Discovery of Web Robot Sessions based on their Navigational Patterns *

Pang-Ning Tan
Department of Computer Science,
University of Minnesota,
200 Union Street SE,
Minneapolis, MN 55455

Vipin Kumar
Department of Computer Science,
University of Minnesota,
200 Union Street SE,
Minneapolis, MN 55455

March 6, 2001

Abstract

In recent years, it is becoming increasingly difficult to ignore the impact Web robots have on both commercial and research institutional Web sites. In particular, e-commerce retailers are concerned about the unauthorized deployment of robots for gathering business intelligence at their Web sites. Web robots also tend to consume considerable network bandwidth at the expense of other users. Sessions due to Web robots are making it more difficult to perform clickstream analysis effectively on the Web data. Thus, it is crucial to identify visits by Web robots and distinguish them from other accesses. Conventional techniques for detecting Web robot sessions are often based on the User Agent and IP Address of clients. These techniques are not sufficient for detecting previously unidentified robots. In this paper, we propose a solution to this problem by detecting Web robots based on the characteristics of their access patterns. Our experimental results showed that highly accurate robot classification models can be obtained using these access features. We have used our models to isolate mislabeled sessions and found that most of the mislabeling are due to camouflaging and previously unidentified robots.

1 Introduction

Web robots are software programs or agents that automatically traverse the hyperlink structure of the World Wide Web in order to locate and retrieve information from the Internet. The emergence of the World Wide Web as an information dissemination medium,

*This work was supported by NSF ACI−9982274 and by Army High Performance Computing Research Center contract number DAAH04−95−C−0008. Access to computing facilities was provided by AHPCRC, Minnesota Supercomputer Institute.

along with the availability of many Web robot authoring tools have resulted in the rapid proliferation of Web robots unleashed on the Internet today. These robots are sent out to scour the Web for various purposes. For instance, they can be used to collect statistics about the structure of the World Wide Web [10]. Internet search engines such as Google [9] and Altavista [1] rely on the documents retrieved by Web robots to build their index databases. Web administrators employ Web robots to perform site maintenance tasks such as mirroring and checking for broken hyperlinks. Web robots are also used to collect email addresses and online resumes, monitor product prices and corporate news, etc. The widespread deployment of robots has made it important to understand the impact of Web robot visits to any given Web site.

There are many situations in which it is desirable to identify visits by Web robots and distinguish them from other users. Firstly, e-commerce retailers are particularly concerned about unauthorized deployment of Web robots, which are used for gathering business intelligence at their sites. In such a situation, the e-commerce site may want to stop responding to HTTP requests coming from the unauthorized robot. For example, eBay filed a lawsuit against an auction aggregator site last year for using unauthorized shopbots to retrieve auction information from their Web site.[1]

Secondly, many of the e-commerce Web sites perform Web traffic analysis in order to infer the demographic and browsing behavior of their site visitors. Unfortunately, such analysis can be severely distorted by the presence of Web robots. For example, Figure 1 shows the total number of sessions and HTML pages requested at the University of Minnesota Computer Science department Web site between the period of January 1, 2001 and January 31, 2001. On average, about 5% of the total sessions are due to visits by Web robots. However, Web robot sessions may account for as many as 85% of the total number of HTML pages requested. If these robot sessions are not identified and eliminated, an analyst may end up making the wrong inferences about his/her site visitors.

Thirdly, the deployment of Web robots usually comes at the expense of other users because they often consume considerable network and server resources. Poorly-designed robots may tie up these resources and overload the Web server. In this situation, it will be desirable to detect the disruptive robots and reduce their priority of service immediately.

Fourthly, Web robot accesses could be indicative of fraudulent behavior. For example,

[1] An auction aggregator combines information from various on-line auction sites and list the integrated results at their own Web site. As a result, consumers using an aggregator site can buy products from sellers who posted their auctions at another auction site without ever visiting the auction site. This is of great concern to many auction site operators because consumers and sellers may stop visiting their Web site and use the services of aggregator sites instead.

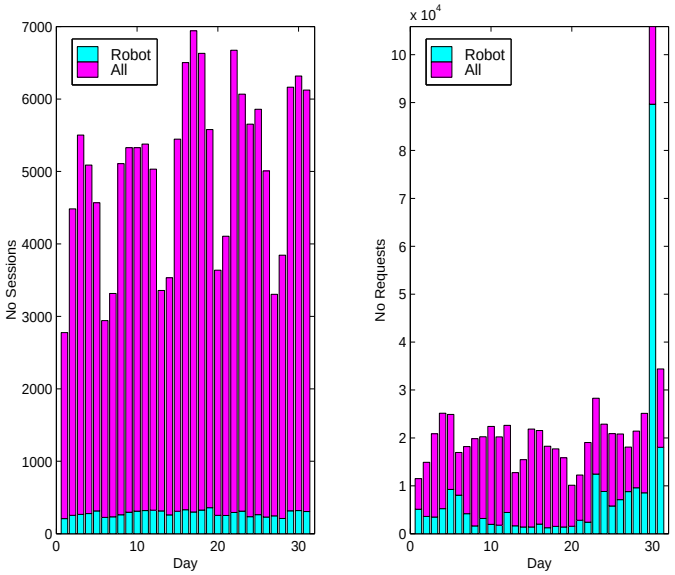


Figure 1: Total number of sessions and HTML requests due to robot sessions (compare to the overall number of sessions and HTML requests). The anomaly on day 30 is due to sessions by a site mapping robot called linbot.

there are many click-through payment programs established on the Web, in which an advertiser (i.e. the target site) would reward the referring Web site for every visitor who reach the target site by clicking on the referrer's advertisement banner. Such a payment scheme can be easily abused by unscrupulous referrer site owners who use Web robots to inflate the clickthrough rate. Thus, detection of Web robot sessions is absolutely necessary to protect the target site owner from such malpractice.

Even though Web robot detection is a widely recognized problem, there are very few published papers in this area. A standard way to identify robots is by examining the client's identity in the HTTP request messages sent to a Web server [30, 24]. By comparing the IP Address and User Agent fields of the request messages against those of known robots[2], accesses by many of the well-known robots can be detected. Unfortunately, since Web robots can be easily constructed and deployed, it has become almost impossible to keep a comprehensive database of all robots. This problem is exacerbated by robots that attempt to disguise their identities by declaring their User Agents to be similar to conventional Web browsers such as Netscape or Microsoft Internet Explorer (e.g. the last entry in Table 1). Thus, standard techniques may fail to detect the presence of such robots.

[2]Currently, there are various Web robot repositories available on the Internet. These repositories maintain a list of User Agents and/or IP addresses of known robots. The most popular one is the Web Robots Database at http://info.webcrawler.com/mak/projects/robots/active/.

Table 1: List of IP Addresses and User Agents for several Web clients.

Client's Type	IP Address	User Agent
Browser (Netscape)	160.94.178.152	Mozilla/4.7 [en] (X11; I; Linux 2.2.14-5.0 i686)
Browser (Microsoft IE)	160.94.178.205	Mozilla/4.0 (compatible; MSIE 5.01; Windows NT)
Browser (Opera)	160.94.103.248	Opera/5.01 (Windows NT & Opera 5.0; U) [en]
Search Engine	64.208.37.53	Googlebot/2.1 (+http://www.googlebot.com/bot.html)
Email Harvester	4.41.77.204	EmailSiphon
Offline Browser	24.43.172.37	Teleport Pro/1.29
Link Checker	204.94.209.1	LinkScan/6.1b Unix http://www.elsop.com/
Search Engine (looksmart.com)	207.138.42.10	Mozilla/4.5 [en] (Win95; I)

In this paper, we offer a potential solution by detecting Web robots according to their navigational behavior. Our main assumption is that Web robots traversing a Web site with the same information need will exhibit similar access behavior, regardless of the identities they present to the Web server. Our goal is to build classification models that will distinguish robot from non-robot sessions. Since a Web client's behavior is dynamic in nature, our classification models must be able to capture the temporal changes of the navigational patterns.

The main contributions of the paper are as follows:

1. We present an analysis of the navigational behavior for various types of Web robots and show empirically that such behavior depends on their navigational goals.

2. We propose a robust session identification technique to preprocess the Web server logs. This technique can identify sessions having multiple IP addresses (e.g. accesses by AOL users).

3. We show that highly accurate robot classification models can be induced using the navigational features of Web clients.

4. We present a technique for identifying mislabeled training and validation samples. This technique can be used to detect both camouflaging and previously unknown Web robots.

The rest of the paper is organized as follows. In Section 2, we present an overview of the Web robot detection problem and discuss some of the techniques used to solve the problem. Section 3 describes the preprocessing steps needed to convert the raw click-stream data

Table 2: Some of the common Web Robots and their typical characteristics.

Client's Type	Examples	Navigational Goals	Characteristics
Search Engine Robots	T-Rex[18], Scooter[1]	maximize coverage of a Web site	Breadth first search, Unassigned referrer
Offline Browser	Teleport Pro[25], Offline Explorer[19]	download entire or portion of Web site to local disk	varied behavior
Email Collector	EmailDigger[7], Extractor Pro[8]	maximize coverage of home pages	unassigned referrer, ignore image files
Link Checker	LinkScan[17], Xenu's Link Sleuth[28]	check for broken links	use HEAD request method, unassigned referrer

into server sessions. A discussion about how to derive the session features and class label is also presented. This is followed by our experimental results in Sect. 4. Finally, Section 5 concludes with suggestions for future work.

2 Web Robot Detection: Overview

2.1 Characteristics of Web Robots

Before presenting the various techniques for Web robot detection, it is important to know what are the different types of Web robots that are available today (Table 2). This is because each type of robot may exhibit different characteristics based on the goal of their navigation. Knowing the navigational goals of these robots can help us to identify the set of relevant features for predicting robot sessions.

Eichman [6] divides Web robots into two distinct categories: (1) agents that are designed to accomplish a specific task (such as browsing assistants and hyperlink checkers), and (2) agents that are used to build information bases (such as email collectors and search engine robots).

The goal of an Internet search engine is to index the Web pages of all the Web sites. Search engine robots are deployed with the goal of maximizing their coverage of a particular Web site. As a result, they tend to use a breadth-first Web-retrieval strategy or parallel retrieval in order to speed up their operations. Most HTTP requests coming from popular search engine robots do not assign any values to their referrer fields. The referrer field is provided by the HTTP protocol to allow a Web client (particularly, a Web browser) to specify the address of the Web page that contains the link the client followed in order to reach the current requested page. For example, a user, who wants to access

the page http://www.xyz.com/A.html by clicking on a link found at http://www.xyz.com, causes the Web browser to generate an HTTP request with the referrer value equals to http://www.xyz.com. Most search engine robots do not care about assigning a value to their referrer fields. As a result, the referrer fields due to these robot accesses appear as "-" in the Web server logs.

Link checkers [28, 17] are utility programs that are designed to assist Web site administrators in checking for broken hyperlinks and missing pages. Many link checkers would send a special type of HTTP request message (called a HEAD request type) to determine the validity of a hyperlink. A Web server responds to a HEAD request by sending an HTTP response header, which contains a status code indicating whether the request has succeeded or failed. The response to a HEAD request message does not involve a transfer of the requested file, unlike the typical GET request message from Web browsers. Note that Web browsers can also send HEAD request messages to validate the recency of a cached HTML page.

Web robots are also designed for various other reasons. For example, email collectors [8, 7] are robots that automatically collect email addresses posted on the Web. These robots tend to retrieve HTML pages only, and ignore image and other file formats. Offline browsers are either stand-alone browsers or add-on utilities that allow a Web user to download an entire Web site (or portion of it) to a local directory for offline viewing [27, 25]. The characteristics of these robots vary, depending on their navigational goals. For instance, offline browsers that download an entire Web site behaves similarly to search engine robots, while those that download a small portion of the Web site (for pre-caching purposes) resemble the characteristics of human users.

Table 2 summarizes the characteristics and navigational goals of several types of Web robots. Other types of Web robots include personal browsing assistants [2, 16], shopbots [11, 3], resume hunters and other special-purposed software agents.

2.2 Common Robot Detection Techniques

In this section, we will present some of the common techniques used to identify Web robot sessions:

1. **By examining sessions that access a specially-formatted file called robots.txt**

   The Robot Exclusion Standard [13, 22] was proposed to allow Web administrators to specify which part of their site is off-limits to visiting robots, by using a specially-formatted file called robots.txt. According to this Standard, whenever a robot visits a Web site, say at http://www.xyz.com/, it should first look for a file called

http://www.xyz.com/robots.txt. This file contains a list of access restrictions specified by the administrator. For example, the following entry in robots.txt forbids all robots from accessing the file http://www.xyz.com/private.html.

```
User-agent: *
Disallow: /private.html
```

Hence, Web robot visits can be inferred from sessions that access the robots.txt file. This is a reasonably good heuristic because most Web sites do not provide a hyperlink from any of its other pages to this file. Therefore, normal users are seldom aware of the existence of this file. However, one can not rely solely on this criteria because compliance to the Robot Exclusion standard is voluntary, and many robots do not follow this standard.

2. **By examining the User Agent field of HTTP request messages from Web clients**

   It is commonly agreed that poor implementation of Web robots can lead to serious network and server overload problems. Thus, a protocol is needed to provide guidance to appropriate robot behavior. Eichman [6] and Koster [14, 12] have proposed several ethical guidelines for Web robot developers. The purpose of these guidelines is to ensure that both the Web robot and Web server can cooperate with each other in a way that will be beneficiary for both parties. Under these guidelines, a cooperative robot should declare its identity to a Web server via its User Agent field. For instance, the User Agent field for many of the well-known browsers often contain the string "Mozilla". Figure 2 illustrates such an example where an Internet Explorer browser, identified by its user agent Mozilla/4.0 (compatible; MSIE 5.01), was used to request for the HTML page, http://www.xyz.com/A.html. In practice, there are many exceptions to this rule. Some robots (and browsers) would use multiple User Agent fields within the same session. For example, an offline browser called Teleport Pro has an empty User Agent field when accessing the robots.txt file, but uses "Teleport Pro" when downloading other documents. Even standard Web browsers may issue requests with multiple User Agents. For instance, when plugins are used by the Microsoft Internet Explorer browser to download certain types of documents, such as PDF files, an additional HTTP request is generated with a User Agent field called "contype"; producing the following entries in the Web log:

```
203.94.250.186 - - [01/Jan/2001:15:18:04 -0600] "GET /grad-info/finapp.pdf
```

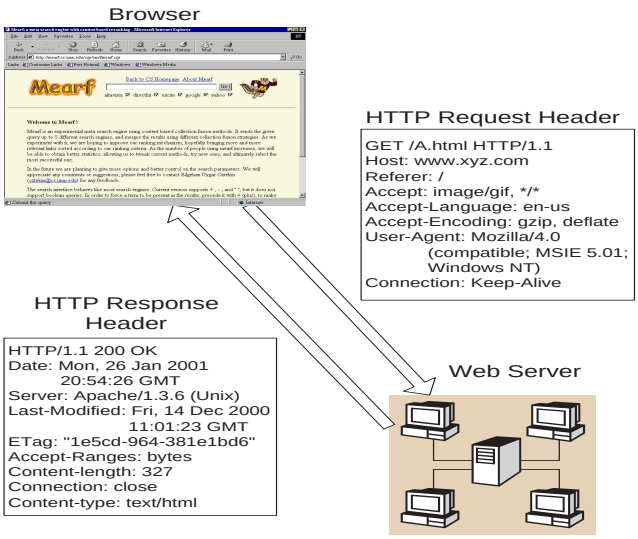


Figure 2: HTTP request and response header messages.

```
HTTP/1.1" 200 3993 "http://www.cs.umn.edu/grad-info/" "Mozilla/4.0 (compatible;
MSIE 5.01; Windows 98; bplnet-100)"

203.94.250.186 - - [01/Jan/2001:15:18:08 -0600] "GET /grad-info/finapp.pdf
HTTP/1.1" 200 3993 "-" "contype"
```

The problem becomes more complicated when robot designers attempt to disguise their identities by using the same User Agent information as standard browsers. In such a situation, detecting Web robots using the User Agent field is a hopeless solution. Similarly, the presence of anonymizer Web sites can disguise the appearance of Web users by changing the User Agent field of a browser to robot-like values such as "SilentSurf" [3] and "Turing Machine" [4].

3. **By matching the IP address of sessions with those of known robot clients**

   Various Web sites have begun to compile a list of IP addresses (and User Agents) for known Web robots. However, such a list is often incomplete because it is infeasible to obtain a comprehensive listing of all robots. Furthermore, the same IP Address can be used by both humans, to browse the Web, and robots, to automatically download some files. This approach also fails to detect camouflaging and previously unknown robots. Alternatively, one can examine only the top visiting IP addresses of clients and verify the origin of each client. Unfortunately, this technique often discovers only robots that are already well-known. Some robots use multiple IP

[3] Anonymizer Web site at http://www.noproxy.com.

[4] Anonymizer Web site at http://www.free.anonymizer.com.

addresses to parallelize their Web document retrieval. This complicates both the session identification and robot detection problem. For example, a robot may access the robots.txt file using one of its available IP addresses and fetches other documents using the rest of the IP addresses. If accesses by these IP addresses are not identified to be the same session, one can potentially lose information about the actual traversal path of the robot.

4. **By examining sessions with an unusually large number of HEAD requests or HTTP requests with unassigned referrer fields**

   The guidelines for Web robot designers suggest that ethical robots should help to reduce the burden on Web servers by using a low retrieval rate and the HEAD request method, whenever possible, or by operating only when the server is lightly loaded (e.g. at night). Therefore, one can examine sessions with a large number HEAD requests to discover potential robots. Another reasonable heuristic is to look for sessions having large number of requests with unassigned referrer fields. Most robots (except for offline browsers and some utility programs such as Wget) do not assign any value to this field in their HTTP request messages. Nevertheless, these two heuristics are not entirely reliable because Web browsers can sometimes generate both HEAD request types (to check the validity of a cached page) and HTTP messages with unassigned referrer values (e.g. when a user clicks on a bookmarked page or type in a new URL in the address window).

2.3 Proposed Robot Detection Technique

The previous discussion suggests that a more robust technique is needed to identify visits by camouflaging or previously unknown Web robots. In this paper, we propose to build a classification model to identify robot sessions. This work is based on the assumption that the navigational behavior of Web robots is distinct from the navigational behavior of human users. In this paper, the navigational behavior of a Web client is characterized in terms of what are the different types of pages being requested, how long is the session or time between requests, what is the coverage of the Web site, etc.

Figure 3 shows a graphical comparison of the characteristics for several known robots in contrast to those of human users. The width and depth parameters are used to infer the search strategy employed by the Web client. For instance, search engine robots tend to have large traversal width and shallow depth, indicative of a breadth-first behavior. Discussion about how the width and depth parameters are computed will be given in the next section. Note the agreement between the observed characteristics of robots and their

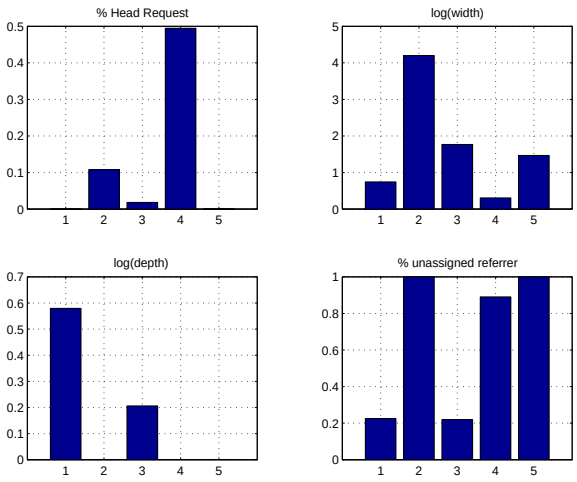


Figure 3: Comparison of navigational patterns for serveral known Web clients: (1) A group of users from Computer Science department at University of Minnesota (2) Search engine robots (3) Offline browsers (4) Link checkers (5) Email collectors. Note that the width and depth parameters are plotted on a logarithmic scale.

navigational goals as specified in Table 2. For example, most of the search engine robots, link checkers and email collectors do not assign any values to their referrer fields. Offline browsers have very similar characteristics as human users, in terms of the rate of HEAD requests and unassigned referrer fields. Nevertheless, it is possible to distinguish the two clients based on the width and depth of their access patterns.

The results of Figure 3 show that different Web clients collectively exhibit different access characteristics, depending on their navigational goals. This suggests that it could be possible to construct reasonably accurate classification models to detect the presence of Web robots based on their navigational features. This is exactly the approach taken in this paper.

3 Methodology

3.1 Data Source and Preprocessing

Web server logs are the data source used in our experiments. A typical Web log entry contains information such as the IP address of the client, the date and time a request is made, the request method and protocol used, the URI of the requested page, the status code of the response message, the size of the document transferred, the referrer page and its User Agent information.

During preprocessing, the log entries are grouped into server sessions using a variation of the session identification heuristic proposed in [4]. Unlike [4], our approach is capable of identifying sessions having multiple IP Addresses or User Agents. The session identification technique will be described in the Appendix.

3.2 Feature Vector Construction

Once the server sessions are created, the next step is to construct a feature vector to represent each session. Table 3 presents a summary of attributes that can be derived from the server sessions. Each session can be broken up into several episodes. In our analysis, the events of interest within a session are the requests for HTML pages. Thus, each episode is associated with a tuple, (p_i, p_j), where p_i and p_j are the requested and referred HTML pages[5]. The computation of temporal attributes such as totalTime, AvgTime and stdevTime is illustrated in Fig. 4. The totalTime attribute is approximated by the interval between the first and last log entry of the session. On the other hand, avgTime and stdevTime is computed using the intervals between successive episodes in the session.

The width and depth attributes are computed by constructing a graph representing all episodes within a session. For example, if a session contains the following episodes, { (/A,-), (/A/B,/A), (/A/B/C,/A/B)}, then its width will be 1 and its depth will be 3. Basically, the width attribute measures the number of leaf nodes generated in the graph while the depth attribute measures the maximum depth of the tree(s) within the graph. Therefore, a session that contains requests for {(/A,-) (/A/B,/A), (/C,-) (/D,-) } will have a width of 3 and a depth of 2. Sessions without HTML requests, denoted as {(-,-)}, are assumed to have depth and width equal to 1.

MultiIP and *MultiAgent* are binary flags to indicate whether a session contains log entries with multiple IP addresses and User Agents. Attributes 2 to 10 correspond to the various types of files requested, whereas attributes 17 to 20 measure the various request methods used during a particular session. The *Night* attribute is used to determine if the session made at least one request between 12am and 7am (local server time). The *Repeated* attribute computes the percentage of non-unique page requests within a session. For instance, if the total number of pages requested in a session is 10 and the total number of unique pages is 4, then the *Repeated* value is $(10-4)/10 = 0.6$. The *Error* attribute computes the rate of unsuccessful requests made within the session. The rest of the attributes

[5]Note that our definition of an episode is different from the terminology adopted by the W3C committee. Also, a session that do not contain any HTML requests will have a single episode, $(-,-)$, associated with its last log entry.

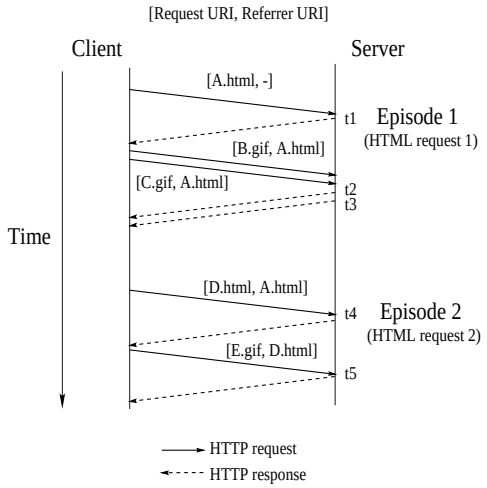


Figure 4: This session contains two episodes. $t1$, $t2$, $t3$, $t4$ and $t5$ are the timestamps recorded in the server logs. The approximated total time of the session is $t5 - t1$ while the period between the two episodes is $t4 - t1$.

are self-explanatory.

3.3 Session Labeling

The assignment of class label to a session is based on the following heuristics:

1. If a session contains a request for the robots.txt file, then the session is identified as a robot session (denoted by Class = 1).

2. If the agent field of a session corresponds to the agent of a known or suspected Web robot, then it will be assigned the value Class = 1. In our work, we have divided the User Agents into 4 distinct categories: Type 1 (known robots), Type 2 (known browsers), Type 3 (possible robots) and Type 4 (possible browsers or plugins used to retrieve files of specific formats). The partitioning of User Agents according to their categories is done semi-automatically. We manually verify the identity of some of the unknown agents by doing a Web search on the unidentified User Agent. Type 3 and Type 4 agents contain names that would suggest they are likely to be robots and browsers, respectively. Some examples of these agents are shown in Table 4. In our experiments, we assign sessions with Type 4 agents as browsers (Class = 0), while those with Type 3 agents as robots (Class = 1).

There are other heuristics we can use to supplement the above labeling scheme. For example, if all the requests are made using the HEAD method, then the session is most

Table 3: Summary of attributes derived from server sessions. The attributes are used for class labeling (denoted as Classify) or constructing the feature vector representation (Feature).

Id	Attribute Name	Remark	Purpose
1	totalPages	Total number of pages requested.	Feature
2	% Image	% of image pages (.gif/.jpg) requested.	Feature
3	% Binary Doc	% of binary documents (.ps/.pdf) requested.	Feature
4	% Binary Exec	% binary program files (.cgi/.exe/.class) requested.	Feature
5	robots.txt	binary; indicates whether robots.txt file is requested	Classify
6	% HTML	% of HTML pages requested.	Feature
7	% Ascii	% of Ascii files (.txt/.c/.java) requested.	Feature
8	% Zip	% of compressed files (.zip/.gz) requested.	Feature
9	% Multimedia	% of multimedia files (.wav/.mpg) requested.	Feature
10	% Other	% of other file formats requested.	Feature
11	$totalTime$	Server session length (Fig. 4).	Feature
12	$avgTime$	Average time between episodes (Fig. 4).	Feature
13	$stdevTime$	Standard deviation of time between episodes (Fig. 4).	Feature
14	$Night$	binary; for requests made between 12am and 7am (local time).	Feature
15	$Repeated$	Reoccurence rate of file requests.	Feature
16	$Error$	% of requests with status $\geq$ 400.	Feature
17	GET	% of requests made with GET method.	Feature
18	POST	% of requests made with POST method.	Feature
19	HEAD	% of page requests made with HEAD method.	Classify
20	OTHER	% of requests made with other methods.	Feature
21	$width$	width of the traversal (in the URL space).	Feature
22	$depth$	depth of the traversal (in the URL space).	Feature
23	$length$	Session length (total no of episodes).	Ignore
24	referrer = "-"	% of requests with unassigned referrer	Classify
25	$MultiIP$	binary; indicates whether session contains multiple IP	Feature
26	$MultiAgent$	binary; indicates whether session contains multiple agents	Feature

Table 4: Examples of different User Agent types.

User Agent	Agent Type
ArchitextSpider	Type 1
Mozilla/4.0 (compatible; MuscatFerret/2.0; http://www.webtop.com/)	Type 1
Mozilla/4.0 (compatible; MSIE 4.01; Windows NT)	Type 2
Mozilla/4.0 (compatible; MSIE 5.0; AOL 6.0; Windows 98; DigExt)	Type 2
Mozilla/4.0 (compatible; MSIE 5.0; Windows 98) Opera 5.01 [en]	Type 2
Lynx/2.8.3rel.1 libwww-FM/2.14 SSL-MM/1.4.1 OpenSSL/0.9.6	Type 2
www4mail/2.4 libwww-FM/2.14 (Unix; I)	Type 3
unknown/1.0	Type 3
contype	Type 4
Windows-Media-Player/7.00.00.1956	Type 4

likely created by a link checker robot. Another heuristic could be based on the referrer field of the session. If a Web client does not assign a referrer value to any of its requests, then there is a strong possibility that the Web client is a robot, as long as the number of requests is large. If number of requests is small, the session is more likely created by a Web user. This is because a Web browser does not have a referrer value when a user submits a URI from the address window or clicks on a bookmark entry (these are known as user-input clicks). For long sessions, the likelihood that a Web browser generates only user-input clicks are minimal. By selecting an appropriate threshold on the minimum number of requests, one can potentially identify new robot sessions.

A summary of the session labeling algorithm is shown in Table 5. First, the algorithm would find all types of User Agents that appear in a given session. Sessions that contain only a single agent type will be identified as robots if their User Agents are of Type 1 or 3, and non-robots otherwise (line 6). A labeling scheme that favors non-robots is used to handle sessions with more than one agent type (lines 10 to 12). There are two reasons for using such a labeling scheme. Firstly, it was observed that the majority of the multi-agent type sessions contain either combinations of Type 2 and Type 3 agents, or Type 3 and Type 4 agents. These sessions are due to users who invoke a helper application while browsing the Web as illustrated in the following example:[6]

```
155.239.194.112 - - [01/Jan/2001:14:38:37 -0600] "GET /~mein/blender/plugins/
HTTP/1.1" 200 1562 "http://www.rash.f2s.com/links.htm"
"Mozilla/4.0 (compatible; MSIE 5.0; Windows 98)"
```

[6]Go!Zilla is a download manager that allows a user to recover from any failed downloads.

Table 5: Session Labeling Algorithm.

```
Labeling Algorithm (H: array of sessions, t: length threshold) {
1.  for each session s ∈ H do
2.     if s contains a request for robots.txt then s.Class = 1
3.     Let Agents = getUserAgent(s)
4.     Let AgentTypes = getAgentTypes(Agents)
5.     if s.MultiAgent = 1 then
6.         if Type1 ∈ AgentTypes or Type3 ∈ AgentTypes
7.             then s.Class = 1
8.             else s.Class = 0
9.    else
10.        if (Type2 ∈ AgentTypes or Type4 ∈ AgentTypes)
11.            then s.Class = 0
12.            else s.Class = 1
13.    if Class(s) = 0 and s.length > t then
14.        if s.HEAD = 100% or s.(referrer="-") = 100%
15.            then s.Class = 1
16. end;
```

```
155.239.194.112 - - [01/Jan/2001:14:43:34 -0600] "GET /~mein/blender/plugins/plugins.zip
HTTP/1.1" 206 626775 "http://www-users.cs.umn.edu/~mein/blender/plugins/"
"Go!Zilla 3.5 (www.gozilla.com)"
```

Secondly, there are a few multi-agent type sessions with combinations of Type 1 and Type 2 agents. These sessions contain a browser-like Type 2 agent as well as a robot-like Type 1 agent such as "Java1.1"[7]. Further analysis revealed that such sessions are created by Web browsers accessing HTML pages containing Java applets. This explains the rationale for choosing a labeling scheme that favors non-robots. Finally, for sessions that are still classified as non-robots, we use the HEAD and referrer tests to verify the correctness of their class labels (line 14).

3.4 Classification

Once the set of relevant features have been identified, classification models are built using the well-known C4.5 decision tree algorithm [21]. There are two main classification objectives we would like to achieve : (1) to find a good predictive model for detecting Web

[7]This User Agent is often associated with the various Java-based agents crawling our Web site. This is why it is initially categorized as a Type 1 agent.

Table 6: Summary of Data set.

G1 : contains samples with Type 1 (known Robot) and Type 2 (known Browser) User Agents. G2 : contains samples with Type 3 (possible Robot) and Type 4 (possible Browser) User Agents

Experiment	Description
E0	Both training and test data sets contain only G1 samples.
E1	Both training and test data sets contain G1 and G2 samples.

robots based upon their access patterns, and (2) to determine the minimum number of episodes (HTML requests) needed to produce reasonably accurate models.

The overall data set is partitioned into two groups:

1. G1 (clean data), which contains all samples of Type 1 and Type 2 User Agents; and
2. G2 (noisy data), which contains all samples of Type 3 and Type 4 User Agents.

Our classification models can be built using samples from G1 (E0), or mixture of G1 and G2 (E1). A summary of the properties of the different data sets is given in Table 6.

There are various metrics we can use to evaluate the classifier performance. Accuracy is a reasonable metric, as long as the data set remains evenly distributed (between robots and non-robots). Otherwise, we need to compensate the imbalanced class distribution via stratification, or use other meta-learning techniques such as bagging and boosting. In the area of information retrieval, recall and precision are two popular metrics used to evaluate binary classifiers :

$$\text{recall}, r = \frac{\text{no of robot sessions found correctly}}{\text{total no of actual robot sessions}} \quad (1)$$

$$\text{precision}, p = \frac{\text{no of robot sessions found correctly}}{\text{total no of predicted robot sessions}} \ . \quad (2)$$

A classifier that assigns the value 1 to every session will have perfect recall but poor precision. In practice, the two metrics are often summarized into a single value, called the F_1-measure [26] :

$$F_1 = \frac{2rp}{(r+p)} \ . \quad (3)$$

This value is maximized when r and p are close to each other. Otherwise, the value of F_1-measure is dominated by the smaller of r and p [29].

3.5 Identifying Mislabeled Sessions

Despite our concerted effort, some robot sessions are still wrongly labeled. These are mostly robots that have the same User Agent field as Web browsers. In this section, we present an ensemble technique for identifying the mislabeled sessions. Basically, this technique assigns a score to each sample, predicting the likelihood of the sample being mislabeled.

The technique uses the C4.5 classification models built from all the attributes described in Table 3. This may include attributes that are used to determine the class label of the session (robots.txt, HEAD request, etc). Since C4.5 uses a pessimistic pruning strategy to avoid overfitting, the leaf nodes of the decision tree it produces contain a probability distribution for each class. We denote these probabilities as $P(0|X,m)$ and $P(1|X,m)$, where $P(i|X,m)$ is the probability that a sample X belongs to class i according to classifier C_m.

Suppose there are k classifiers, $C_1, C_2, \cdots, C_k$, built from the training samples. Let $t(X)$ be the true class of sample X according to our labeling heuristics, while $c(X,m)$ is the predicted class label assigned by classifier C_m. Furthermore, let $A(m)$ denotes the accuracy of classifier C_m.

Using the above definitions, for each sample X and classifier C_m, we define a false positive $FP(X|C_m)$ or false negative $FN(X|C_m)$ score according to the following formulas:

$$FP(X|C_m) = \begin{cases} 0 & \text{if } t(X) = c(X,m), \\ A(m) \times |P(c(X,m)|X,m) - P(t(X)|X,m| & \text{if } t(X) \neq c(X,m) \text{ and } t(X) = 0. \end{cases} \quad (4)$$

$$FN(X) = \begin{cases} 0 & \text{if } t(X) = c(X,m), \\ A(m) \times |P(c(X,m)|X,m) - P(t(X)|X,m)| & \text{if } t(X) \neq c(X,m) \text{ and } t(X) = 1. \end{cases} \quad (5)$$

The overall false positive or false negative score of a sample, X, is given by $FP(X) = \sum_{m=1}^{k} FP(X|C_m)$ and $FN(X) = \sum_{m=1}^{k} FN(X|C_m)$. High $FP(X)$ scores indicate that these sessions are currently assigned as non-robots, but the classification models suggest that they are very likely to be robots. By examining the log entries for these sessions, one can verify whether the sessions are indeed non-robots or are mislabeled by the session labeling heuristics. Later, we will show that many of the high FP-score sessions are indeed mislabeled sessions due to camouflaging robots.

Sessions with high FN-score are most likely due to offline browsers and Type 3 robots. As shown in Figure 3, the characteristics of an offline browser may resemble that of human users. This explains why they are often mistakenly identified as non-robots. Type 3 robots are mostly utilities that are used to download files from the Web. Sessions due to these robots are often very short, thus making it difficult for our classifiers to distinguish them from other non-robot accesses.

4 Experimental Evaluation

4.1 Experimental Data Set

Our experiments were performed on the University of Minnesota Computer Science department server logs collected from January 1st to January 31st, 2001. We have consolidated the logs from the two main Computer Science department servers, http://www.cs.umn.edu (main server) and http://www-users.cs.umn.edu. Log entries that correspond to redirection requests from the main server to the other are also removed to eliminate duplicate entries. The consolidated Web logs contain a total of 1,639,119 entries. After preprocessing, 180,602 sessions are created; with different proportions of agent types as shown in Table 7. Class labels are assigned to every session according to the session labeling heuristic described in Section 3.3 (with threshold $t = 100$).

Each session is then converted into a feature vector representation and broken up into several episodes (i.e. HTML requests). A data set is created for each episode in the following way. The data set for one episode is generated from all the sessions because each session has at least one episode[8]. Sessions with more than one episode will be truncated by computing their feature values up to the first HTML request. For dataset with two episodes, we ignore all single episode sessions, and consider only those sessions with at least two episodes. Again, sessions with more than two episodes are truncated. This procedure is repeated up to sessions of length 7 (i.e. sessions having at least 7 episodes) as shown in Table 7.

The training and test sets are created by randomly sampling into each data set. In order to account for the unequal sizes of robot and non-robot sessions, we stratify the training and test samples such that both robots and non-robot sessions have equal representation. Stratification can be done by oversampling (E0 and E1) or undersampling (E3 and E4) the overall population. For instance, suppose the original data set contains 200 robot and 2000 non-robot sessions. We first divide both the robot and non-robot sessions equally between the training and test sets. As a result, both the training and test sets contain 100 robots and 1000 non-robots. For stratification by oversampling, each robot session is duplicated 10 times to ensure that both classes are equally represented during model building. In practice, the ratio of non-robot sessions to robot sessions can be quite large. Replicating the robot sessions by a factor larger than 10 slows down the performance of the C4.5 algorithm considerably. We decided to sample the non-robot sessions in the training and test data sets such that the number of non-robot sessions is at most 10 times larger

[8]It was previously stated that sessions that do not contain any request for HTML pages are assumed to have one episode $(-,-)$ associated with its last log entry.

Table 7: Number of sessions with different agent types for various session lengths.

Session length	# Type 1	# Type 2	# Type 3	# Type 4	Total Sessions
1	8487	165354	2795	3966	180602
2	3171	49311	549	1234	54265
3	2115	30189	246	827	33377
4	1678	21367	158	658	23861
5	1458	15560	99	539	17656
6	1127	12057	74	439	13697
7	937	9732	59	393	11121

than the number of unique robot sessions. For stratification by undersampling, we sample 100 out of 1000 non-robots in both training and test sets. In addition to stratification, we have also experimented with the full unstratified data set for E1. The unstratified data set will be denoted as E2. A summary of the size of each data set, E0, E1, E2, E3 and E4 is given in Table 8. The C4.5 algorithm is then used to build the classification models for each data set. Finally, the random sampling and model building procedure is repeated 10 times for all five data sets.

4.2 Correlation Analysis

Figure 5 shows the correlation [9] between each attribute with the class label. The bar graph is plotted for various session lengths (i.e. number of episodes in a session): The following observations can be made from the results of Figure 5:

1. As expected, the attributes used for creating class labels (i.e. attributes 5, 19 and 24) have very strong positive correlation with robot sessions, even though the majority of the robot sessions are identified by their agent types rather than by the values of these attributes. This confirms the validity of our session labeling heuristics. Nevertheless, the correlation coefficient for each of these attributes are less than 1. This suggests that using any of these attributes alone are insufficient to determine Web robot sessions. More importantly, the values of these attributes can be easily manipulated by robot designers.

2. After one request, the best predictors for robots, beside the attributes used for class labeling, are % image (attribute 2) and % GET request (attribute 17). These at-

[9]Note that linear correlation may not be the best measure of attribute dependence when non-linear dependencies exist in the data.

Table 8: Size of training and test sets for various experiments. E0 and E1 are data sets created using stratification by oversampling the population. E2 is the unstratified data set. E3 and E4 are samples from the same population as E1 and E2 respectively, except they are created using stratification by undersampling the population.

Session length (# Requests)	Data Set	# Unique Robots	# Unique Non-Robots	# Train Robots	# Train Non-Robots	# Test Robots	# Test Non-Robots
1	E0	8487	165354	42430	42430	42440	42440
	E1	11282	169320	56410	56410	56410	56410
	E2	11282	169320	5641	56410	5641	56410
	E3	8487	165354	4243	4243	4244	4244
	E4	11282	169320	5641	5641	5641	5641
2	E0	3171	49311	15850	15850	15860	15860
	E1	3720	50545	18600	18600	18600	18600
	E2	3720	50545	1860	18600	1860	18600
	E3	3171	49311	1585	1585	1586	1586
	E4	3720	50545	1860	1860	1860	1860
3	E0	2115	30189	10570	10570	10580	10580
	E1	2361	31016	11800	11800	11810	11810
	E2	2361	31016	1180	11800	1181	11810
	E3	2115	30189	1057	1057	1058	1058
	E4	2361	31016	1180	1180	1181	1181
4	E0	1678	21367	8390	8390	8390	8390
	E1	1836	22025	9180	9180	9180	9180
	E2	1836	22025	918	9180	918	9180
	E3	1678	21367	839	839	839	839
	E4	1836	22025	918	918	918	918
5	E0	1458	15560	7290	7290	7290	7290
	E1	1557	16099	7780	7780	7790	7790
	E2	1557	16099	778	7780	779	7790
	E3	1458	15560	729	729	729	729
	E4	1557	16099	778	778	779	779
6	E0	1127	12057	5630	5630	5640	5640
	E1	1201	12496	6000	6000	6010	6010
	E2	1201	12496	600	6000	601	6010
	E3	1127	12057	563	563	564	564
	E4	1201	12496	600	600	601	601
7	E0	937	9732	4680	4680	4690	4690
	E1	996	10125	4980	4980	4980	4980
	E2	996	10125	498	4980	498	4980
	E3	937	9732	468	468	469	469
	E4	996	10125	498	498	498	498

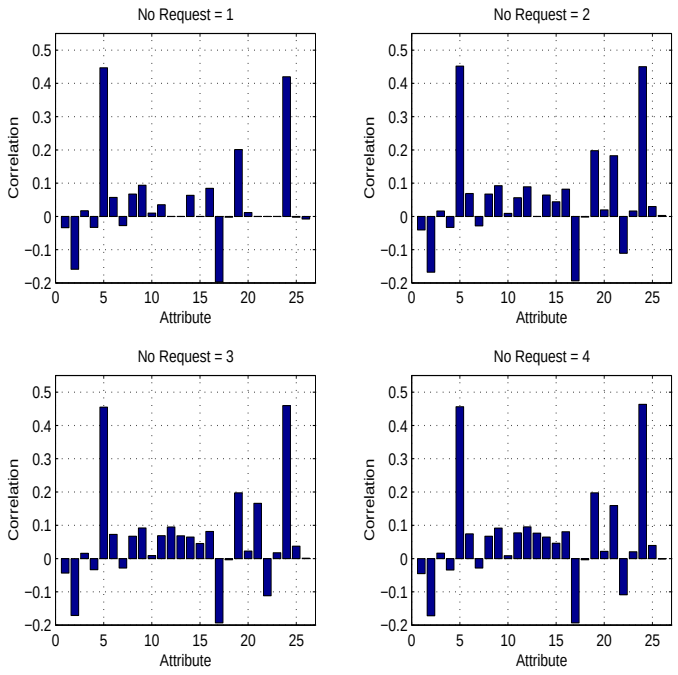


Figure 5: Correlation between access attributes and the Robot class label for various session lengths. The x-axis corresponds to the attribute Ids given in Table 3.

tributes have strong anti-correlation with robot sessions, agreeing with our intuition that most robots tend to ignore image files and use other HTTP request methods to retrieve the files (such as the HEAD request method). Another HTTP request method called POST (attribute 18) has very small negative correlation because it is used mostly by browsers to send HTML forms. Attributes 8 (% Zip) and 9 (% Multimedia) are positively correlated due to sessions with Type 3 agents (which are mostly downloading robots). Attributes such as $avgTime$, $stdevTime$, $width$ and $depth$ do not play a significant role because their values are either all zeros or all ones.

3. After two requests, the $avgTime$, $width$ and $depth$ attributes become more significant. The *Repeated* attribute also emerges as another predictor for robot sessions. This is because with a single request, the *Repeated* flag is always zero. The width is positively correlated with robot sessions, whereas the depth attribute is negatively correlated. This confirms our previous claim that many of the robots, especially the search engine ones, use a breadth-first search strategy and unassigned referrer fields to retrieve documents from a Web site. Also, notice that the MultiIP attribute is positively correlated, due to robots that parallelize their retrieval process.

4. After three requests, the *stdevTime* attribute becomes non-zero. A somewhat surprising fact is that this attribute is positively correlated with robot sessions, indicating that robots seem to have more irregular interval between requests compare to human users. We verified this by comparing the average standard deviations for various User Agents as shown in Figure 6.

5. The *Night* attribute has a positive correlation with the robot session. Figure 7 illustrates the hourly traffic at our Web server, after filtering out the anomalous Linbot session of figure 1. Notice that the number of page requests due to Web robots are almost uniformly distributed throughout the day, while the number of page requests due to non-robot sessions peaked at normal business hours[10]. Thus, it is surprising that the *Night* attribute is positively correlated with robot sessions. Upon closer examination, we found that this is because most of the robot sessions have long session interval, spanning into the 12am to 7am time window, which was used to determine the *Night* attribute. Out of the 10845 robot sessions, 3127 (28.8%) of them are night crawlers, compare to 30661 (18.1%) out of 169757 non-robot sessions that have $Night = 1$.

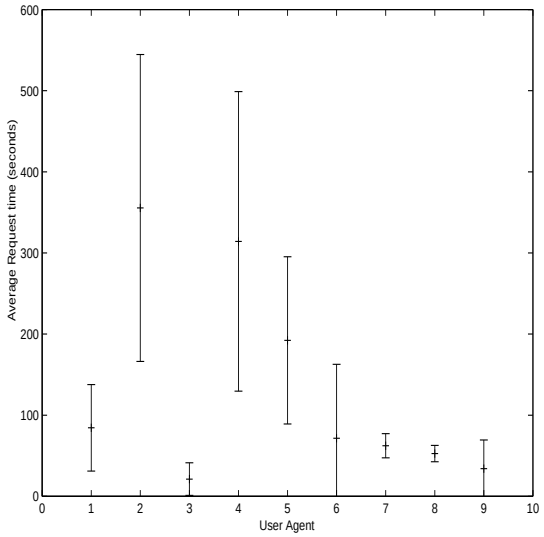
4.3 Classifier Performance

Figure 8 illustrates the overall classification accuracies for various models induced from our data sets. Our results show that after four requests, we can attain an overall accuracy close to 90%. Also, the precision and recall results in Figure 9 consistently reach above 82 % and 95% respectively, after more than three requests. The addition of noisy data (for E1 and E4) does not degrade the classifier precision accuracy by much. The recall however will decrease by as large as 5%. The small difference between the E0 and E1 (along with E3 and E4) curves for large session lengths can be explained by the relatively few number of Type 3 and Type 4 agents (see Table 7). Our results for E2 indicate that the accuracy measure can be misleading especially when there is an uneven distribution of robot and non-robot sessions. The recall and precision for E2 are extremely poor compare to those for E0 and E1 (E3 and E4). The different stratification strategies (oversampling versus undersampling) also seems to affect the precision-recall curve. Undersampling seems to aid recall at the expense of higher precision, in comparison to oversampling. However, more experiments are needed to confirm this phenomena.

There is a dramatic improvement in all three performance measures when the number of

[10]The observed traffic pattern is very similar to the e-commerce traffic observed by Rosenstein in [23].

Figure 6: Comparison of average $avgTime$ and $stdevTime$ for various User Agents.



1. University of Minnesota users	6. Ultraseek
2. Architext Spider	7. Email collector
3. Google	8. Link Checker
4. Lycos Spider	9. Offline Browser
5. Scooter	

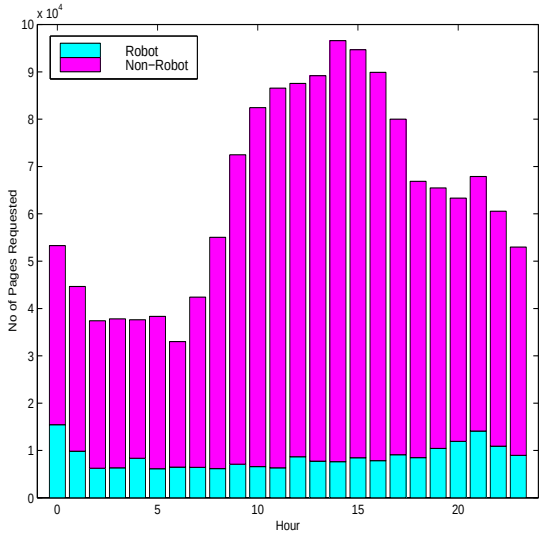


Figure 7: Summary of Hourly Web traffic at the University of Minnesota Computer Science department Web server.

request increases from one to two. This is due to the fact that attributes such as *avgTime*, *width*, *depth* and *Repeated* have different values for different sessions after more than one request is made.

The decision trees produced by the C4.5 algorithm can be used to generate classification rules, by using an auxiliary program called C4.5rules. Table 9 presents some of the high-confidence rules for the robot classes generated for each session length. Most of the rules seem to agree intuitively with our initial correlation analysis. For sessions of length 1, the rules that characterize the robot sessions are rather spurious, and tend to contain many attributes in their antecedent. This is because many of the good predictors (such as time attributes, width and depth attributes) are insignificant when the session length is 1. This explains the low recall of the results. Table 9 show one such spurious rule which states that a robot is a client that retrieves more than 4 files at night, out of which 1 of them is an HTML file, while the rest could be image, binary executable, ascii or other type of document files. Classifiers built with sessions of length 1 are often characterized by the absence of requests for image files and the presence of binary executable, ascii or zipped files.

For sessions of length 2, the *avgTime*, *width* and *depth* attributes help to improve the accuracy of predicting non-robot sessions. In the example rule given in Table 9, robots are classified by sessions that access the server at night, with average request time between 32 and 737 seconds, having low traversal depth and retrieves very few image and binary executable files. For longer sessions, notice the importance of the width attribute in characterizing robot sessions compare to non-robot sessions.

4.4 Finding Mislabeled Data

In this section, we analyze the samples that are often misclassified by the classification models generated from the C4.5 algorithm. The misclassification could be due to inaccuracy of the classifiers or incorrect class labels of the samples. We believe that such analysis can reveal useful information about some of the previously unknown Web robots.

The technique described in Section 3.5 is used to find sessions that are classified wrongly by most of the classifiers. The collection of classifiers used to determine the false positive and false negative scores could be based on all (or a subset of) the classifiers built in the previous section. However, one drawback of doing this approach is that different samples may appear in different number of classifiers (e.g. a session of length 7 will be classified by many more classifiers compare to those of length 1). One way to get around this problem is by taking a weighted sum of false positive and false negative scores, i.e. by

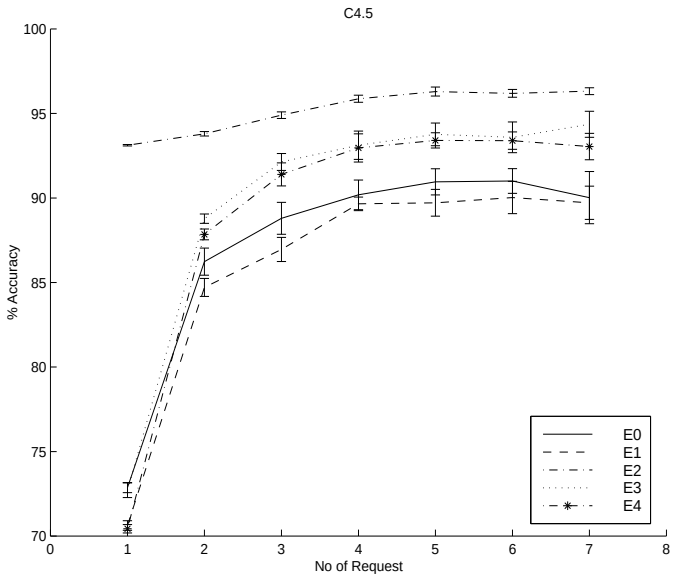


Figure 8: Accuracies of classification models using different session lengths.

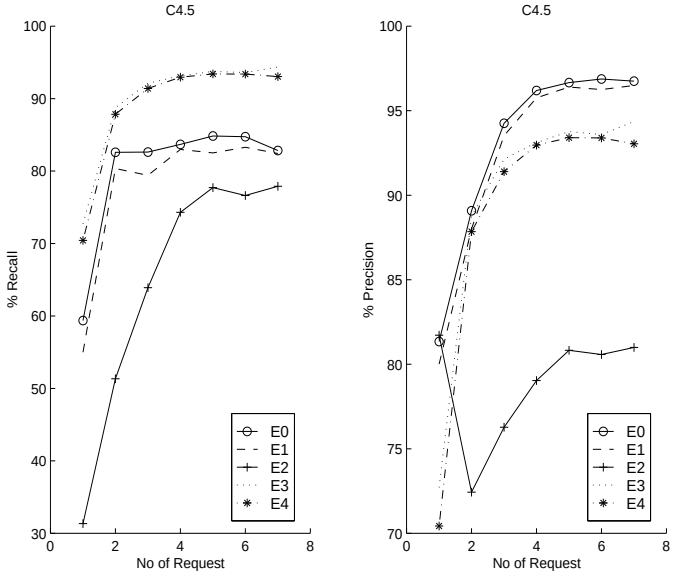


Figure 9: Recall and Precision of classification models using different session lengths.

Table 9: Some of the high-confidence decision rules produced by the different classification models.

Session length	Induced rules [Confidence]
1	$Night = 1$, $totalPages > 3$, % Image ≤ 0.0026, % Binary Exec ≤ 0.9285, % HTML ≤ 0.0526, % Ascii ≤ 0.9, $MultiAgent = 0$, % Other ≤ 0.0312 $\longrightarrow$ class 1 [97.1%]
2	$avgTime \leq 737$, $Night = 1$, % Image ≤ 0.2592, % Binary Exec ≤ 0.3636, $avgTime > 32$, $depth \leq 1$, $multiIP = 1$ $\longrightarrow$ class 1 [98.2%]
3	% Binary Doc ≤ 0.0096, % Binary Exec ≤ 0, $totalTime > 1861$, $Error > 0.1$, $width > 2$ $\longrightarrow$ class 1 [99.6%]
4	$totalPages > 4$, % Image ≤ 0.1, % HTML > 0.6666, $width > 2$, $MultiAgent = 1$, $GET > 0.9473$ $\longrightarrow$ class 1 [98.5%] $Night = 1$, $width \leq 1$, $height > 1$ $\longrightarrow$ class 0 [99.7%]

dividing each $FP(X)$ and $FN(X)$ score with the number of classifiers used to classify the sample. However, we found that this approach is rather unsatisfactory because most of the samples with high $FP(X)$ and $FP(Y)$ scores are the ones that are misclassified only once or twice. An alternative approach is to build k classifiers using the overall data set, where each sample is represented by the features of a session at its maximum session length. In addition, attributes that are used to determine the class labels are also included in the classification task. This is a reasonable approach considering our goal here is to find samples that are most likely being mislabeled, rather than to build accurate predictive models.

Figure 10 showed some of the sessions having the highest false positive scores (i.e. sessions predicted to be a robot by classifier but labeled as a non-robot).

1. The first session contains a User Agent that looks similar to a Netscape Navigator browser. However, the session seems to cover a significant portion of the Web site[11] without focusing on any specific topic, which is why it is highly likely that the session in fact belongs to a Web robot. Upon resolving the hostname of the session, our suspicion becomes even greater since NEC Research, which owns the domain address, are known to have a Scientific Bibliography Search Engine. Thus, the session is very likely created by a search engine robot.

2. The second session also looks suspicious, despite having a User Agent declared as

[11] We have only showed five of the requested pages.

Microsoft Internet Explorer. It is highly unlikely that a human user will be able to access all the four separate HTML pages within the same timeframe. Unfortunately, we were unable to resolve their IP Address to confirm the origin of the client.

3. The third example is especially interesting since almost all of the pages retrieved during the session are resumes. Our classifiers were able to detect the large width of the traversal to infer that the session belongs to a Web robot. This observation is confirmed after resolving the IP address of the session (i.e. hire.com).

4. The fourth example is another session we believe is created by a resume hunter robot since all the retrieved files are resumes. It is interesting to note that the domain name of the client belongs to a broadband Internet Service Provider. Thus, traditional techniques of finding robots based on the User Agent and IP Address fields will not work in this example.

Note that these four sessions also have high false positive scores when we apply the technique on classifiers built without using the class labeling attributes (i.e. robots.txt, % HEAD and referrer = "-").

The false negative sessions contain mostly robots that behave almost similar to human users (e.g. offline browsers) and robots with extremely short session lengths. Figure 11 showed some of the robots that are being mislabeled as non-robots. Note that SilentSurf (the fourth session) was initially thought to be a Type 3 robot. However, the classifiers identified it to be a non-robot. Upon further examination, we discovered that SilentSurf is in fact an anonymizer Web site which changes the User Agent of a browser into a robot-like value. Thus, it should be labeled as a non-robot session.

5 Conclusion

Our results show that highly accurate robot classifiers can be built using features based upon the access patterns of Web clients. These features are easily derived from Web server logs. Unlike attributes such as robots.txt, HEAD request methods and unassigned referrers, these features are harder to camouflage since they depend on the navigational goals of the client. Some of the most discriminating features used to predict robot sessions include the % of image files requested, *width* and *depth* of the traversal, % GET request methods and average time between request. Our experimental results suggest that Web robots can be detected using these features with reasonably high accuracy after 4 episodes. We have also shown that classifiers built using this technique can effectively identify camouflaging robots that have similar access patterns as other well-known robots.

IP Address/Hostname	Time	Requested Page	User Agent
kablam.nj.nec.com	14:47:23	/~jcui	Mozilla/4.05 [en] (Win95; U)
kablam.nj.nec.com	14:48:32	/~hsieh/misc/misc.html	Mozilla/4.05 [en] (Win95; U)
kablam.nj.nec.com	14:49:06	/Research/dmc/html/abstracts.html	Mozilla/4.05 [en] (Win95; U)
kablam.nj.nec.com	14:49:15	/~kencham/./abstracts/VCR-Ops.html	Mozilla/4.05 [en] (Win95; U)
kablam.nj.nec.com	15:14:03	/~gini/motion.html	Mozilla/4.05 [en] (Win95; U)
kablam.nj.nec.com	15:36:13	/~vijesek/research/qosMetrics.html	Mozilla/4.05 [en] (Win95; U)
64.3.57.99	5:06:42	/employment	Microsoft Internet Explorer/4.40.426 (Windows 95)
64.3.57.99	5:06:43	/grad-info	Microsoft Internet Explorer/4.40.426 (Windows 95)
64.3.57.99	5:06:43	/reg-info/csMinor.html	Microsoft Internet Explorer/4.40.426 (Windows 95)
64.3.57.99	5:06:43	/industry.html	Microsoft Internet Explorer/4.40.426 (Windows 95)
tpa1.hire.com	13:59:42	~hngo/vnsa/may27-jul24/msg01844.html	Mozilla/4.75 [en] (X11; U; Linux 2.2.16-3 i686)
tpa1.hire.com	14:01:20	/~sparikh/resume/shwetal_resume.html	Mozilla/4.75 [en] (X11; U; Linux 2.2.16-3 i686)
tpa1.hire.com	14:12:27	/~whalen/resume.html	Mozilla/4.75 [en] (X11; U; Linux 2.2.16-3 i686)
tpa1.hire.com	4:31:38	/~steinmet/pages/steinmetzresume.html	Mozilla/4.75 [en] (X11; U; Linux 2.2.16-3 i686)
rfx-64-6-194-38.users.reflexcom.com	14:51:00	/~myers/resume.html	Mozilla/4.0 (compatible; MSIE 5.01; Windows 98; DigExt)
rfx-64-6-194-38.users.reflexcom.com	14:58:25	/~tiang/resume.html	Mozilla/4.0 (compatible; MSIE 5.01; Windows 98; DigExt)
rfx-64-6-194-38.users.reflexcom.com	15:03:45	/~lttau/resume.html	Mozilla/4.0 (compatible; MSIE 5.01; Windows 98; DigExt)
rfx-64-6-194-38.users.reflexcom.com	15:11:17	/~tnnguyen/resume	Mozilla/4.0 (compatible; MSIE 5.01; Windows 98; DigExt)

Figure 10: Sessions identified as having large false positive scores.

IP Address/Hostname	Time	Requested Page	User Agent
ns.mof.go.jp	18:42:16	/~subraman/cgi-bin/art.cgi	-
ns.mof.go.jp	18:48:13	/~subraman/cgi-bin/art.cgi	-
ns.mof.go.jp	18:48:20	/~subraman/arts/main.html	-
ns.mof.go.jp	18:48:20	/~subraman/arts	-
cip123.studcs.uni-sb.de	7:06:13	/~mobasher/webminer/survey/survey.html	Java1.1.8
cip123.studcs.uni-sb.de	7:20:51	/~mobasher/webminer/survey/survey.html	Java1.1.8
cip123.studcs.uni-sb.de	7:28:20	/~mobasher/webminer/survey/survey.html	Java1.1.8
212.160.138.34	8:16:31	/~hougen	Offline Explorer/1.3
212.160.138.34	8:21:05	/departmental	Offline Explorer/1.3
212.160.138.34	8:21:06	/Research/airvl	Offline Explorer/1.3
63.87.244.21	4:45:02	/~sparikh	SilentSurf/1.1x [en] (X11; I; $MyVersion)
63.87.244.21	4:45:05	/~sparikh/images/headsil.jpg	SilentSurf/1.1x [en] (X11; I; $MyVersion)
63.87.244.21	4:45:06	/~sparikh/images/back/ivy.gif	SilentSurf/1.1x [en] (X11; I; $MyVersion)

Figure 11: Sessions identified as having large false negative scores.

However, our technique may fail for robots that behave in a manner similar to human users. For example, we observe that most of our false negatives are due to offline browsers and other download utility programs that have very similar characteristics with human users. Further investigation is needed to study the effect of other types of navigational patterns not captured by our data.

Our models can be made much more accurate by refining the features used for building the classifiers. For instance, we can incorporate other metrics as defined by W3C Web Characterization Metrics [15] into the feature vector construction. Our current model can also be improved by incorporating Web content and structure information. Our techniques could also be improved by using more reliable session tracking techniques such as cookies and embedded session Ids.

References

[1] Altavista search engine. http://www.altavista.com.

[2] M. Balabanovic and Y. Shoham. Learning information retrieval agents: Experiments with automated web browsing. In *On-line Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, 1995.

[3] D. Clark. Shopbots become agents for business change. *IEEE Computer*, pages 18–21, February 2000.

[4] R. Cooley. *Web Usage Mining: Discovery and Application of Interesting Patterns from Web Data.* PhD thesis, University of Minnesota, 1999.

[5] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Knowledge and Information Systems*, 1(1), 1999.

[6] D. Eichmann. Ethical web agents. *Computer Networks and ISDN Systems*, 28(1), 1995.

[7] Email digger. http://www.strayernet.com/webdesign/emailpro.html.

[8] Extractor pro. http://www.extract.com.

[9] Google search engine. http://www.google.com.

[10] M. Gray. Measuring the growth of the web. http://www.mit.edu/people/mkgray/growth/, 1993.

[11] J. Kephart and A. Greenwald. Shopbot economics. In *Agents*, 1999.

[12] M. Koster. Guidelines for robot writers. http://info.webcrawler.com/mak/projects/robots/guidelines.html, 1994.

[13] M. Koster. A standard for robot exclusion. http://info.webcrawler.com/mak/projects/robots/norobots.html, 1994.

[14] M. Koster. Robots in the web: threat or treat. *ConneXions*, 9(4), 1995.

[15] B. Lavoie. Web characterization metrics. http://www.oclc.org/oclc/research/projects/webstats/ currmetrics.htm, 1999.

[16] H. Lieberman. Letizia: An agent that assists web browsing. In *Proc. of the 1995 International Joint Conference on Artificial Intelligence*, Montreal, Canada, 1995.

[17] Link scan. http://www.elsop.com/linkscan/.

[18] Lycos search engine. http://www.lycos.com.

[19] Offline explorer. http://www.metaproducts.com.

[20] Peter Pirolli, James Pitkow, and Ramana Rao. Silk from a sow's ear: Extracting usable structures from the web. In *CHI-96*, Vancouver, 1996.

[21] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[22] Robot exclusion standard revisited. http://www.kollar.com/robots.html, 1996.

[23] M. Rosenstein. What is actually taking place on web sites: E-commerce lessons from web server logs. In *ACM Conference on Electronic Commerce*, Minneapolis, MN, 2000.

[24] Spider hunter. http://www.spiderhunter.com.

[25] Teleport pro. http://www.tenmax.com/teleport/pro/home.htm.

[26] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[27] Windows 95/98 offline browser tools. http://winfiles.cnet.com/apps/98/offline.html.

[28] Xenu's link sleuth. http://home.snafu.de/tilman/xenulink.html.

[29] Yiming Yang. An evaluation of statistical approaches to text categorization. *Information Retrieval*, 1(1–2), 1999.

[30] M. Yoon. Web robot detection. http://www.arsdigita.com/doc/robot-detection.

A Session Identification Heuristic

One of the main challenges of mining Web data from server logs is to transform the click-stream data into server sesions. Without the benefits of client-side tracking, cookies or embedded session Ids, it is extremely difficult to reliably preprocess the Web server logs into server sessions. A standard way of doing this is to group together log entries that have the same IP Address and User Agent fields [20, 5]. However, the drawback of this approach is that requests with the same IP Address and User Agent may have come from more than one active session. Cooley [4] proposed a potential solution to this problem by using the referrer field to distinguish sessions that share a common IP Address and User Agent.

In this work, we have extended the session identification heuristic suggested in [4] to handle the situation in which a session may contain multiple IP addresses and User agents. This is of great importance to our work because we have noted that Web robots can

multiplex several IP addresses together in order to parallelize their retrieval operations. Both Web robots and browsers can also use more than one User Agent in the same session.

There are several notable differences between the session identification heuristic of [4] and our proposed work:

1. We do not have to sort the log entries according to their User Agent and IP Address combination. This allows sessions to contain log entries from multiple IP addresses or User Agents.

2. In order to match a log entry l_j to its corresponding session, we partition the list of active sessions H into 4 candidate groups: candidateSet[1], candidateSet[2], candidateSet[3] and candidateSet[4]. The first group contains sessions that have the same IP Address and User Agent as l_j[12]. They are the ones that l_j will most likely belong to. candidateSet[2] contains sessions that share a common domain name as l_j (e.g. requests from crawler1.googlebot.com and crawler2.googlebot.com have the same domain name, google.com) and have the same User Agent. We use a reverse DNS lookup program to resolve the hostname of each Web client. A suffix of the hostname is used to represent the domain name of the client. The third group, candidateSet[3], contains sessions that have the same User Agent and share a common prefix IP address as l_j. This step is needed because not all hostnames can be resolved by our DNS server. Some hostnames are not resolved due to server timeout, non-existent host/domain errors, etc. The fourth group, candidateSet[4], contains sessions that have the same IP Address but not the same User Agent field[13] as l_j. They are the last set of candidate sessions that will be matched against l_j.

Table 10 summarizes the key steps of our algorithm. For each log entry l_j, we use the getCandidates function to generate the four sets of candidate sessions that will most likely contain l_j. Next, the BestCandidate function will select the most likely session among the sets of candidate sessions. In the BestCandidate function, sessions in candidateSet[1] are compared first, since they are the ones that have the most similar characteristics to l_j. If no similar sessions are found, then candidateSet[2] is compared, followed by candidateSet[3] and finally, candidateSet[4]. A log entry is said to be similar to an active session if its referrer field is the same as one of the requested or referred pages in the active session and the time interval between the last request of the active session and l_j is not too large. This similarity measure is analogous to the Distance function of [4]. However, unlike [4], the

[12]This is the only candidate set used in [4].

[13]As mentioned in Section 2, some browsers use a different User Agent field, such as contype, when retrieving non-HTML files

referrer field is matched against both the request field and referrer field of active sessions, depending on whether the referred page is an internal or external Web page.

One potential pitfall of our session identification heuristic is that it could group together sessions that actually belong to different users. For example, different sessions coming from the same domain (e.g. lnx02.cs.umn.edu and lnx03.cs.umn.edu) could be grouped together even though they are created by different users. This seems to be the case for many sessions that originate from our own university. This is because for any given 30 minute time window (which is our session timeout interval, T), there is a large number of requests that come from the domain umn.edu. The likelihood that the referrer field of any one of the requests being the requested or referred page of another requests (from the same domain) is extremely high; thus causing the different requests to be grouped together in the same session (even though they may be part of different sessions). This problem is not as critical for other domains.

The problem can be resolved by restricting the candidateSet[2] group to contain sessions with domains other than umn.edu (and several other known non-sharing ISPs). We verified this step by inspecting the domains of all the remaining sessions that contain multiple hostnames.

Table 10: Modified Session Identification Heuristic.

type logEntry {	type session {
ip : string, *request* : URI	*count* : integer
time : seconds, *referrer* : URI	*list* : array of logEntry
agent : string, *method* : string	}
status : string *protocol* : string	
}	

```
1.  Let H denotes the set of active sessions.
2.  Let L denotes a time-ordered log entries.
3.  Let T denote the session timeout.
4.  for each l_j ∈ L do
5.      for each s_j ∈ H do
6.          if (s_j.list[s_j.count].time − l_j.time > T) then
7.              close session s_j
8.      end;
9.      candidateSet = getCandidates(H, l_j)
10.     if (candidateSet is NULL)
11.         create new session s'
12.         add l_j to s' and increment s'.count.
13.         add s' to H.
14.     else
15.         assign = bestCandidate(candidateSet, l_j)
16.         if (assign is NULL)
17.             create new session s'
18.             add l_j to s' and increment s'.count.
19.             add s' to H.
20.         else
21.             add l_j to assign
22.             increment assign.count
23. end;
```

Table 11: getCandidate and bestCandidate functions.

```
function getCandidate(H: set of session, l_j: logEntry)
1.  Let candidateSet[][] be a two-dimensional array of sessions
2.  for each s_j ∈ H do
3.     if (containsAgent(s_j, l_j.agent)) then
4.        if (containsIP(s_j, l_j.ip) then
5.           add s_j to candidateSet[1]
6.        else if (sameDomain(s_j, l_j.ip)) then
7.           add s_j to candidateSet[2]
8.        else if (sameAddressClass(s_j, l_j.ip)) then
9.           add s_j to candidateSet[3]
10.    else
11.       if (containsIP(s_j, l_j.ip)) then
12.          add s_j to candidateSet[4]
13. end;
```

```
function bestCandidate(C: two dimensional array of sessions, l_j: logEntry)
1.  assign = NULL
2.  if (l_j.referrer is a local page) then
3.     for i=1 to 4 do
4.        assign = find s_k ∈ C[i] such that (l_j.time − s_k.list[s_k.count].time)
               is minimum and l_j.referrer ∈ requestSet(s_k)
5,        if assign is not NULL then return assign
6.     end;
7.  else    /* if l_j.referrer is an external page or unassigned */
8.     for i=1 to 4 do
9.        assign = find s_k ∈ C[i] such that (l_j.time − s_k.list[s_k.count].time)
               is minimum and l_j.referrer ∈ referrerSet(s_k)
10        if assign is not NULL then return assign
11.    end;
12. return NULL
```