Knowledge-Based Systems 223 (2021) 107074

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Efficient on-the-fly Web bot detection

Grażyna Suchacka [a,*], Alberto Cabri [b,c,1], Stefano Rovetta [b,c,1], Francesco Masulli [b,c,1]

[a] Institute of Informatics, University of Opole, Opole, Poland
[b] Department of Informatics, Bioengineering, Robotics, and Systems Engineering (DIBRIS), University of Genoa, Genoa, Italy
[c] Vega Research Laboratories S.r.l., Genoa, Italy

ARTICLE INFO

Article history:
Received 7 January 2021
Received in revised form 6 April 2021
Accepted 20 April 2021
Available online 22 April 2021

Keywords:
Web bot
Internet robot
Real-time bot detection
Machine learning
Sequential analysis
Neural network
Early decision

ABSTRACT

A large fraction of traffic on present-day Web servers is generated by bots — intelligent agents able to traverse the Web and execute various advanced tasks. Since bots' activity may raise concerns about server security and performance, many studies have investigated traffic features discriminating bots from human visitors and developed methods for automated traffic classification. Very few previous works, however, aim at identifying bots on-the-fly, trying to classify active sessions as early as possible. This paper proposes a novel method for binary classification of streams of Web server requests in order to label each active session as "bot" or "human". A machine learning approach has been developed to discover traffic patterns from historical usage data. The model, built on a neural network, is used to classify each incoming HTTP request and a sequential probabilistic analysis approach is then applied to capture relationships between subsequent HTTP requests in an ongoing session to assess the likelihood of the session being generated by a bot or a human, as soon as possible. A performance evaluation study with real server traffic data confirmed the effectiveness of the proposed classifier in discriminating bots from humans at early stages of their visits, leaving very few of them undecided, with very low number of false positives.

1. Introduction

In the present era of huge proliferation of Internet and mobile technologies, more and more of our everyday life activities have been shifting to virtual platforms. Social life, communication, entertainment, shopping, or search for any kind of information are only a few examples of this trend. At the same time Web analytics and online marketing tools have been increasingly used to gain competitive advantage in this newly shaped market. The rapid development of these technologies caused the rise of Web bot traffic, thanks to which advanced Web-based applications may provide users with up-to-date, accurate, and customized services.

A Web bot, also called Internet robot, Web agent, or intelligent agent, is a software tool which carries out specific tasks on the Web, usually autonomously, following the structure of hyper-links according to a specific algorithm [1]. Many bots are benign and useful, like search engine crawlers, shopping bots for collecting information on behalf of product search engines or price comparers, link checkers to help website administrators in detecting broken and backlisted links, or feed fetchers ferrying website content to mobile applications. However, activities of some bots raise concerns about ethics and users' privacy, such as in the case of e-mail harvesters, spambots, or content-stealing bots. Furthermore, some robots are undoubtedly harmful: hacking bots used to steal sensitive data or to inject malware, bots for generating click frauds in pay-per-click advertising, or malware used for DDoS (Distributed Denial of Service) attacks are only a few examples.

A significant part of the overall Web traffic is generated by bots, many of which have clearly malicious goals [2]. Bad bots tend to obfuscate their true identities by taking on user agent strings typical of legitimate Web browsers and ignore the file `robots.txt` that contains website access rules for bots [3–5]. This makes it difficult to identify bot traffic on Web servers. In practice, relatively simple bot detection techniques are enforced, like comparison of an IP address or a user agent string with a blacklist of known bots' data or investigating some keywords indicative of a bot in the user agent string. A request stream may be also tested for some atypical statistical characteristics, like extremely short inter-arrival times but these tests are often ineffective because bots tend to mimic human behavior to conceal themselves.

* Corresponding author.
E-mail addresses: gsuchacka@uni.opole.pl (G. Suchacka), alberto.cabri@dibris.unige.it (A. Cabri), stefano.rovetta@unige.it (S. Rovetta), francesco.masulli@unige.it (F. Masulli).

1 Alberto Cabri, Stefano Rovetta and Francesco Masulli are members of the National Group for Scientific Computing (GNCS) of INdAM, the Italian Istituto Nazionale di Alta Matematica "Francesco Severi".

https://doi.org/10.1016/j.knosys.2021.107074

The benefits of the aforementioned techniques are limited to recognizing only well-known bots or to identifying aggressive attacking behaviors, like DDoS attacks. To overcome this problem, a CAPTCHA authentication test is commonly applied. However, as a kind of a challenge–response test, its big disadvantage is bothering the user with additional tasks, e.g., typing letters of distorted images or solving a puzzle. This is burdensome for users and negatively affects their Web experience, so despite its efficacy in telling bots and humans apart, CAPTCHA has received criticism and exposed the need to develop bot detection methods transparent to Web users.

A large body of research has been dedicated to the problem of robot traffic analysis, characterization, and classification. However, robot detection has been mostly addressed in the offline scenario, working on historical session data. To the best of our knowledge, very few studies have dealt with the problem of identifying bots on-the-fly, while subsequent HTTP requests in active sessions come to the server.

The ability to identify Web robots in real time is of crucial importance for the security and performance of a website since it makes it possible to mitigate threats before the end of robot visits and, thus, to limit the impact of their presence. This demands effective methods for early Web bot detection in real time, based on HTTP request features and relationships between requests in ongoing sessions.

The major contribution of this paper is a novel machine learning approach to identify ongoing robot visits on a website.

- The proposed method is particularly suitable to work in real time. It can make a decision as soon as sufficient information is gathered from the incoming requests of a given session; experiments show that this usually occurs in the first few requests, and in a number of cases in as few as two requests.
- The method relies on HTTP-level features which can be easily extracted or computed from HTTP headers. Since this information is easy to acquire in real-time for each current observation, the method can be transparently implemented on a real Web server software, without any need to modify Web-based applications, like e-commerce software.
- The problem of early bot detection is formulated as a binary classification task of sequentially sampled multivariate data and solved by applying a Multi-Layer Perceptron (MLP) classifier combined with a Wald's Sequential Probability Ratio Test (SPRT) module. A multi-objective classification approach allows the method to find the desired trade-off between classification confidence and early decision.

Preliminary results of our method (here named NNSEQ), presented at the WIRN'17 [6] and EDMA'18 [7] conferences, showed its great potential in early Web bot detection. In the present study we applied our method, as well as a recently published approach for real-time bot detection [8], to historical Web traffic data from a real e-commerce site. The latter has been considered as a reference because it addresses the same problem and, to the best of our knowledge, is the only HTTP-level approach for recognizing bots in real-time published so far. For each approach a multi-objective optimization was performed to select the best hyper-parameter values for a comparative analysis. An in-depth performance study for our method was performed and discussed in comparison with the corresponding results of the reference method.

The proposed approach is more efficient than the reference one in terms of recall, precision, accuracy, and percentage of classified sessions. Moreover, it is especially powerful in classification of bots or humans given a very limited number of observations.

Critical challenges for on-the-fly bot detection include solutions with high real-time performance and the need for their adaptability in the face of inherent uncertainty in Web session labeling due to the presence of unknown bots [9,10]. Our study takes a step forward towards dealing with these challenges.

The rest of the paper is organized as follows. Section 2 presents the preliminaries and formulates the problem. Section 3 describes the related approaches to bot recognition on Web servers. The proposed method is presented in Section 4, the reference method in Section 5, while the experimental evaluation details are clarified in Section 6. Section 7 discusses performance results of the method in comparison with the reference approach, followed by conclusions in Section 8.

2. Preliminaries and problem statement

The environment under consideration is a Web server hosting an e-commerce website, hence all the traffic is based on the HTTP protocol, defined at the application layer of the ISO-OSI stack [11,12]. Interaction through the HTTP protocol occurs between Web clients and Web servers. A single client–server transaction involves a *request*, issued by the client, and a *response* sent by the server. The request indicates one of a number of request methods, the most common being GET, POST and HEAD; a header, containing meta-information including a *user agent* string to identify the client; an URI (Unified Resource Identifier); and a body. The response consists of a status line describing the result of the transaction by means of a status code and a verbal description ("reason"); a header; and a body containing the requested data.

Web clients are typically Internet browsers or mobile apps operated by human users, but may be also automated agents. In the case of a browser, the interaction with the server takes place by downloading consecutive Web pages, typically linked to each other. For each new page, the browser generates a sequence of HTTP requests, firstly for the page description file and then for embedded objects, like images. In the case of intelligent agents, however, a sequence of requests is not limited by the website structure, as bots tend to traverse the site according to a given strategy, e.g., depth-first or breadth-first, and may request only selected types of server resources.

HTTP is a *stateless* protocol that does not define any server–client permanent connection. When a session has to be maintained, such as during a customer visit in a Web store, other mechanisms are used, e.g. cookies [13]. As opposed to HTTP transaction data, however, such additional information is not necessarily standardized across Web servers, nor easy to obtain from website administrators, especially for e-commerce sites. Therefore, given only a stream of HTTP requests observed at the server, a common practice is to define a session as a sequence of requests with the same IP address, the same user agent string, and where the time interval between any two subsequent requests does not exceed 30 min [8,14–17].

The problem of real-time session classification on a Web server can be stated as an instance of early classification of multivariate data streams, i.e., classification in the shortest possible time. Since a session corresponds to a single visit of a given client, it is reasonable to assume that, within a session, the website is accessed in a consistent style, hence the corresponding data stream can be thought of as generated by a stationary source. However, temporal dependencies are difficult to model.

Therefore, the problem of on-the-fly Web bot detection may be stated as the task of identifying whether a sequence of HTTP requests for a given session can be labeled as performed by a bot or a human as early as possible before the end of the sequence.

The main assumption underlying the presented approach is its compatibility with the real-time operating conditions of actual Web servers and, consequently, a possibility to integrate it with a Web server software in the future. This could be done throughout a proxy server intercepting and analyzing incoming HTTP requests and eventually enforcing decisions on further handling by the server.

Our goal was to design a method as universal as possible, which does not require the analysis of website semantics, modifications of a website structure, or a special process of data collection on the server or client side.

3. Related work

In recent years, advances in bot technologies to run cons in popular Web-based services motivated the quest for specialized bot detection methods, targeted at specific applications.

The problem affects most websites and has direct economic implications for an online business. Many companies rely on commercial cyber security services, or have studied their own countermeasures. The scholarly literature, however, can only account for those methods that have been publicly disclosed; in contrast, many of these are business secrets or use proprietary data, and do not contribute to the state of the art since they do not allow replication, experimentation, third party validation, or further development. In this work we will therefore restrict ourselves to known methods described in scholarly sources.

Some studies investigate attacking bot behaviors on the network level [18–21]. On the other hand, many techniques rely on investigating the statistical differences in behavioral patterns of bots and humans with regard to application dependent features. For instance, [22] aims at detecting apps subject to search rank frauds in Google Play by analyzing a set of relational, behavioral, and linguistic features. [23] proposes detection of click frauds in pay-per-click advertising by identifying duplicate clicks. Features derived from the analysis of user profiles and product reviews are used to detect spammer groups [24] and shilling attacks [25,26] in online recommender systems. A large number of dedicated bot detection approaches use supervised classification techniques to detect spambots [27], blog bots [28,29], shopping bots [30] and click frauds [31]. Machine learning techniques proved to be also very effective at identifying frauds in online social networks [32–34].

These approaches, developed for specific Web services, are based on selected high-level, application-specific features so their application is limited only to some kind of websites. There are also approaches based on semantic features, which require the prior analysis of the website semantics [35]. Other recent bot detection methods need initial website's pre-processing or modification, like generating a special graph-based sitemap [36] or adding special markers to website URLs [37]. Methods designed specifically to recognize camouflaged robots require adding honeypots or trap files to the site, invisible for human visitors but naturally followed by bot algorithms [38–40]. Methods using biometric or biostatics features [41–43], derived from users' mouse movements and keystroke data, in practice require introducing a client-side event logger.

Although additional data collecting mechanisms introduced in the aforementioned bot detection methods may provide more data, they may limit versatility of such methods and increase the cost of implementing them on various websites. On the other hand, HTTP-level features of the traffic observed on Web servers without interfering with the website or server software turn out to be efficient discriminators between bots and legitimate users [44,45]. Differences in bot and human traffic patterns have been widely investigated, typically based on data recorded in Web server access logs. Some features, representative for bot requests, have been identified for various websites [5,14,46,47], including the following: a tendency to ignore image files, lower volumes of HTTP response data, higher rates of unassigned referrers, requests of type HEAD, and erroneous requests. Moreover, resource request patterns, represented as sequences of resource types, can strongly differentiate the behavior of bots and humans [8,48,49].

HTTP feature–based approaches to bot detection employ traffic pattern analysis or machine learning techniques. They typically aim at offline bot detection, i.e., the task of categorizing the historical HTTP data on entire sessions completed on the server.

Traffic pattern-based approaches utilize statistical properties of HTTP request features, like types of downloaded resources [8,48–50] or inter-arrival times [51], to construct probabilistic session models.

Approaches exploiting machine learning techniques differ in the selection of relevant session features, techniques used, as well as methodology for session extraction and experimental classifier evaluation. Supervised session classifiers have been implemented with decision trees [14,17,47,52–54], [55], random forest [38,55], neural networks [6,14], [38,55], logistic regression [14], [55], support vector machines [6,17,53], [38,55],Saputra13,Jacob12, Bayesian classifiers [17,56–58], [55], k-Nearest Neighbors (kNN) [17], [55], [56], and ensemble methods [15], [38]. Unsupervised classification approaches have used such algorithms as k-means and GPCM (Graded Possibilistic c Means) [6,59], Information Bottleneck [60], PSO (Particle Swarm Optimization)-based clustering [61], DBSCAN (density-based spatial clustering of applications with noise) [62], SOM (self-organizing maps) [63,64], Modified ART2 (Adaptive Resonance Theory 2) [64], and MCL (Markov clustering) algorithm [65].

The high efficiency of bot detection approaches, reported in literature, confirmed the inherent differences characterizing bot and human traffic. However, the analysis of related work showed that very few studies addressed the problem of on-the-fly detection. In this context, the most relevant study is [8], which presents a method based on resource request patterns to classify sessions in real time: this method has been selected as a reference approach for a comparative analysis study detailed in Section 5. Two other studies, [47] and [52], analyzed performance of decision tree-based classification methods for various numbers of requests observed in session. However, they both considered page requests (corresponding to user clicks) instead of HTTP requests. In [47], the focus was rather on determining a minimum number of page requests needed to identify robot sessions with reasonably high accuracy. In [52] evaluation experiments were performed for some fixed minimum numbers of page requests that must be observed in session before taking up a classification decision. Due to these methodological options, the efficacy of the methods in [47] and [52] for a full, real-server session dataset is not known. Conversely, we address the problem of detecting Web bots as soon as possible, without enforcing the minimum number of observations. Furthermore, we both evaluate "global" performance scores of our method for the whole session dataset and analyze the scores in terms of the number of requests sufficient to classify a session.

4. Methodological framework

4.1. System architecture

The general framework of our approach is illustrated in Fig. 1. It is logically divided into two main processing lines.

The *back-end line* represents activities that are performed periodically, triggered by specific performance conditions: preprocessing of historical HTTP request data, training the neural network model, monitoring and storing classification performance scores at the output. Decrease in performance will trigger re-training of the neural network estimator.

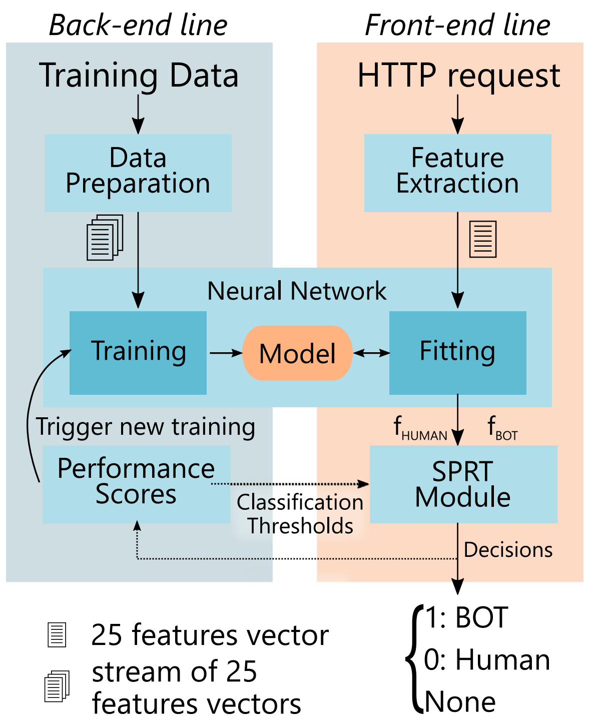


Fig. 1. A graphical sketch of the implemented framework.

The *front-end line* includes activities performed in real time for each incoming request. These include feature extraction and two-stage classification. For each individual request the neural network estimates its a posteriori class probability (f_{HUMAN} and f_{BOT} in Fig. 1). A sequential estimation for requests of the same session is used to make a decision to output either 1 (bot) or 0 (human). If the session ends before deciding, it is labeled as *undecided* and possibly processed according to some application-specific policy: for instance, a CAPTCHA might be presented, or the session can be marked for off-line analysis.

4.2. Choice of features

The choice of features is a crucial step in any recognition task. Several considerations play a role in this design decision. In the light of the survey in Section 3, we now outline the rationale followed for selecting the feature set.

As a first criterion, it is possible to use application-specific quantities, related to the semantics of the web site or application, for instance, by analyzing the content provided by users in a forum; or alternatively to select application-agnostic features, related to the HTTP(S) protocol. We opted for the latter choice because it has the advantage of being independent of the website semantics, and largely of the server software as well.

Another criterion involves the choice of quantities measurable at each request, versus summary indexes, like average, counts, and percentages. The latter carry information that is useful for offline recognition, and as noted in Section 3, this is the generally adopted approach in the literature when using HTTP-related features. Indeed, previous work has demonstrated their discriminating power even in an unsupervised setting [59,60]. However, when one of the goals is to minimize decision time, it is not feasible to wait for sufficient data to allow meaningful statistics. The experiments show that it is possible to have a decision lag as short as two, or even one request. To attain these performances it is mandatory to resort to instantaneously available real-time data.

For completeness we mention a third criterion, which however does not apply to the case at hand. Features can be engineered by experts, or alternatively optimized (learned from data). When dealing with signals, e.g., images, the data themselves come in a natural representation, for instance pixels. Learning features amounts to use optimization to find meaningful embeddings of pixel patterns. However, in the present application there is no unique natural set of measurements, so the only possible choice is expert-engineered features.

Once a – possibly redundant – set of features has been identified, a minimal and most useful subset can be selected either by reviewing experimental results, or systematically by means of *feature selection* procedures. This approach has been preliminarily applied [66] to the choice of the feature set to be used in the present work.

In summary, the choice of the features was motivated by their availability in real time, proven efficiency of using HTTP-level features in the offline bot detection, and by results of our preliminary exploratory data analysis. The resulting feature set is described below.

4.3. HTTP request features used in classification

As shown in Fig. 1 in the *front-end* processing line, descriptive features are extracted from HTTP headers (nine features) and Apache server timestamps (one feature).

Seven features are taken from HTTP request headers:

- Feature *method* corresponds to HTTP method specifying an action to be performed on a target resource. Methods typical for human users are GET, used to download the Web contents, and POST, used to submit data to the server. In contrast, robots often use HEAD method to retrieve only Web metadata [5,14]. Related features applied in previous offline bot detection studies have been based on percentages of requests in session made with GET, POST, and HEAD methods [6,14,15,17,47,52,63,65].
- Feature *is_referrer_empty* takes `true` if HTTP referrer is not provided and `false` otherwise. Most bots are designed to send requests with empty referrers in contrast to human users, who interact with the site via the Web interface [5,14,47]. Related features of Web sessions considered in the literature have been percentage of requests with empty referrers [6,14,15,17,47,62–65], and a switching factor on empty referrer [54,63,65].
- Five features are determined from request URI and describe the type of a target resource, which may (but does not have to) be one of the following: a page description file (feature *is_page* = `true`), a graphic file (*is_graphic* = `true`), a script/program file (*is_script* = `true`), a style sheet file (*is_style* = `true`), or a data file (*is_datafile* = `true`). Humans and bots reveal evident differences in target resource types; in particular, humans navigate through the website according to a logical structure of hyperlinks and a common access pattern for each visited page includes one page description file and a subsequent sequence of embedded (mostly graphic) files. Robots differ in types of requested resources depending on their functionalities but two common tendencies may be observed: they either ignore embedded files or limit themselves to only a certain type of files [5,16,46]. Hence, the following session features have been applied in the offline bot detection approaches so far: percentages of requests for various resource types [14–17,47,52,62–65], ratio of requested file type switch [54,62,63,65], and image-to-page ratio [6,17,62–65]. The type of target resource was also a basis of DTMC method, used as a reference approach in our experimental study.

Two request features are determined from HTTP response headers:

- Feature *response_status* corresponds to HTTP response status code, which indicates whether a request was successfully completed on the server. Code 200 is the standard response for successful requests, dominant in human sessions. Robots instead are more likely to request outdated or non-existent files so they typically have a higher rate of erroneous requests, especially the ones with status codes of type 4xx (meaning the client error) [14,46,52]. Percentage of requests with response status 4xx has been the most common feature applied in Web session classification [6,14–17,47,52,57,62–65].
- Feature *response_size* is taken directly from the response header field specifying volume of data sent to the client. Since most bots tend to ignore graphic files, data transfers are usually much smaller for them than for humans. Consequently, the total response size in session has been commonly used in the offline bot detection studies [6,14,15,58,61–65].

The last request feature is derived from Apache HTTP server timestamps of request arrival times so it is also easily available in real time:

- Feature *inter_arrival_time* is the time interval between timestamps of the current request and the preceding one. Due to typically distinct styles of browsing the Web by humans and bots (page views via the Web interface vs. algorithmically-driven accesses), temporal access patterns of both client types are inherently different. This has been reflected in classification tasks by such session features as the average or standard deviation of time between subsequent requests [14,15,65].

The features chosen are listed in Table 1 and can be grouped into three data types, each requiring different pre-processing actions:

- numerical features (N) are standardized by subtracting the mean and scaling to unit variance;
- categorical features (C) are one-hot encoded, i.e., represented as a bit vector of all zeros except one;
- boolean features (B) are represented by 0 for `false` and 1 for `true`.

Although the data for real-time operation are obtained directly from requests, historical data for classifier training may also be collected from server logs or from a database maintained at the server (Fig. 1, *back-end* line).

After feature extraction and pre-processing, each request at the server is represented as a 25-feature vector, which is then submitted to the two-stage classification process.

4.4. The idea behind two-stage classification

The described task is to state whether a session can be referred to a human or a bot agent. It is a binary classification task with sequentially sampled input.

Although requests are treated as independent by the HTTP protocol, their sequence depends on the navigational pattern of the visiting agent. This is likely to be different between humans , identified by class 1 in our experiments, and bots, conversely associated to class 0 [47].

At step k, we consider the sequence of observations $x_1, \ldots, x_k$. The probability $p_1(k)$ that the sequence is of class 1 is given by

$$p_1(k) = \frac{\Pr(\text{class} = 1)}{\Pr(x_1, \ldots, x_k)} \times$$

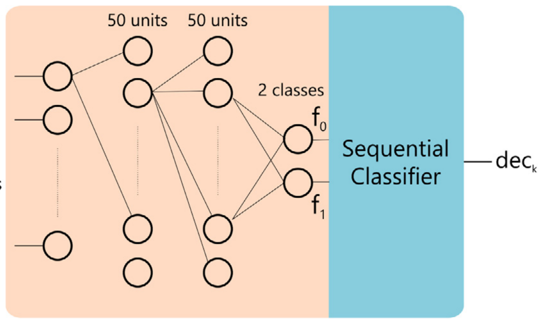


Fig. 2. The two-stage model.

$$\Pr(x_1|\text{class} = 1) \prod_{i=2}^{k} \Pr(x_i|x_1, \ldots, x_{i-1}, \text{class} = 1),$$

where $\Pr(A)$ is probability, $\Pr(A|B)$ conditional probability, and $\prod$ product of an indexed sequence.

The intractable complexity of this model is usually tackled by assuming a limited extent of temporal dependency. A common choice is to assume a fixed number of past observations (Markov assumption), most frequently one:

$$p_1(k) \propto \Pr(x_1|\text{class} = 1) \prod_{i=2}^{k} \Pr(x_i|x_{i-1}, \text{class} = 1),$$

where $\propto$ indicates proportionality.

One such method [8], presented in Section 5, is adopted in this paper as the reference for comparative evaluation. The method proposed here, however, follows a *naive* approach by ignoring any conditional dependency over successive requests:

$$p_1(k) = \prod_{i=1}^{k} \Pr(\text{class} = 1|x_i).$$

In the first stage it scores each individual HTTP request with a probability of being from either a human (class=0) or a bot (class=1). Then, in the second stage, it uses a sequential classification approach to combine the estimated probabilities of multiple requests and make the final decision as soon as the degree of confidence is satisfactory.

The naïve assumption corresponds to the hypothesis that, given an appropriate description, looking at the mix of request types is sufficient to discriminate between bots and humans. To avoid overfitting, the number of adaptable parameters must be kept at a minimum, especially when dealing with estimates of probabilities which are in principle an ill-posed problem. Accounting for previous observations exponentially increases the parameter count, so in this respect the optimum choice is a memoryless system, while a Markov structure with memory 1 appears as a possibly reasonable trade-off. However, the comparison with an order-1 Markov method presented here confirms that the memoryless approach achieves very good results, outperforming the reference while featuring less model parameters to be fitted.

4.5. The two-stage classification model

The classification model is depicted in Fig. 2 as a schematic representation of the main building blocks. The left box represents the internal structure of a neural network, while the right block is the sequential classification module, implementing a Sequential Probability Ratio Test.

We will refer to this model as NNSEQ, as the cascade of a neural network estimator and a sequential decision maker.

Table 1
Original request features before pre-processing.

Name	Type		Description
inter_arrival_time	N	int	Time interval between timestamps of the current request and the preceding one (in seconds)
method	C	string	HTTP method specifying an action to be performed on a given resource (e.g., GET, HEAD)
response_status	C	int	HTTP response status code (e.g., 200, 403, 404)
response_size	N	double	Volume of data in the HTTP response (in kilobytes)
is_referrer_empty	B	bool	Whether the HTTP referrer is known (`false`) or not (`true`)
is_page	B	bool	Whether the requested resource is a page description file (`true`) or an embedded object file (`false`)
is_graphic	B	bool	Whether the requested resource is a graphic file (`true`) or not (`false`)
is_script	B	bool	Whether the requested resource is a script/program file (`true`) or not (`false`)
is_style	B	bool	Whether the requested resource is a style sheet file (`true`) or not (`false`)
is_datafile	B	bool	Whether the requested resource is a specific data file (e.g., a zipped file) (`true`) or not (`false`)

4.5.1. Stage 1: Posterior probability estimation for individual requests

The first stage is a soft classifier that, given one observation x (a single request), outputs an estimate of the posterior probability $f_1(x)$ (the conditional probability of class 1 given the observation x) that the request is from a bot, as well as the corresponding probability $f_0(x) = 1 - f_1(x)$ that the request is from a human. This is done without considering any context information, i.e., independently of the previous requests received, according to the naive assumption previously described.

To produce this estimate, there are several options. Our choice is a dense neural network with a logistic (equivalent to softmax) output and optimized with respect to the cross-entropy objective. As a non-linear generalization of logistic regression, this classifier is especially well-suited to posterior probability estimation problems.

4.5.2. Stage 2: Sequential classification

The sequential nature of the session is handled by the second decision stage which implements Wald's Sequential Probability Ratio Test [67] to estimate the class posterior probability as new requests are received by the system. The original test is based on a sequential independence assumption and therefore is suitable for the case at hand, which is designed under the same assumption.

Let x_k be the kth observation in a sequence and $f_1(x_k), f_0(x_k)$ be the posterior probabilities of observation x_k to be of class 1 (bot) or class 0 (human), respectively. Under the memoryless "naïve" assumption, these probabilities reduce to

$$p_0(k) = \prod_{i=1}^{k} f_0(x_i)\,; \qquad p_1(k) = \prod_{i=1}^{k} f_1(x_i) \tag{1}$$

The ratio of these probabilities at step k is therefore:

$$R = \frac{p_1(k)}{p_0(k)} = \frac{\prod_{i=1}^{k} f_1(x_i)}{\prod_{i=1}^{k} f_0(x_i)} = \prod_{i=1}^{k} \frac{f_1(x_i)}{f_0(x_i)} \tag{2}$$

or, in terms of cumulative log-likelihoods:

$$L = \log p_1(k) - \log p_0(k) = \sum_{i=1}^{k} \Big(\log f_1(x_i) - \log f_0(x_i) \Big). \tag{3}$$

The log transformation reduces sensitivity to strongly imbalanced values and to numerical precision limits.

Given two predefined threshold values $T_0 < T_1$, the decision output at the sequential classification stage at step k is:

$$dec_k = \begin{cases} 1 & \text{if} \quad L \geq T_1 \\ 0 & \text{if} \quad L \leq T_0 \\ \mathit{None} & \text{otherwise.} \end{cases} \tag{4}$$

If desired probabilities of false positives α and false negatives β are given, in principle T_1 and T_0 can be computed as:

$$T_1 = \log \frac{\beta}{1-\alpha}\,, \qquad T_0 = \log \frac{\alpha}{1-\beta}\,. \tag{5}$$

Under ideal hypotheses, this sequential probability ratio approach is guaranteed to converge to a decision with the minimum number of observations for a given pair (α, β).

However, the problem as stated does not satisfy the ideal assumptions. The first issue is that, obviously, it is not possible to actively sample requests from a session at will: the number of requests is not controllable and may be lower than that required for a decision at a prescribed confidence level. As a result, some sessions may ultimately remain *undecided*. The classification problem therefore has a three-state output since it includes a reject option [68]. Validation of the result (Section 7) takes this fact into account.

A second issue refers to the applicability of (5). Given the "black box" approach used to estimate probabilities, a non-quantifiable estimation error might be present in the values of f_0 and f_1. Moreover, the naive assumption may introduce additional errors in the evaluation of L. Consequently, (5) is an approximation whose error cannot be assessed a priori. The values of T_0 and T_1 must be estimated from the data (see 4.5.3).

4.5.3. Parameter values

The problem of real-time detection with minimum lag has two different goals: classification quality and detection speed (minimum number of observations before making a decision). These are inherently conflicting goals, so it is an intrinsically multi-objective optimization problem [69].

The NNSEQ method requires setting the thresholds T_0 and T_1. These influence the trade-off between classification with high confidence and early decision, so their values should maximize classification quality metrics (in particular we focus on *F1*) and at the same time minimize the number of classification steps (k_{90}, the 90th percentile of the maximum number of classification steps, in our case). A similar problem is encountered in the case of DTMC [8], where values for two parameters have to be properly set: the minimum number of requests that have to be analyzed in a session, namely $k_{\min}$, and a probability decision threshold Δ.

We therefore adopt the multi-objective setting. Rather than combining multiple different scores into a single objective with arbitrary weights, we independently measure the two objectives, obtaining a set of pairs of measurements which, being two-dimensional, cannot be ordered naturally.

Among all the measurement pairs obtained while exploring the search space, the *Pareto optima* or nondominated solutions are these solutions that cannot be improved by one objective without sacrificing their quality in the second objective. The set of Pareto optima is called the *Pareto frontier*. From the Pareto frontier one may decide *a posteriori* the desired balance between the two objectives [70].

With this approach we are able to select empirically (based on data) the values of the two tunable parameters T_0 and T_1, choosing the best available trade-off according to desired criteria. Since it applies generally to any method-specific parameters, it also makes it possible to match the performance of the NNSEQ

Table 2
List of parameters used.

Name	Algorithm	Description
T_0	NNSEQ	decision threshold for the negative class (humans), tuned through Pareto optimization
T_1	NNSEQ	decision threshold for the positive class (bots), tuned through Pareto optimization, $T_0 < T_1$
α	SPRT [67]	given desired false positives rate – not used in NNSEQ due to the use of data-driven approach to tune T_0 and T_1
β	SPRT [67]	given desired false negatives rate – not used in NNSEQ due to the use of data-driven approach to tune T_0 and T_1
Δ	DTMC	confidence threshold to exceed in order to make a decision
k_{min}	DTMC	minimum number of request to be analyzed before making any decision

```
1:  Input:
2:      $S = \{\mathbf{x}_1, \mathbf{x}_2, \dots \mathbf{x}_K\}$          ▷ session (seq. of $K$ requests)
3:      $T_1$ and $T_0$                                      ▷ the decision thresholds
4:  Output:
5:      $dec$                                                ▷ decision on session
6:      $k$                                                  ▷ number of steps taken
7:  procedure CLASSIFY-SESSION($S$)
8:      $L \leftarrow 0$
9:      for $k = 1 \dots K$ do
10:         $dec, L \leftarrow$ CLASSIFY-REQUEST($\mathbf{x}_k, L$)
11:         if $dec \neq None$ then
12:             return $dec, k$
13:     return $undecided$
14: procedure CLASSIFY-REQUEST($\mathbf{x}, L$)
15:     $f_0(\mathbf{x}), f_1(\mathbf{x}) \leftarrow$ NNET($\mathbf{x}$)
16:     Compute $L$ using (3)
17:     if $L \geq T_1$ then
18:         $dec_k \leftarrow 1$                             ▷ decision is bot
19:     else if $L \leq T_0$ then
20:         $dec \leftarrow 0$                               ▷ decision is human
21:     else
22:         $dec \leftarrow None$                            ▷ no decision
23:     return $dec, L$
24: procedure NNET($\mathbf{x}$)
25:     Estimate $f_0(\mathbf{x}), f_1(\mathbf{x})$ with neural network
26:     return $f_0(\mathbf{x}), f_1(\mathbf{x})$
```

Fig. 3. On-the-fly bot detection algorithm.

method with that of the DTMC method, so as to obtain a fair comparison.

The parameters described in the theoretical background and considered for the experimental setup of this research activity are summarized in Table 2.

4.6. The algorithm

The decision process is described in Fig. 3. The algorithm (procedure CLASSIFY-SESSION) is parameterized by the decision thresholds T_0 and T_1. It reads each HTTP request in a session, as vectors $\mathbf{x}_k$. The class likelihoods $f_1(x_k)$ and $f_0(x_k)$ for observation k are produced by the estimator (procedure NNET).

All estimates from observations 1 to k are then combined together to compute L according to (3) (procedure CLASSIFY-REQUEST). If either threshold is crossed, the corresponding option (bot or human) is taken and no further observations are considered. Otherwise, decision is set to *None* and additional requests in the session will be processed until a decision is taken, or the session ends.

5. A reference bot detection method

To demonstrate the effectiveness of the proposed method we compare it against another real-time bot detection approach, recently proposed in [8]. To the best of our knowledge, this is the only method from the literature that can be directly compared to ours as it also uses HTTP request features and aims at taking a classification decision as early as possible. To allow for a fair and thorough comparison, for lack of an available implementation and data, the method was entirely re-implemented. This made it possible to comparatively test the two methods using exactly the same datasets and evaluation methodology.

The basic idea of the method is representing a session as a sequence of requested resource types, a "resource request pattern". This is modeled with a first-order Discrete-Time Markov Chain (DTMC) whose states are resource types.

To this end, each server resource, identified by its URI, is assigned a type depending on the file extension. We defined the following resource types:

- *web* for web page and script files (e.g., html, htm, php, cgi, asp, jsp, js),

- *text* for text-formatted files (e.g., txt, xml, sty, tex, c, cpp, java, css),
- *doc* for rich-text documents (e.g., doc, xls, ppt, pdf),
- *img* for images (e.g., bmp, jpg, png, tiff, raw, ico),
- *av* for multimedia files (e.g., avi, mp3, mpg, au),
- *prog* for program files (e.g., exe, dat, bat, dll, msi, jar),
- *compressed* for compressed files (e.g., zip, gz, 7z, rar),
- *malformed* for malformed requests or unknown file types.

This categorization is consistent with the one proposed in [8], which however has an additional request type for directory contents, unused in our case.

A DTMC model is defined by a vector of starting probabilities $\mathbf{s} = (s_i)$ and a transition probability matrix $\mathbf{P} = \left(p_{ij}\right)$. These are trained based on resource request patterns of sessions in a training dataset. During the training phase a separate DTMC model is developed for each class. Let $\mathbb{R} = (\mathbf{s}_r, \mathbf{P}_r)$ and $\mathbb{H} = (\mathbf{s}_h, \mathbf{P}_h)$ be DTMCs trained with robot and human sessions, respectively.

To make the decision, let $\boldsymbol{X} = (x_1, \ldots, x_K)$ be the resource request pattern of a session with K requests observed on the server. The log-probability that a DTMC will generate $\boldsymbol{X}$ at step $k \le K$ is:

$$\log \Pr(\mathbf{X}|\mathbf{s}, \mathbf{P}) = \log \mathbf{s}_{x_1} + \sum_{i=2}^{k} \log p_{x_{i-1},x_i}. \tag{6}$$

Given $\mathbf{X}$, $\Pr(\mathbf{X}|\mathbb{R})$ represents the probability computed using (6) for $\mathbb{R}$ (bot), and similarly $\Pr(\mathbf{X}|\mathbb{H})$ for $\mathbb{H}$ (human).

The method requires two decision parameters: $k_{\min}$ is the minimum number of requests to be analyzed, to gain a degree of confidence before making a decision; and Δ is the probability decision threshold. The recognition procedure at the kth request x_k in a session $\mathbf{X}$ is the following:

1. Update $\log \Pr(\mathbf{X}|\mathbb{R})$ and $\log \Pr(\mathbf{X}|\mathbb{H})$
2. If $k \ge k_{\min}$, compute $D = \log\left(\Pr(\mathbf{X}|\mathbb{R})/\Pr(\mathbf{X}|\mathbb{H})\right)$.
3.
   - If $|D| < \Delta$, do not classify $\mathbf{X}$ at step k.
   - Otherwise, if $D \ge 0$, classify $\mathbf{X}$ as a robot session;
   - or if $D < 0$, classify $\mathbf{X}$ as a human session.

If a session ends before being classified, it is *undecided*. This is an important difference from the procedure described in [8], where each undecided session was classified using "offline" detection, i.e., taking all the requests of the session into account. Such approach might be feasible for performance evaluation; in on-field operation, however, waiting for a session to complete is not compatible with real-time operation. In the experiments, we left these sessions *undecided*.

6. Experimental evaluation

6.1. Dataset description

Access log data of a real e-commerce website were used.[2] The website consists of an online bookstore, offering mainly conventional books, audiobooks, and computer games, as well as pages with entertainment and multimedia content, like short movies, mini games, or quizzes.

The bookstore is implemented on the osCommerce platform and hosted on Linux Apache HTTP server with PHP and MySQL support. The access log data is recorded according to *NCSA Combined* log format and covers the period from April 1st to 30th, 2014. The whole request dataset contains 1 397 838 HTTP request entries, totaling 13 395 sessions reconstructed according to the procedure described in Section 2, after eliminating one-request sessions, one-page sessions, and sessions generated by the website administrator and administrative software.

[2] The website identity cannot be revealed in the paper due to a non-disclosure agreement.

6.2. Data preparation

Each session was assigned a ground truth label (bot or human) based on two online databases of user agent strings and IP addresses, recognized by experts as representing bots or Web browsers, augmented with heuristics concerning session features indicative of a bot [48]. A primary source was the Udger online database [71], containing 2832 and 843 known user agent strings of bots and browsers respectively, and 996 657 known bot IP addresses. A supplementary source was User–agents online database [72] with 2459 known user agent strings.

Session labeling rules were defined as follows:

1. A session was labeled as a bot if at least one of the following conditions was met:
   - user agent classified in the Udger database as "crawler", "e-mail client", "library", "validator", "multimedia player", or "offline browser";
   - user agent classified in the User–agents database as a robot;
   - user agent contained a keyword suggesting a bot ("spider", "crawler", "robot", "worm", "search", "track", "harvest", "hack", "trap", "archive", "scrap", etc.);
   - IP address classified in the Udger database as "crawler", "fake crawler", "known attack source — http", "known attack source — mail", or "known attack source — ssh";
   - at least one of the following indicators was true: the image to page ratio equal to 0, 100% of page requests with empty referrers, 100% of response status codes of type 4xx, or 100% of requests with HEAD method;
   - file robots.txt requested.
2. A session was labeled as a human if the user agent was classified in the Udger database as "browser" or "mobile browser".
3. Otherwise, a session remained unlabeled.

Unlabeled sessions were excluded from the dataset, whereas 6190 sessions labeled as bots and 7200 labeled as humans were used in the analysis (therefore classes were roughly balanced).

Training and model selection for the neural network estimator were performed by 10-fold cross-validation, using proportion-preserving subsets containing 619 bot and 720 human sessions.

6.3. Performance evaluation

Experimental studies were performed separately for the NNSEQ and DTMC methods on the same training and test streams and comparing the obtained scores. In the following, the methodology and metrics used are described.

6.3.1. Evaluation scenarios

By considering bot-generated sessions as *positive* and human-generated ones as *negative*, the overall classification results may be quantified by means of a confusion matrix reporting the number of true and false positives (*TP* and *FP*, respectively) and true and false negatives (*TN* and *FN*, respectively).

An additional measure is the number of *undecided* sessions. Two evaluation scenarios can be defined, depending on whether this information is used or not:

Scenario 1: Performance scores are determined only on account of the sessions classified as a bot or human: no *undecided* sessions are considered.

Scenario 2: Performance scores are determined taking into account *undecided* sessions. To reflect this real-life aspect in our study, here we assume that undetected bots are humans. Thus, *undecided* sessions are included into *negatives*: *FN* is increased by undecided bot sessions and *TN* by undecided human sessions.

6.3.2. Performance metrics

Classifier performance is described by the following metrics.

- *Recall* = *TP* / (*TP*+*FN*), fraction of positive sessions that are correctly classified.
- *Precision* = *TP* / (*TP*+*FP*), fraction of positive decisions that are correct.
- *Accuracy* = (*TP*+*TN*) / (*TP*+*TN*+*FP*+*FN*), fraction of correct classifications.
- *F1* = 2 · *Precision* · *Recall*/(*Precision* + *Recall*), harmonic mean of recall and precision, overall quality of the classifier.
- k_{90} – the 90th percentile of the maximum number of classification steps, i.e., the maximum number of steps to classify 90% of non-undecided sessions; ability to take up early decisions.
- $P_c(k)$, percentage of sessions classified at step k.
- $P_u(k)$, percentage of undecided sessions that ended at step k.
- $CP_c(k) = \sum_{i=1}^{k} P_c(k)$, cumulative percentage of sessions classified in $\leq k$ steps.
- $CP_u(k) = \sum_{i=1}^{k} P_u(k)$, cumulative percentage of sessions that ended in $\leq k$ steps but not classified.
- P_c, percentage of classified sessions.

Performance rates were averaged over the ten cross-validation rounds. In the following, results are first discussed regarding the overall classifier effectiveness, from the perspective of all completed sessions. Afterwards, the scores are analyzed considering the number of requests which was sufficient to classify a session (i.e., as a function of k).

6.3.3. Parameter setting

The architecture of the neural network was optimized empirically, since reliable objective criteria are not available. The resulting layout includes two hidden layers and a structure of 25-50-50-1, that is, the input layer has 25 units, corresponding to the input data size, and is followed by two 50-unit hidden layers with ReLU activation function and a single logistic output unit, corresponding to softmax in the two-class case.

The decision thresholds T_0 and T_1 were optimized according to the procedure described in Section 4.5.3; the results of the procedure are discussed in the following.

7. Results and discussion

In this section, first, values of the method-specific parameters are determined through Pareto optimization, and then, classification results for both methods are analyzed. The computational burden and some implementation issues are also discussed.

7.1. Tuning the parameter values

By simultaneous varying values of thresholds T_0 in [−5.5, −0.1] and T_1 in [0.1, 5.5], a set of solutions was computed for both methods. As a guideline, if we set $\alpha = \beta$, these values correspond to varying them from a minimum of about 10^{-5} to a maximum of about 0.44. For each solution, mean performance scores were determined by cross-validation under scenario 2. Regarding the objective functions: maximization of *F1* and minimization of k_{90}, the search space used to find the Pareto frontier is two-dimensional. Similarly, the search space was determined for the DTMC method by varying values of Δ from 0.01 to 1.9 and $k_{\min}$ from 1 to 21. Fig. 4 visualizes the resulting Pareto frontiers for both methods (in Fig. 4(b) only solutions for Δ in the range of 0.01 to 0.29 are shown since, for $\Delta > 0.29$, *F1* was too low).

As it can be seen in Fig. 4(a), our method yielded extremely good results. For all combinations of the threshold values *F1* exceeds 0.92, which indicates very high rates of both recall and precision. Moreover, in all cases $k_{90} \leq 3.2$, which means that in 90% of classified sessions a decision was determined well before the 4th request. The Pareto frontier contains 18 points; among them there is one extreme solution with the minimum k_{90} equal to 1.0 and *F1* equal to 0.93, which was achieved for $T_0 = -2.2$ and T_1 = 1.9; the second extreme is the point with the maximum accuracy equal to 0.96 and k_{90} equal to 3.0, achieved for $T_0 = -5.4$ and T_1 = 4.6.

Fig. 4(b) shows that the DTMC method is much more sensitive to its parameter values and the achieved performance scores are much worse than for NNSEQ. Its search space covers much wider range of values in both dimensions (in Fig. 4(b) only the best solutions, with *F1* above 0.4 and $k_{90} \leq 16$, are plotted). The Pareto frontier contains 33 points, overlapping in some cases. There are 15 points with the minimum k_{90} equal to 1.0 and the corresponding *F1* equal to 0.51 — they were achieved for $k_{\min} = 1$ and Δ ranging from 0.01 to 0.15. The maximum *F1*, equal to 0.78, with the corresponding $k_{90} = 4$, was achieved for three solutions with $k_{\min} = 2$ and Δ ranging from 0.18 to 0.2.

As an illustration, Fig. 5 shows the values of NNSEQ, both pointwise and cumulative, in two different short sessions, one of positive class (bot) and one of negative class (human). The sessions were chosen among those that required at least 8 requests before deciding, using as a reference the thresholds indicated ($T_0 = -5.5$, $T_1 = 4.6$).

7.2. Comparative analysis

From among all the Pareto-optimal solutions for each method, a case providing the maximum *F1* was selected for direct comparative analysis. In the case of multiple equivalent candidate points, ties were broken using the additional criteria of highest accuracy and lowest number of *undecided* sessions. As a result, the following parameter values have been used in the comparative experimental study: $T_0 = -5.4$, T_1 = 4.6 for NNSEQ and $k_{\min} = 2$, Δ = 0.18 for DTMC.

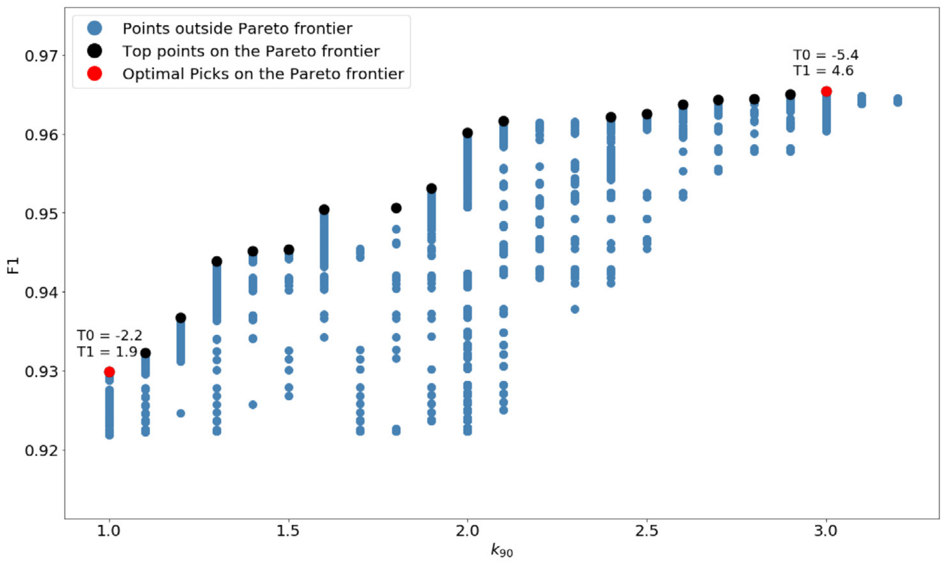
Overall results are summarized in Table 3.

Table 3
Classification results (10-fold cross-validation)

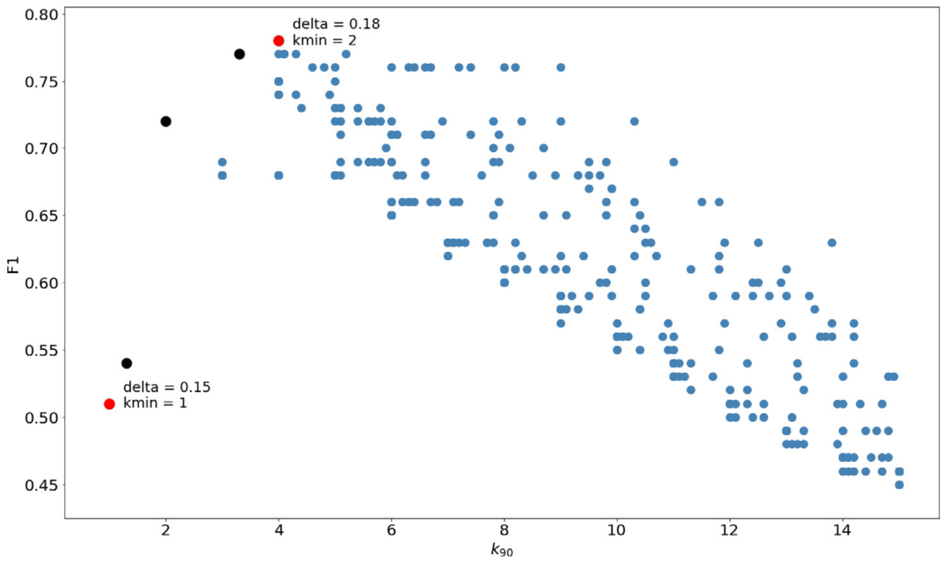
Metric (avg.)	NNSEQ	DTMC
#*TP*	577.3	429.2
#*TN*	710.5	494.7
#*FP*	9.5	50.4
#*FN*	32.5	58.5
#*undecided* – bots	9.2	131.3
#*undecided* – humans	0	174.9
k_{90}	3.0	4.0
P_c	99.31	77.13

The NNSEQ method was able to determine a decision for nearly all active sessions: only 0.69% sessions were left undecided and all of them were generated by bots. As for the DTMC method as many as 22.87% sessions ended before being classified, including 21.2% bots and 24.3% humans.

Fig. 6 shows cumulative percentage of classified and undecided sessions for initial steps (requests in sessions) for both methods. Fig. 7 illustrates percentages of sessions classified and undecided by NNSEQ at the initial steps, with the session breakdown according to their ground truth labels; Fig. 8 presents analogous results for DTMC.

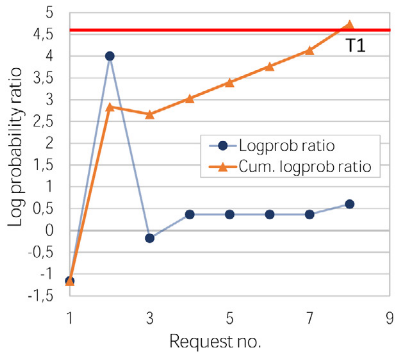


(a) NNSEQ

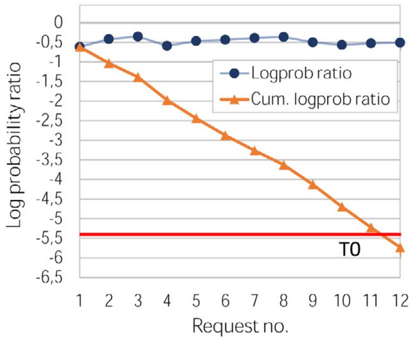


(b) DTMC

Fig. 4. Pareto frontier for NNSEQ and for DTMC.

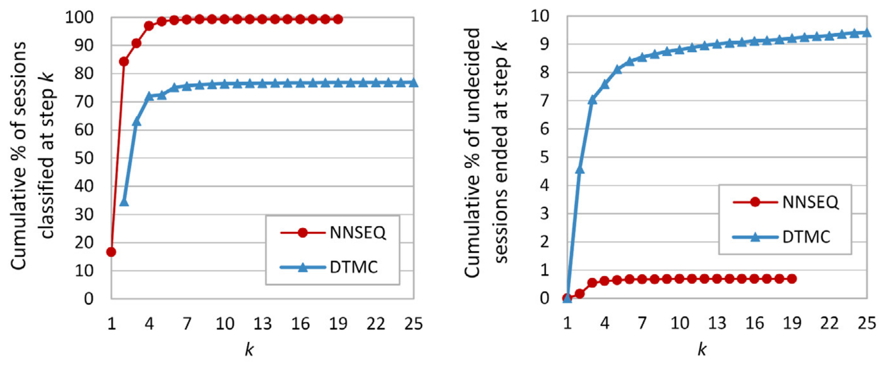


(a) Positive class



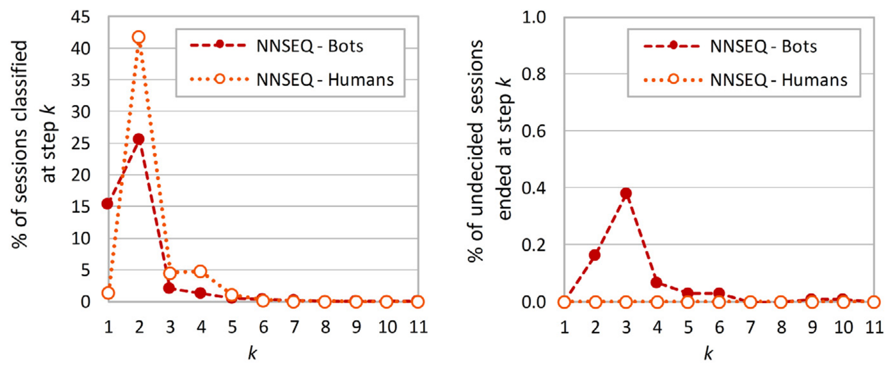
(b) Negative class

Fig. 5. Two sample sessions, shown up to the request at which the decision was taken. For each request, log probability ratio and cumulative log probability ratio as computed by NNSEQ are shown.



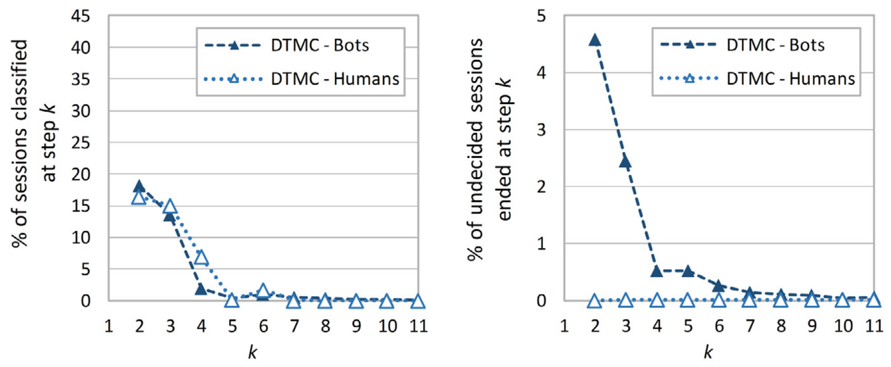
(a) $CP_c(k)$ – Classified sessions (b) $CP_u(k)$ – *Undecided* sessions

Fig. 6. Cumulative percentages of classified and *undecided* sessions as a function of the number of steps.



(a) $P_c(k)$ for NNSEQ (b) $P_u(k)$ for NNSEQ

Fig. 7. Percentages of classified and *undecided* NNSEQ sessions as a function of the number of steps.



(a) $P_c(k)$ for DTMC (b) $P_u(k)$ for DTMC

Fig. 8. Percentages of classified and *undecided* DTMC sessions as a function of the number of steps.

As it can be seen in Fig. 6(a), NNSEQ was able to classify 99% of sessions within six requests, with 16.7% being classified at the first step and the next 67.5% – at the second step. 15% of visitors classified just after analyzing the first request in session were bots; among 67% of visitors identified after two requests, 25% were bots and 42% were humans (Fig. 7(a)). Moreover, for NNSEQ the maximum number of requests before decision over the whole test dataset was 19 (Fig. 6(a)).

On the other hand, the DTMC-based method was able to classify only 75% of sessions after six requests and larger numbers of observations in session did not improve the results (the cumulative percentage of classified sessions in Fig. 6(a) flattens at about the 7th step), although some sessions are still being analyzed, even after receiving more than fifty requests. Furthermore, due to $k_{\min} = 2$, no decision was taken at the first step. Fig. 6(b) shows that the cumulative percentage of sessions ended before being classified for DTMC constantly increases with the increase in the number of requests observed in session. For both methods most of *undecided* sessions were extremely short ones, containing only two or three requests (Fig. 6(b)) and they were mostly bots (Figs. 7(b), 8(b)).

Regarding the quality of decisions, performance scores for both evaluation scenarios are reported in Table 4. The scores are also presented for initial steps in a cumulative form, considering

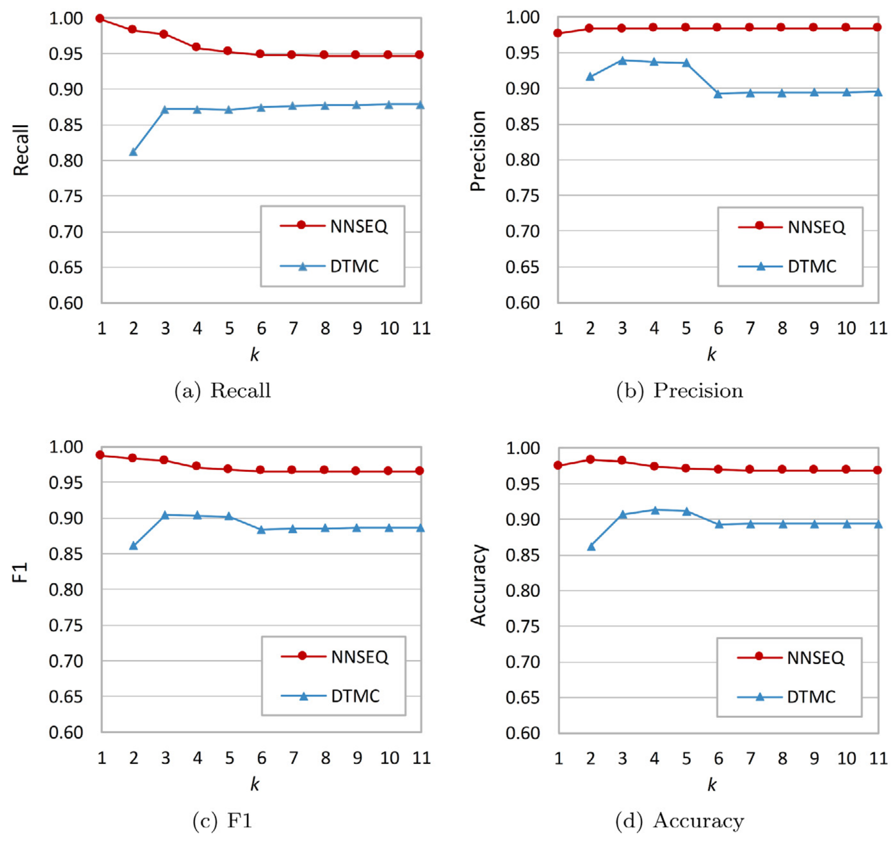


Fig. 9. Cumulative performance scores as a function of the number of steps (scenario 1 – excluding *undecided* sessions).

Table 4
Performance scores (10-fold cross-validation), computed without considering *undecided* sessions (scenario 1) and counting *undecided* sessions as humans (scenario 2).

Evaluation scenario	Metric (avg.)	NNSEQ	DTMC
Scenario 1	Recall	0.95	0.88
	Precision	0.98	0.90
	F1	0.96	0.89
	Accuracy	0.97	0.89
Scenario 2	Recall	0.93	0.69
	Precision	0.98	0.90
	F1	0.96	0.78
	Accuracy	0.96	0.82

the number of requests that were sufficient to take a classification decision (Fig. 9 for scenario 1 and Fig. 10 for scenario 2): for each step and the preceding ones the incremental number of true and false positives and negatives were computed and used to determine the corresponding performance scores.

Considering only classified sessions (scenario 1), one can observe in Table 4 very high efficiency of both approaches, with a clear advantage of the proposed NNSEQ which was able to correctly identify as many as 97% of all visitors, compared to 89% recognized by DTMC. Furthermore, NNSEQ achieved very high recall of 0.95 which confirms its capability of detecting bots on the fly to a high extent. Precision, which measures the performance with respect to erroneously classifying humans as bots, is even higher (0.98), which indicates that human visitors are not penalized by the algorithm. For comparison, recall and precision for DTMC amounted only to 0.88 and 0.90, respectively.

On the other hand, when undecided sessions were treated as undetected bots (scenario 2), performance scores of NNSEQ decreased only slightly whereas for DTMC they deteriorated drastically (except precision, which remained unchanged for both classifiers). NNSEQ was able to detect 93% of all bots on the fly, compared to only 69% of bots recognized by DTMC. Our method achieved both *F1* and accuracy of 0.96 while the corresponding scores for DTMC were only 0.78 and 0.82.

This leads to the conclusion that under stricter, real-life assumptions the proposed approach is very effective in detecting bots interacting with a Web server while DTMC leaves many of them undecided or misclassified. However, this scenario is more realistic, because in practice leaving a session undecided means that a bot was not detected. Thus, evaluating performance score with undecided sessions treated as undetected bots reflects the real power of a classifier. For this reason, the classifier performance at the initial classification steps is further analyzed just for scenario 2.

Fig. 10 demonstrates a very high efficiency of NNSEQ at the initial classification steps over the greatest amount of sessions, which is the main objective of our early bot detection approach. *F1* score was in the range of 0.96 to over 0.98, depending on the classification step, and the earlier the sessions were classified, the higher *F1* was achieved. Recall was evidently the highest for the first three steps and amounted to 0.998, 0.979, and 0.964, respectively. Precision was slightly lower for the first-step decisions, but nevertheless it always exceeded 0.98. All the performance scores for NNSEQ flatten out at about the 7th step, where most sessions have already been classified.

For DTMC the cumulative performance scores also stabilize after about seven requests (this is a similar conclusion as in [8]), but at much lower levels. Furthermore, in contrast to NNSEQ, which was able to classify all the sessions up to the 19th request, DTMC kept analyzing very long sessions which ultimately remained undecided (Fig. 6(b)).

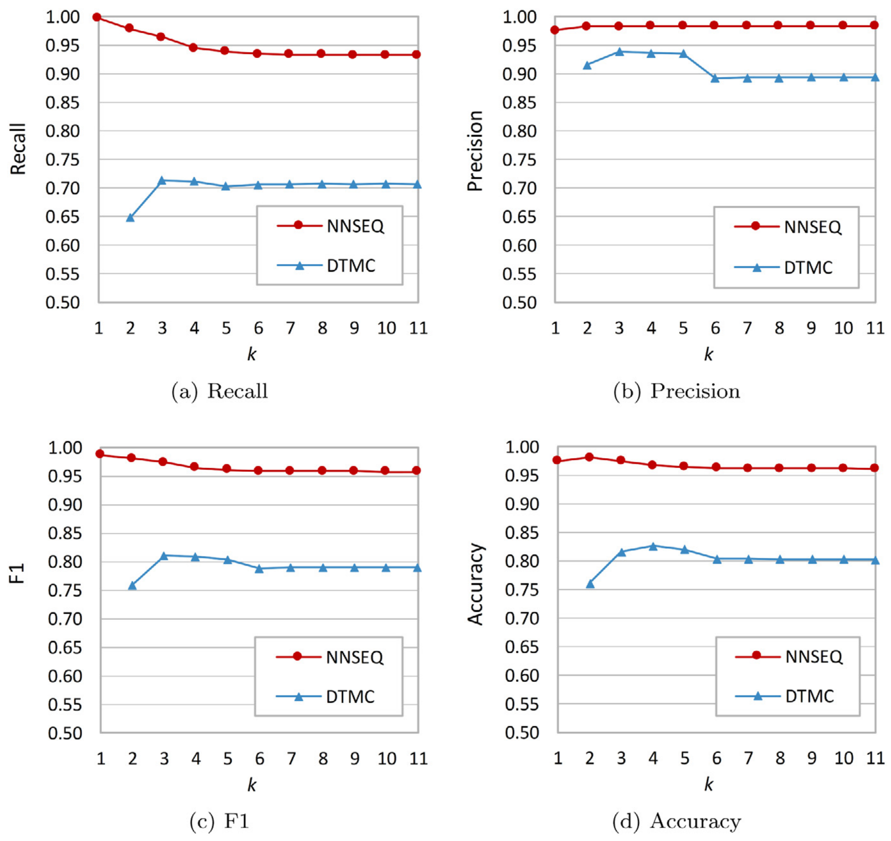


Fig. 10. Cumulative performance scores as a function of the number of steps (scenario 2 – with *undecided* sessions counted as humans).

7.3. Estimation of the computational complexity

An on-the-fly bot detection solution should evaluate every single incoming request to a server. As a consequence, for scalability it should be unintrusive to the Web server and should add only a minimal computation overhead to the request processing in real time. The NNSEQ method uses data that are readily obtainable from HTTP request headers without intervention by the server, so that it can be easily integrated.

From the computational complexity perspective, classification of a single request is performed in constant time. The neural network model, presented in Fig. 2, has a fixed structure, so $f_1(\mathbf{x})$ and $f_0(\mathbf{x})$ are computed in O(1) time (as well as O(1) space). Values of $\log p_1(k)$ and $\log p_0(k)$ may be stored as session variables on the server. The sequential probability ratio is also computed in O(1) time, so for N sessions the complexity of a sequential implementation is O(N), but the problem is inherently parallel, without inter-thread communication. Compared to the DTMC method, NNSEQ requires a similar amount of memory and limited additional time to classify one request. However, due to its better performance in early classification, the expected computational burden is smaller, since much fewer requests per session will be analyzed.

7.4. Limitations of the presented work

NNSEQ relies on binary classification with reject option of HTTP requests, assuming two classes: one class for human sessions and the second one for all Web bots aggregated. An alternative and natural extension of this approach would be a multinomial classification of bot requests, either consisting in differentiating benign and malicious robots or defining multiple robot classes depending on the connotation of their traffic patterns .

Reconstructing sessions from Web traffic at the HTTP level, despite being a common practice in bot detection research, has some disadvantages. Sessions based on IP addresses and user–agents may not be an exact representation of real Web sessions, mainly due to the existence of firewalls and proxies in the network, caching of popular Web content, as well as the fact that the advanced bot software may interact with a website using multiple IP addresses at the same time. Nevertheless, since we do not aim at characterizing bot traffic itself but focus on investigating differences in bot and human traffic, an HTTP-based approach is sufficient.

Our method has a point of weakness in being based on ground truth that may possibly contain errors — this is common for Web bot detection approaches based on supervised learning. On the other hand, previous works using unsupervised learning (e.g., [59]) have proven that there is a notable overlap between clusters and classes (cluster purity). Therefore, the ground truth is mostly reliable because it corresponds to actual differences in the data, and it was observed that – when there is a mismatch between unsupervised result and labels – there may be annotation mistakes in the ground truth. So mistakes can be identified. This also introduces the possibility of self-learning during operation, using semi-supervised learning.

Furthermore, classification performance evaluation through simulation experiments may be perceived as a limitation of our work. Implementing a bot detection method on a real Web server would make it possible to assess its efficacy, computational overhead and possible benefits of early detection of Web bots. On the other hand, a simulation-based approach provides a fully controllable experimental environment with exactly the same server workload and operating conditions arranged for two different bot detection methods, which ensures their fair comparison.

Our work is based on one e-commerce traffic dataset. Experimenting with multiple different datasets would undoubtedly provide a more complete and comprehensive picture of the study. We leave this issue to our future work.

8. Conclusion

In the paper a novel method for Web bot detection on a Web server in real time was proposed. The two-stage classifier combines a neural network model and a sequential probability ratio test to classify an active visitor as a bot or human as early as possible.

The proposed NNSEQ method is one of the first machine learning approaches for real-time Web bot detection. The extensive experimental study, tested on traffic streams from an actual server, showed that under strict real-time assumptions of server operation NNSEQ is able to determine a decision for nearly all active sessions, leaving only 0.69% of them undecided. The method is especially powerful given a very limited number of requests observed in session and achieves a very high efficiency at the initial classification steps. 99% of sessions were classified within six requests and the maximum number of requests before decision over the whole test dataset was 19.

Decision quality is very high: the method is able to correctly classify as many as 96% sessions; regarding Web bots, it is able to detect as much as 93% of all bots on the fly. Depending on the classification step, *F1* score ranges from 0.96 to over 0.98, and the earlier the sessions are classified, the higher score is achieved.

In contrast to other bot detection approaches (e.g., [8,47,52]), NNSEQ does not impose a threshold for the minimum number of requests that must be observed in a session before determining a classification decision. Furthermore, precision of 0.98 confirms that very few human visitors are erroneously classified as bots. This observation is especially important since bot detecting and blocking mechanisms should be as transparent as possible to human users.

The classifier can be easily implemented as a simple extension to a real Web server software and integrated with other fallback bot detection mechanisms, like a CAPTCHA test. In future work we are going to extend our approach with a feature selection stage, experimenting with different techniques for feature selection and extraction. We will also investigate some sequence modeling techniques, like recurrent neural networks [73] at the second decision stage. Furthermore, we will perform the experimental verification of the approach performance using multiple Web traffic traces from different domains, aiming at the generalization of our conclusions.

On the implementation side, if the recognition system is designed as a filter (for instance as a proxy server), its throughput might constitute a performance and scalability bottleneck, since each request should be analyzed before it is sent to the server. Therefore, future work will concentrate on a side-chain realization operating in parallel to the HTTP stream. This will allow the system integrator to choose, at deployment time, a suitable compromise between timeliness and scalability.

CRediT authorship contribution statement

Grażyna Suchacka: Conceptualization, Data curation, Investigation, Methodology, Software, Visualization, Writing. **Alberto Cabri:** Formal analysis, Investigation, Software, Visualization, Writing. **Stefano Rovetta:** Conceptualization, Formal analysis, Methodology, Writing. **Francesco Masulli:** Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This paper is based upon work from COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), supported by COST (European Cooperation in Science and Technology).

Alberto Cabri, Stefano Rovetta and Francesco Masulli are members of the National Group for Scientific Computing (GNCS) of INdAM, the Italian Istituto Nazionale di Alta Matematica "Francesco Severi".

References

[1] V. Geroimenko, Dictionary of XML Technologies and the Semantic Web, Springer-Verlag, London, UK, 2004.
[2] Bad Bot Report 2020: Bad Bots Strike Back, Tech. Rep., Imperva Incapsula, 2020, https://www.imperva.com/resources/resource-library/reports/2020-Bad-Bot-Report.
[3] I. Zeifman, Bot Traffic Report 2016, Tech. Rep., Imperva Incapsula, 2017, https://www.incapsula.com/blog/bot-traffic-report-2016.html.
[4] S. Gianvecchio, M. Xie, Z. Wu, H. Wang, Humans and bots in Internet chat: Measurement, analysis, and automated classification, IEEE ACM T. Netw. 19 (5) (2011) 1557–1571, http://dx.doi.org/10.1109/TNET.2011.2126591.
[5] G. Suchacka, Analysis of aggregated bot and human traffic on e-commerce site, in: Proc. FedCSIS'14, 2014, pp. 1123–1130, http://dx.doi.org/10.15439/2014F346.
[6] S. Rovetta, A. Cabri, F. Masulli, G. Suchacka, Bot or not? A case study on bot recognition from Web session logs, in: Quantifying and Processing Biomedical and Behavioral Signals, in: Smart Innovation, Systems and Technologies, vol. 103, Springer, 2019, pp. 197–206, http://dx.doi.org/10.1007/978-3-319-95095-2_19.
[7] A. Cabri, G. Suchacka, S. Rovetta, F. Masulli, Online web bot detection using a sequential classification approach, in: Proc. HPCC/SmartCity/DSS'18, 2018, pp. 1536–1540, http://dx.doi.org/10.1109/HPCC/SmartCity/DSS.2018.00252.
[8] D. Doran, S.S. Gokhale, An integrated method for real time and offline web robot detection, Expert Syst. 33 (6) (2016) 592–606, http://dx.doi.org/10.1111/exsy.12184.
[9] H. Chen, H. He, A. Starr, An overview of web robots detection techniques, in: Proc. Int. Conf. Cyber Security and Protection of Digital Services, Cyber Security'20, IEEE, 2020, pp. 1–6, http://dx.doi.org/10.1109/CyberSecurity49315.2020.9138856.
[10] A. Mason, Y. Zhao, H. He, R. Gompelman, S. Mandava, Online anomaly detection of time series at scale, in: Proc. Int. Conf. Cyber Situational Awareness, Data Analytics and Assessment, Cyber SA'19, IEEE, 2019, pp. 1–8, http://dx.doi.org/10.1109/CyberSA.2019.8899398.
[11] T. Berners-Lee, R. Fielding, H. Frystyk, Hypertext Transfer Protocol – HTTP/1.0, IETF RFC 1945, 1996.
[12] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, Hypertext Transfer Protocol – HTTP/1.1, IETF RFC 2616, 1999.
[13] D. Kristol, L. Montulli, HTTP State Management Mechanism, IETF RFC 2109, 1997.
[14] C. Bomhardt, W. Gaul, L. Schmidt-Thieme, Web robot detection – preprocessing Web logfiles for robot detection, in: New Developments in Classification and Data Analysis, Springer, Berlin, Heidelberg, 2005, pp. 113–124.
[15] D.S. Sisodia, S. Verma, O.P. Vyas, Agglomerative approach for identification and elimination of web robots from web server logs to extract knowledge about actual visitors, J. Data Anal. Inf. Process. 03 (2015) 1–10, http://dx.doi.org/10.4236/jdaip.2015.31001.
[16] A. Stassopoulou, M.D. Dikaiakos, Web robot detection: a probabilistic reasoning approach, Comput. Netw. 53 (3) (2009) 265–278, http://dx.doi.org/10.1016/j.comnet.2008.09.021.
[17] D. Stevanovic, A. An, N. Vlajic, Feature evaluation for web crawler detection with data mining techniques, Expert Syst. Appl. 39 (10) (2012) 8707–8717, http://dx.doi.org/10.1016/j.eswa.2012.01.210.
[18] D. Acarali, M. Rajarajan, N. Komninos, I. Herwono, Survey of approaches and features for the identification of HTTP-based botnet traffic, J. Netw. Comput. Appl. 76 (2016) 1–15, http://dx.doi.org/10.1016/j.jnca.2016.10.007.
[19] A. Jakóbik, F. Palmieri, J. Kołodziej, Stackelberg games for modeling defense scenarios against cloud security threats, J. Netw. Comput. Appl. 110 (2018) 99–107, http://dx.doi.org/10.1016/j.jnca.2018.02.015.
[20] S. Lysenko, K. Bobrovnikova, O. Savenko, A. Kryshchuk, BotGRABBER: SVM-based self-adaptive system for the network resilience against the botnets' cyberattacks, in: Proc. Computer Networks, CN'19, in: CCIS, vol. 1039, Springer, 2019, pp. 127–143, http://dx.doi.org/10.1007/978-3-030-21952-9_10.

[21] S. Ustebay, Z. Turgut, M.A. Aydin, Cyber attack detection by using neural network approaches: shallow neural network, deep neural network and autoencoder, in: Proc. Computer Networks, CN'19, in: CCIS, vol. 1039, Springer, 2019, pp. 144–155, http://dx.doi.org/10.1007/978-3-030-21952-9_11.

[22] M. Rahman, M. Rahman, B. Carbunar, D.H. Chau, Search rank fraud and malware detection in google play, IEEE Trans. Knowl. Data Eng. 29 (6) (2017) 1329–1342, http://dx.doi.org/10.1109/TKDE.2017.2667658.

[23] L. Zhang, Y. Guan, Detecting click fraud in pay-per-click streams of online advertising networks, in: Proc. ICDCSW'08, IEEE Computer Society, 2008, pp. 77–84, http://dx.doi.org/10.1109/ICDCS.2008.98.

[24] F. Zhang, X. Hao, J. Chao, S. Yuan, Label propagation-based approach for detecting review spammer groups on e-commerce websites, Know.-Based Syst. 193 (2020) 105520, http://dx.doi.org/10.1016/j.knosys.2020.105520.

[25] F. Zhang, Y. Qu, Y. Xu, S. Wang, Graph embedding-based approach for detecting group shilling attacks in collaborative recommender systems, Know.-Based Syst. 199 (2020) 105984, http://dx.doi.org/10.1016/j.knosys.2020.105984.

[26] W. Zhou, J. Wen, Q. Qu, J. Zeng, T. Cheng, Shilling attack detection for recommender systems based on credibility of group users and rating time series, PLoS One 13 (2018) e0196533, http://dx.doi.org/10.1371/journal.pone.0196533.

[27] P. Hayati, V. Potdar, K. Chai, A. Talevski, Web spambot detection based on Web navigation behaviour, in: Proc. AINA'10, 2010, pp. 797–803, http://dx.doi.org/10.1109/AINA.2010.92.

[28] Z. Chu, S. Gianvecchio, A. Koehl, H. Wang, S. Jajodia, Blog or block: Detecting blog bots through behavioral biometrics, Comput. Netw. 57 (3) (2013) 634–646, http://dx.doi.org/10.1016/j.comnet.2012.10.005.

[29] Y. Wu, Y. Fang, S. Shang, J. Jin, L. Wei, H. Wang, A novel framework for detecting social bots with deep neural networks and active learning, Know.-Based Syst. 211 (2021) 106525, http://dx.doi.org/10.1016/j.knosys.2020.106525.

[30] H. Xu, Z. Li, C. Chu, Y. Chen, Y. Yang, H. Lu, H. Wang, A. Stavrou, Detecting and characterizing web bot traffic in a large e-commerce marketplace, in: J. Lopez, J. Zhou, M. Soriano (Eds.), Computer Security, Springer, Cham, 2018, pp. 143–163, http://dx.doi.org/10.1007/978-3-319-98989-1_8.

[31] C. Walgampaya, M. Kantardzic, Cracking the smart clickbot, in: Proc. IEEE Int. S. Web Syst., WSE'11, 2011, pp. 125–134, http://dx.doi.org/10.1109/WSE.2011.6081830.

[32] E.M. Clark, J.R. Williams, C.A. Jones, R.A. Galbraith, C.M. Danforth, P.S. Dodds, Sifting robotic from organic text: A natural language approach for detecting automation on Twitter, J. Comput. Sci. 16 (2016) 1–7, http://dx.doi.org/10.1016/j.jocs.2015.11.002.

[33] S. Sadiq, Y. Yan, A. Taylor, M.-L. Shyu, S.-C. Chen, D. Feaster, AAFA: Associative affinity factor analysis for bot detection and stance classification in Twitter, in: Proc. IEEE Int. Conf. Information Reuse and Integration, IRI'17, 2017, pp. 356–365, http://dx.doi.org/10.1109/IRI.2017.25.

[34] V. Sharma, R. Kumar, W.-H. Cheng, M. Atiquzzaman, K. Srinivasan, A.Y. Zomaya, NHAD: Neuro-fuzzy based horizontal anomaly detection in online social networks, IEEE Trans. Knowl. Data Eng. 30 (11) (2018) 2171–2184, http://dx.doi.org/10.1109/TKDE.2018.2818163.

[35] A. Lagopoulos, G. Tsoumakas, G. Papadopoulos, Web robot detection: A semantic approach, in: Proc. Int. Conf. Tools with Artificial Intelligence, ICTAI'18, IEEE, 2018, pp. 968–974.

[36] Y. Luo, G. She, P. Cheng, Y. Xiong, Botgraph: Web bot detection based on sitemap, 2019, arXiv:1903.08074.

[37] S. Wan, Y. Li, K. Sun, PathMarker: protecting web contents against inside crawlers, Cybersecurity 2 (2019) 1–17, http://dx.doi.org/10.1186/s42400-019-0023-1.

[38] C. Iliou, T. Kostoulas, T. Tsikrika, V. Katos, S. Vrochidis, Y. Kompatsiaris, Towards a framework for detecting advanced Web bots, in: Proc. Int. Conf. Availability, Reliability and Security, ARES'19, Association for Computing Machinery, New York, NY, USA, 2019, http://dx.doi.org/10.1145/3339252.3339267.

[39] A. Laughter, S. Omari, P. Szczurek, J. Perry, Detection of malicious HTTP requests using header and URL features, in: K. Arai, S. Kapoor, R. Bhatia (Eds.), Future Technologies Conference (FTC'20) II, in: AISC, vol. 1289, Springer, Cham, 2021, pp. 449–468, http://dx.doi.org/10.1007/978-3-030-63089-8_29.

[40] W. Zhu, H. Gao, Z. He, J. Qin, B. Han, A hybrid approach for recognizing Web crawlers, in: E.S. Biagioni, Y. Zheng, S. Cheng (Eds.), Wireless Algorithms, Systems, and Applications, WASA'19, in: LNCS, vol. 11604, Springer, Cham, 2019, pp. 507–519, http://dx.doi.org/10.1007/978-3-030-23597-0_41.

[41] Z. Chu, S. Gianvecchio, H. Wang, Bot or human? A behavior-based online bot detection system, in: P. Samarati, I. Ray, I. Ray (Eds.), From Database To Cyber Security: Essays Dedicated To Sushil Jajodia on the Occasion of His 70th Birthday, in: LNCS, vol. 11170, Springer, Cham, 2018, pp. 432–449, http://dx.doi.org/10.1007/978-3-030-04834-1_21.

[42] R.U. Rahman, D.S. Tomar, New biostatistics features for detecting web bot activity on web applications, Comput. Secur. 97 (2020) 102001, http://dx.doi.org/10.1016/j.cose.2020.102001.

[43] A. Acien, A. Morales, J. Fierrez, R. Vera-Rodriguez, BeCAPTCHA-Mouse: Synthetic mouse trajectories and improved bot detection, 2021, arXiv:2005.00890.

[44] D. Doran, S.S. Gokhale, Web robot detection techniques: Overview and limitations, Data Min. Knowl. Discov. 22 (2011) 183–210, http://dx.doi.org/10.1007/s10618-010-0180-z.

[45] J. Lee, S. Cha, D. Lee, H. Lee, Classification of Web robots: An empirical study based on over one billion requests, Comput. Secur. 28 (8) (2009) 795–802, http://dx.doi.org/10.1016/j.cose.2009.05.004.

[46] M.D. Dikaiakos, A. Stassopoulou, L. Papageorgiou, An investigation of Web crawler behavior: Characterization and metrics, Comput. Commun. 28 (8) (2005) 880–897, http://dx.doi.org/10.1016/j.comcom.2005.01.003.

[47] P.-N. Tan, V. Kumar, Discovery of Web robot sessions based on their navigational patterns, Data Min. Knowl. Discov. 6 (1) (2002) 9–35, http://dx.doi.org/10.1023/A:1013228602957.

[48] G. Suchacka, I. Motyka, Efficiency analysis of resource request patterns in classification of Web robots and humans, in: Proc. Eur. Conf. Modelling and Simulation, ECMS'18, 2018, pp. 475–481, http://dx.doi.org/10.1109/CYBConf.2015.7175961.

[49] S. Kwon, Y.-G. Kim, S. Cha, Web robot detection based on pattern-matching technique, J. Inf. Sci. 38 (2) (2012) 118–126, http://dx.doi.org/10.1177/0165551511435969.

[50] W. Guo, S. Ju, Y. Gu, Web robot detection techniques based on statistics of their requested URL resources, in: Proc. Int. C. Comp. Supp. Coop. (CSCWD'05), 1, 2005, pp. 302–306, http://dx.doi.org/10.1109/CSCWD.2005.194187.

[51] W.-Z. Lu, S.-Z. Yu, Web robot detection based on Hidden Markov Model, in: Proc. Int. C. Commun. Circuit., IC3S'20, 3, 2006, pp. 1806–1810, http://dx.doi.org/10.1109/ICCCAS.2006.285024.

[52] A. Balla, A. Stassopoulou, M.D. Dikaiakos, Real-time Web crawler detection, in: Proc. Int. Conf. Telecommunications, ICT'11, IEEE, 2011, pp. 428–432, http://dx.doi.org/10.1109/CTS.2011.5898963.

[53] T. Gržinić, L. Mršić, J. Šaban, Lino – an intelligent system for detecting malicious Web-robots, in: Intelligent Information and Database Systems, ACIIDS'15, in: LNAI, vol. 9012, Springer International Publishing, Cham, 2015, pp. 559–568, http://dx.doi.org/10.1007/978-3-319-15705-4_54.

[54] S. Kwon, M. Oh, D. Kim, J. Lee, Y.-G. Kim, S. Cha, Web robot detection based on monotonous behavior, in: Proceedings of the Information Science and Industrial Applications, Vol. 4, 2012, pp. 43–48, http://dx.doi.org/10.1109/CSCWD.2005.194187.

[55] R.U. Rahman, D.S. Tomar, Threats of price scraping on e-commerce websites: attack model and its detection using neural network, J. Comput. Virol. Hacking Tech. 17 (2021) 75–89, http://dx.doi.org/10.1007/s11416-020-00368-6.

[56] C.H. Saputra, E. Adi, S. Revina, Comparison of classification algorithms to tell bots and humans apart, J. Next Gener. Inf. Technol. 4 (7) (2013) 23–32.

[57] G. Jacob, E. Kirda, C. Kruegel, G. Vigna, PUBCRAWL: Protecting users and businesses from CRAWLers, in: Proc. USENIX Security Symposium, Security'12, USENIX Association, Berkeley, CA, USA, 2012, p. 25.

[58] G. Suchacka, M. Sobków, Detection of Internet robots using a Bayesian approach, in: Proc. IEEE Int. Conf. Cybernetics, 2015, pp. 365–370, http://dx.doi.org/10.1109/CYBConf.2015.7175961.

[59] S. Rovetta, G. Suchacka, F. Masulli, Bot recognition in a Web store: An approach based on unsupervised learning, J. Netw. Comput. Appl. 157 (2020) 102577, http://dx.doi.org/10.1016/j.jnca.2020.102577.

[60] G. Suchacka, J. Iwański, Identifying legitimate Web users and bots with different traffic profiles – an Information Bottleneck approach, Know.-Based Syst. 197 (2020) 105875, http://dx.doi.org/10.1016/j.knosys.2020.105875.

[61] S. Alam, G. Dobbie, Y.S. Koh, P. Riddle, Web bots detection using Particle Swarm Optimization based clustering, in: Proc. IEEE C. Evol. Computat., CEC'14, IEEE, 2014, pp. 2955–2962, http://dx.doi.org/10.1109/CEC.2014.6900644.

[62] M. Zabihi, M.V. Jahan, J. Hamidzadeh, A density based clustering approach to distinguish between web robot and human requests to a Web server, ISC Int. J. Inf. Secur. 6 (1) (2014) 77–89, http://dx.doi.org/10.1109/ICCKE.2014.6993362.

[63] J. Hamidzadeh, M. Zabihimayvan, R. Sadeghi, Detection of Web site visitors based on fuzzy rough sets, Soft Comput. 22 (7) (2018) 2175–2188, http://dx.doi.org/10.1007/s00500-016-2476-4.

[64] D. Stevanovic, N. Vlajic, A. An, Detection of malicious and non-malicious website visitors using unsupervised neural network learning, Appl. Soft Comput. 13 (1) (2013) 698–708, http://dx.doi.org/10.1016/j.asoc.2012.08.028.

[65] M. Zabihimayvan, R. Sadeghi, H.N. Rude, D. Doran, A soft computing approach for benign and malicious web robot detection, Expert Syst. Appl. 87 (2017) 129–140, http://dx.doi.org/10.1016/j.eswa.2017.06.004.

[66] S. Rovetta, G. Suchacka, A. Cabri, F. Masulli, Feature selection: a multi-objective stochastic optimization approach, in: Proc. Int. Conf. Optimization and Applications, ICOA'20, IEEE, 2020, pp. 1–5, http://dx.doi.org/10.1109/ICOA49421.2020.9094478.

[67] A. Wald, Sequential tests of statistical hypotheses, Ann. Math. Stat. 16 (2) (1945) 117–186, http://dx.doi.org/10.1214/aoms/1177731118.

[68] C.K. Chow, An optimum character recognition system using decision functions, IRE Trans. Electron. Comput. 6 (4) (1957) 247–254, http://dx.doi.org/10.1109/TEC.1957.5222035.

[69] K. Miettinen, Nonlinear Multiobjective Optimization, Kluwer Academic Publishers, Boston, 1999.

[70] C. Saule, R. Giegerich, Pareto optimization in algebraic dynamic programming, Algorithms Mol. Biol. 10 (1) (2015) 22, http://dx.doi.org/10.1186/s13015-015-0051-7.

[71] Udger, 2017, URL https://udger.com (access date: 4 September 2017).

[72] User-agents, 2017, https://http://www.user-agents.org (access date: 4 September 2017).

[73] B. Hidasi, A. Karatzoglou, L. Baltrunas, D. Tikk, Session-based recommendations with recurrent neural networks, in: Proc. Int. Conf. Learning Representations, ICLR'16, 2016.