%
(20,20)	98.13%
(30,30)	98.15%
(50,50)	98.16%

TABLE I
RESULTS FOR K-FOLD CROSS VALIDATION (K = 10).

Also, another reason to have a slightly lower accuracy and significantly less computation cost is that we might be interested in always retrain the neural network when new data is available.

B. Sequential Classification

I $T_1 = 15, T_0 = -15$	
False positives	38
False negatives	60
Correct bot detection rate	98.47%
Correct human detection rate	99.21%
Undecided sessions rate	25.75%
Average number of requests to decide	7.96
Overall correct decision rate	99.02%
II $T_1 = 12, T_0 = -12$	
False positives	25
False negatives	93
Correct bot detection rate	98.80%
Correct human detection rate	98.78%
Undecided sessions rate	28.12%
Average number of requests to decide	6.93
Overall correct decision rate	98.79%
III $T_1 = 10, T_0 = -10$	
False positives	55
False negatives	88
Correct bot detection rate	98.15%
Correct human detection rate	98.85%
Undecided sessions rate	21.41%
Average number of requests to decide	5.93
Overall correct decision rate	98.66%
IV $T_1 = 7, T_0 = -7$	
False positives	98
False negatives	129
Correct bot detection rate	96.72%
Correct human detection rate	98.34%
Undecided sessions rate	20.64%
Average number of requests to decide	5.08
Overall correct decision rate	97.89%

TABLE II
RESULTS FOR SEQUENTIAL CLASSIFICATION WITH CONSERVATIVE THRESHOLDS. T_1 (T_0) IS THE THRESHOLD ABOVE (BELOW) WHICH WE CLASSIFY THE USER AS A BOT (HUMAN).

It was found that restricting the sequential test to not decide immediately after the first request always produced a better correct decision rate. This is because one request alone doesn't contain conclusive information about the nature of the user, i.e., whether it is a bot or not.

The sequential classification used the same neural network obtained as best result in the K-Fold cross validation and the results are shown in tables II and III, where we present the best eight different configurations of thresholds and their respective amount of false positives (type I errors), false negatives (type II errors), undecided sessions and correct decision rates.

The table II shows models with conservative thresholds, i.e., models where the decision is taken after obtaining a relatively high level of certainty, and the table III displays models that can be said to be a relaxed since decisions can be made with a lower level of certainty.

Being more conservative increases the accuracy of the sequential classification; however, the decision requires more time to be made and, in this case, many sessions are left without any decision. Also, the computational effort on the web server can be negatively affected by such delay in deciding. On the other side, a relaxed model provides decisions faster at the cost of a lower accuracy. Figure 7 illustrates the trade-off between correct decision rate (accuracy) and average number of requests need to decide.

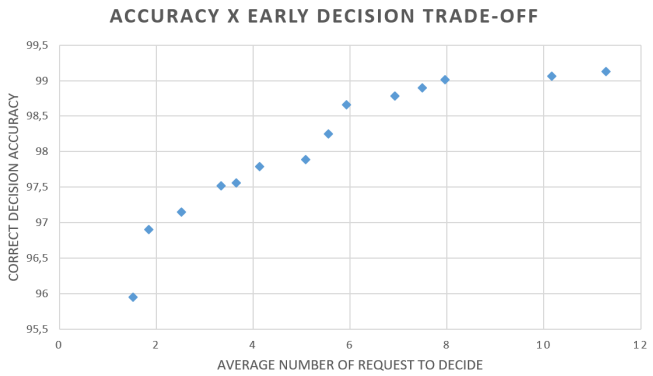

Fig. 7. Relation between accuracy (in percentage) and average number of requests to decide.

To solve this dilemma, we can interpret the bot detection system in a slightly different way. Instead of taking for granted that a bot is sending requests when the bot detection system outputs a positive response, we can consider it as just an indication of bot activity to take further procedures, for example, the indication can be used to display a CAPTCHA. Therefore, with such interpretation, we can adopt a relaxed model, because false positives will not a problem, since the human will be able to solve the CAPTCHA and continue the session normally.

V $T_1 = 5, T_0 = -5$	
False positives	166
False negatives	128
Correct bot detection rate	96.01%
Correct human detection rate	98.37%
Undecided sessions rate	18.13%
Average number of requests to decide	3.65
Overall correct decision rate	98.12%
VI $T_1 = 3, T_0 = -3$	
False positives	181
False negatives	177
Correct bot detection rate	96.07%
Correct human detection rate	97.78%
Undecided sessions rate	7.36%
Average number of requests to decide	2.52
Overall correct decision rate	97.12%
VII $T_1 = 2, T_0 = -2$	
False positives	188
False negatives	218
Correct bot detection rate	96.22%
Correct human detection rate	97.31%
Undecided sessions rate	3.60%
Average number of requests to decide	1.84
Overall correct decision rate	96.90%
VIII $T_1 = 1, T_0 = -1$	
False positives	238
False negatives	309
Correct bot detection rate	95.50%
Correct human detection rate	96.24%
Undecided sessions rate	0.50%
Average number of requests to decide	1.52
Overall correct decision rate	95.95%

TABLE III
RESULTS FOR SEQUENTIAL CLASSIFICATION WITH RELAXED THRESHOLDS. T_1 (T_0) IS THE THRESHOLD ABOVE (BELOW) WHICH WE CLASSIFY THE USER AS A BOT (HUMAN).

V. CONCLUSION

It has been seen that the bot detection system developed is not completely reliable. False positives are unavoidable, in other words, the neural network combined with the sequential classification method is not sufficient to classify request as being generated by bots or humans with perfect precision. However, the use of a CAPTCHA, as discussed, could be a decent solution for this.

Also, there is no guarantee that the neural network will be able to detect a type of bot that it has never been trained to detect. This leads to a necessity of retraining it whenever a new batch of data is available, therefore, the bot detection system has to be implemented in such a way that permits periodical updates with fast and affordable means.

Nevertheless, the bot detection system is a valid solution for the bot recognition problem since it can deliver very solid results with relatively high accuracy for the problem studied in this paper.

REFERENCES

[1] Ewens, W. J., Grant, G. R. Statistical Methods in Bioinformatics: An Introduction. Springer Science & Business Media, 2006.

[2] Gurney, K. An Introduction to Neural Networks. CRC Press, 1997.

[3] Imperva Incapsula. Bot Traffic Report, 2016. *https://www.incapsula.com/blog/bot-traffic-report-2016.html*. Accessed on: July 28th 2017.

[4] Jakobsson, M. The Death of the Internet. Wiley, 2012.

[5] Jarvis, P. Towards a Comprehensive Theory of Human Learning. Routledge, 2006.

[6] Jie, L., Jianwei, S., Changzhen, H. A Novel Framework for Active Detection of HTTP Based Attacks. In: Ma M. (eds) Communication Systems and Information Technology. Lecture Notes in Electrical Engineering, vol 100. Springer, Berlin, Heidelberg, 2011.

[7] Mitchell, T. M. Machine Learning. McGraw Hill, 1997.

[8] Raschka, S. Python Machine Learning. Packt Publishing, 2016.

[9] Reed, R. D. and Marks, R. J., Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks. MIT Press, 1998.

[10] Reitermanová, Z. Data Splitting. WDS'10 Proceedings of Contributed Papers, Part I, 31–36, 2010.

[11] Wald, A. Sequential Analysis. Dover Publications, 2013.

[12] Yin, Y., Kaku, I., Tang, J., Zhu, J. Data Mining: Concepts, Methods and Applications in Management and Engineering design. Springer Science & Business Media, 2011.