Discovery of Web Robot Sessions based on their Navigational Patterns

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

Abstract. Web robots are software programs that automatically traverse the hyperlink structure of the World Wide Web in order to locate and retrieve information. There are many reasons why it is important to identify visits by the Web robots and distinguish them from other users. First of all, e-commerce retailers are particularly concerned about the unauthorized deployment of robots for gathering business intelligence at their Web sites. In addition, Web robots tend to consume considerable network bandwidth at the expense of other users. Sessions due to Web robots also make it more difficult to perform clickstream analysis effectively on the Web data. Conventional techniques for detecting Web robots are often based on identifying the IP address and user agent of the Web clients. While these techniques are applicable to many well-known robots, they may not be sufficient to detect camouflaging and previously unknown robots. In this paper, we propose an alternative approach that uses the navigational patterns in the click-stream data to determine if it is due to a robot. Experimental results on our Computer Science department Web server logs show that highly accurate classification models can be built using this approach. We also show that these models are able to discover many camouflaging and previously unidentified robots.

Keywords: web usage mining, web robot detection, classification, data mining

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. The emergence of the World Wide Web as an information dissemination medium, along with the availability of many Web robot authoring tools have resulted in the rapid proliferation of Web robots unleashed onto 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 (Gray, 1993). As another example, Internet search engines such as Google and Altavista 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 by business organizations

 Appeared in Data Mining and Knowledge Discovery, 6(1): 9-35 (2002).

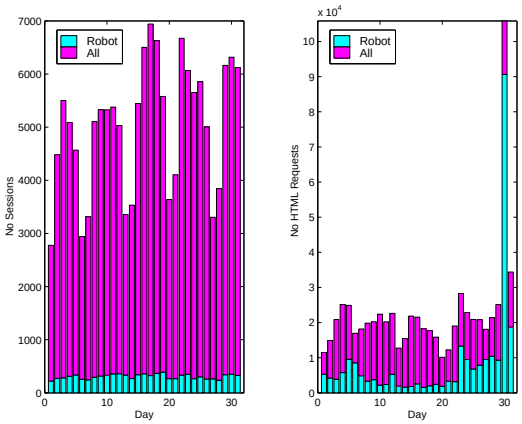

Figure 1. Summary of daily Web traffic at the University of Minnesota Computer Science department Web server. The above figure shows the fraction of total sessions and HTML requests due to Web robots. The anomaly on day 30 for the right-hand figure is due to HTML requests from a site mapping robot called linbot.

to collect email addresses and online resumes, monitor product prices, corporate news, etc.

There are many situations in which it is desirable to identify visits by Web robots and distinguish them from other users. First, e-commerce retailers are particularly concerned about the unauthorized deployment of Web robots for gathering business intelligence at their Web sites. In such a situation, the e-commerce site may want to block HTTP requests coming from the unauthorized robots (Graham, 2000). 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 e-commerce Web sites perform Web traffic analysis in order to infer the demographic and browsing behavior of their customers. Unfortunately, such analysis can be severely distorted by the presence of traffic due to 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 wrong inferences about the site visitors.

Thirdly, the deployment of Web robots usually comes at the expense of other users because they often consume considerable network bandwidth and server resources. Poorly designed robots may tie up

Table I. List of IP addresses and user agents for several known 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(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)
Search Engine	199.172.149.184	ArchitextSpider
Email Harvester	4.41.77.204	EmailSiphon
Link Checker	130.237.234.90	LinkChecker/1.0
Search Engine (looksmart.com)	207.138.42.10	Mozilla/4.5 [en] (Win95; I)

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, 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. Unscrupulous referrer site owners can easily abuse such a payment scheme by inflating the click-through rate using Web robots. Hence, 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 detect robots is by examining the HTTP request messages sent to a Web server (Yoon, 2000; Jackson, 1998). For example, accesses by many well-known robots can be detected by comparing the IP address and user agent fields of their HTTP request messages against those of known robots (cf. Table I). However, since Web robots can be easily constructed and deployed, it is impossible to keep a comprehensive list of IP addresses and user agents for all robots. This problem is exacerbated by robots that attempt to disguise their identities by using user agents that are very similar to conventional browsers (e.g. the last entry of Table I). As a result, this approach may fail to detect the presence of such robots.

In this paper, we offer an alternative solution by building classification models that distinguish robot from non-robot sessions. Our main assumption is that the navigational patterns of Web robots are inherently different than those of human users.

Building classification models from Web usage data is a challenging problem due to the following reasons. First, the click-stream data is sequential and temporal in nature. Therefore, the models must take into account both the static and dynamic properties of each session. Furthermore, since the length of a Web session varies from one session to another, it is not sufficient to generate a single classification model for all the sessions. Instead, the models must be built incrementally, after each request made by the Web client. Secondly, it is not trivial to construct a reliable training set for the classification task. This requires a dependable preprocessing step to convert the raw click-stream data into individual sessions. In addition, the class label for each session must be determined. Manual assignment of the class labels can be a laborious task, while an automated procedure may introduce labeling errors into the training data. Thirdly, it would be desirable to detect the Web robots swiftly, after looking at just a few requests. This is a formidable goal because the smaller the number of requests of a session, the less information it contains to correctly predict whether it is due to a Web robot. Hence, building classification models for Web robots involve a tradeoff between early detection and model accuracy.

The main contributions of this paper are summarized below:

1. We analyze the navigational patterns for various types of Web robots and show that these patterns are quite different from those for human users.

2. We propose a robust session identification technique to preprocess the Web server logs. Our technique can identify sessions with multiple IP addresses and user agents.

3. We present a procedure for labeling the training and test data sets, and a technique for identifying the mislabeled samples. We show that this technique was able to discover many camouflaging and previously unknown Web robots.

4. We show that highly accurate robot classification models can be induced from the access features of the Web sessions.

The rest of the paper is organized as follows. In Section 2, we discuss the limitations of some of the existing techniques used for detecting Web robots. Section 3 describes the preprocessing steps needed to convert the raw click-stream data into server sessions. A discussion about how to derive the session features and class labels is also presented. This is followed by our experimental results in Section 4, while Section 5 concludes with suggestions for future work.

2. Web Robot Detection: Overview

2.1. Limitations of Current Robot Detection Techniques

In this section, we present some of the common techniques used to detect Web robots and describe the limitations of each technique:

1. **Robots.txt Access Check**

   The Robot Exclusion Standard (Koster, 1994b; Kolar et al., 1996) was proposed to allow Web administrators to specify which part of their Web site is off-limits to visiting robots. According to this Standard, whenever a robot visits a Web site, say at `www.xyz.com`, it should first examine the file `http://www.xyz.com/robots.txt`. This file contains a list of access restrictions for the Web site as specified by the Web administrator. For example, the following entry in the robots.txt file forbids all robots from accessing the file `http://www.xyz.com/A.html`.

   User-agent: *
   Disallow: /A.html

   This suggests that Web robots should be easily detected from sessions that access the robots.txt file. Indeed, this is a reasonably good heuristic because many Web sites do not provide a direct hyperlink to this file from any other HTML pages. As a result, most users are unaware of the existence of this file. However, one should not rely on this criterion alone because compliance to the Robot Exclusion standard is voluntary, and many robots simply do not follow the proposed standard.

2. **User Agent Check**

   It is commonly agreed that poor implementation of the Web robots can lead to serious network and server overload problems. Thus, a guideline is needed to ensure that both the Web robot and Web server can cooperate with each other in a way that is beneficial to both parties. Under the proposed ethical guidelines for robot designers (Eichmann, 1995; Koster, 1995; Koster, 1994a), a cooperative robot must declare its identity to a Web server via its user agent field. For instance, the user agent field of Web robots should contain the name of the robot, unlike the user agent field of Web browsers, which often contains the name `Mozilla` (cf. Table I). Figure 2 shows an example where an Internet Explorer browser, identified by its user agent field, `Mozilla/4.0 (compatible; MSIE 5.01)`, was used to request for the HTML page, `http://www.xyz.com/A.html`. In practice, not all robot designers adhere to these guidelines. Some

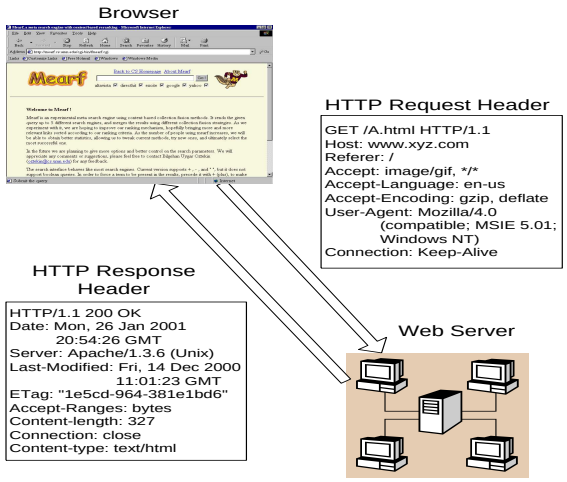

Figure 2. Communication between Web browser and Web server via HTTP protocol.

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 retrieving other documents. Even conventional Web browsers may issue requests with different user agent fields. In the following example, a user with the Microsoft Internet Explorer browser attempts to download a PDF document from our Web site. Two HTTP request messages are generated as a result of this action, each having its own user agent field (i.e. `MSIE 5.01` and `contype`). These requests are recorded in the Web server logs as:

203.94.250.186 - - [01/Jan/2001:15:18:04 -0600] "GET /grad-info/finapp.pdf 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 robot detection problem becomes more complicated when robot designers attempt to hide their identities by using the same user agent field as standard Web browsers. In this situation, it is impossible to detect these robots using their user agent fields alone. Similarly, some anonymizer Web sites allow Web users to camouflage their accesses by transforming the user agent fields of their browsers into robot-like (i.e. non-`Mozilla`) values such as `SilentSurf` and `Turing Machine` [2].

3. **IP Address Check**

   Another way to detect robots is by matching the IP address of a Web client against those of known robots. Even though there are many Web sites that provide a list of IP addresses for known Web robots[3], it is difficult to maintain an up-to-date database of all the robots. This is because new robots keep cropping up as the World Wide Web continues to expand. Another problem with this approach is that the same IP address could be used by Web users for surfing the Web and by robots to automatically download some files from a Web site. The IP address check approach is applicable only if the robot has been previously identified. One way to detect new robots is by examining the top visiting IP addresses of the Web clients and manually verify the origin of each client. Unfortunately, this technique is time-consuming, and often discovers robots that are already well-known.

4. **Count of HEAD requests and HTTP requests with unassigned referrers**

   The guidelines for Web robot designers also suggest that ethical robots should (1) moderate their rate of information acquisition, (2) operate only when the server is lightly loaded (e.g. at night), and (3) use the HEAD request method, whenever possible. The request method (e.g. GET, HEAD and POST) of an HTTP request message determines what type of action the Web server should perform on the resource requested by the Web client. For example, a Web server responds to a GET request by sending a message, consisting of some header information along with a message body, which contains the requested file. On the contrary, the response to a HEAD request contains only the message header, thus incurring less communication overhead. This is the reason why Web robots are encouraged to use the HEAD request method. In principle, one can examine sessions with a large number HEAD requests to discover ethical Web robots. In addition, one should also look for sessions that have a large number of requests with unassigned 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, whenever a user requests for the page `http://www.xyz.com/A.html` by clicking on a hyperlink found at `http://www.xyz.com`, the user's browser will generate an HTTP request message with its referrer field assigned to `http://www.xyz.com`. Since most robots do not assign any value to their referrer fields, these values appear as "-" in the Web server

logs. Both of these heuristics are not entirely reliable because non-robots can sometimes generate HEAD request messages (e.g. when proxy servers attempt to validate their cache contents) and HTTP messages with unassigned referrer values (e.g. when a user clicks on a bookmarked page or types in a new URI in the address window).

2.2. Motivation for Proposed Robot Detection Technique

The previous discussion suggests that a more robust technique is needed to identify visits by camouflaging and previously unknown Web robots. Before we describe our proposed technique, it is essential to understand what are the different types of robots that are available today (cf. Table II). This is because each type of robot may exhibit different characteristics depending on their navigational goals. Knowing the characteristics and navigational goals of these robots can help us to identify the set of relevant features for predicting Web robot sessions.

Table II. Typical characteristics and navigational goals of Web Robots.

Client's Type	Examples	Navigational Goals	Characteristics
Search Engine	T-Rex (Lycos), Scooter (Altavista)	maximize coverage of a Web site	breadth first search, unassigned referrer.
Offline Browser	Teleport Pro, Offline Explorer	download Web site to local disk	varied behavior.
Email Collector	EmailDigger, Extractor Pro	maximize coverage of home pages	unassigned referrer, ignores image files.
Link Checker	LinkScan, Xenu's Link Sleuth	check for broken links	HEAD request, unassigned referrer.

For example, consider a search engine robot. The ultimate goal of such robot is to retrieve as many documents as possible in a short period of time. As a result, these robots often use a breadth-first retrieval strategy to maximize their coverage and parallel retrieval strategy in order to speed up their operations.

Another type of Web robot is called a link checker, which is a utility program used by Web site administrators to check for broken hyperlinks and missing pages. Many link checkers would use the HEAD request message to check if a hyperlink still exists. A Web server would respond to the HEAD request by sending a message header containing a status code that indicates whether the request has succeeded or failed.

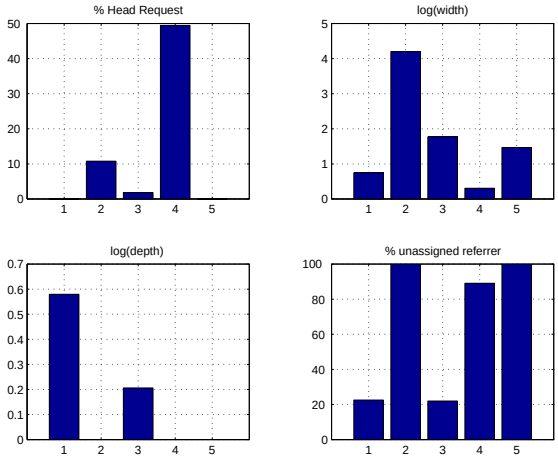

Figure 3. Comparison between the navigational patterns of several known Web clients. The horizontal axis represents each type of client: (1) Users from the Computer Science department at University of Minnesota (2) Search engine robots (3) Offline browsers (4) Link checkers, and (5) Email collectors, while the vertical axis represents the various session characteristics.

Web robots are also designed for various other purposes. For example, email collectors are robots that automatically collect email addresses posted on the Web. These robots tend to retrieve only HTML pages, while ignoring image and other file formats. Offline browsers are either stand-alone browsers or add-on utilities that allow a Web user to download the entire Web site (or portion of it) to a local directory for offline viewing. The characteristics of offline browsers may vary, depending on their navigational goals. For instance, offline browsers that download an entire Web site behaves very similar to search engine robots, while those that download only a small portion of the Web site (for pre-caching purposes) resemble the characteristics of human users. Other types of Web robots include personal browsing assistants (Balabanovic and Shoham, 1995; Lieberman, 1995), shopbots (Kephart and Greenwald, 1999; Clark, 2000), resume hunters, etc.

The main assumption of this paper is that the navigational patterns of Web robots are distinct from human users. These patterns can be characterized in terms of the types of pages being requested, the length of the session, the interval between successive HTML requests, the coverage of the Web site, etc. A more in-depth discussion about the navigational characteristics is presented in Section 3.2.

Figure 3 shows a comparison between the navigational patterns of various Web clients. Note the consistency between the observed patterns and the navigational goals of these robots given in Table II. For example, the HEAD method accounts for almost half of the requests

made by link checkers. The width and depth attributes, which are used to infer the search strategy employed by the clients, show that search engine robots tend to have broad width but shallow depth, indicative of a breadth-first behavior. Figure 3 also shows that most search engine robots, link checkers and email collectors do not assign any values to their referrer fields. On the other hand, offline browsers have very similar characteristics to human users in terms of their rate of HEAD requests and unassigned referrer fields. However, the width and depth of their accesses are quite different from human users.

The results of Figure 3 indicates that the overall navigational patterns of Web robots are different than those of human users. As a result, one should be able to construct accurate classification models to detect the presence of Web robots based on their navigational patterns. This is exactly the approach taken by this paper.

3. Methodology

The goal of our classification task is to build models that will automatically map each session into one of the two pre-defined classes: robot versus non-robot. This requires: (1) a preprocessing step to transform the Web data into individual sessions, (2) extraction of features that best describe the properties of each class, (3) a procedure to the determine the class label of each session (4) a technique to assess how good is the labeling scheme, (5) an algorithm to build classification models, and (6) a metric to evaluate the performance of the models. The first two requirements are described in Sections 3.1 and 3.2, respectively. The third requirement is needed to provide the training data for the classification algorithm, and is described in Section 3.3. The fourth requirement is described in Section 3.5 while the last two requirements are discussed in Section 3.4.

3.1. Data Source and Preprocessing

Web server logs are used as the data source for our experiments. A typical Web server log contains information such as the IP address and user agent of the Web client, the URI of the requested and referred pages, the request method and protocol used, the timestamp and status code of each request, and the size of the file that was transferred. Entries in the Web server logs are ordered according to their timestamps.

The first preprocessing step is to transform the raw click-stream data into server sessions. Without client-side tracking, cookies or embedded session identifiers, it is extremely difficult to identify the individual

sessions in the Web server logs reliably. A standard way to do this is by grouping together the Web log entries according to their IP address and user agent fields (Pirolli et al., 1996; Cooley et al., 1999). This approach may not work well for two reasons. First, each IP address/user agent pair may contain more than one session, e.g. sessions created by Web users who share the same proxy server and use the same type of browser. Secondly, a session that contains multiple IP addresses or user agents will be fragmented into multiple sessions.

Cooley (Cooley, 1999) have addressed the first problem by modifying his session identification heuristic to fragment each IP address/user agent pair into smaller sessions whenever the session contiguity condition is violated. This condition simply states that the referrer field of the current request of a session must match one of the pages previously retrieved by the session (unless the referrer value is not assigned).

Developing heuristics for solving the second problem is essential for detecting robot activities since many search engine robots tend to parallelize their Web retrieval operations. These heuristics are also useful for identifying user sessions if client-tracking technologies are unavailable. In this paper, we have proposed a new session identification algorithm for handling the second problem.

In order to match a log entry l_j to its corresponding session, we partition the list of currently active sessions H into 4 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. The second group, candidateSet[2], contains sessions that have the same user agent as l_j but different IP address. However, these sessions share a common domain name as l_j (e.g., `crawler1.googlebot.com` and `crawler2.googlebot.com` both have the same domain name, `googlebot.com`). We use a reverse DNS lookup program to resolve the hostname of each IP address. The third group, candidateSet[3], contains sessions that have the same user agent and prefix IP address as l_j (e.g. both 160.94.178.151 and 160.94.178.153 share a common class C address, 160.94.178). This group is needed because the hostnames for some sessions can not be resolved by the reverse DNS lookup program due to server timeout, non-existent host/domain error, etc. The last group, candidateSet[4], accounts for sessions that have the same IP address but different user agents as l_j.

Table III summarizes the key steps of our session identification algorithm. For each log entry, l_j, we first find the candidate sessions that may potentially contain l_j. The getCandidates function (Table IV) will generate the four groups of candidate sessions mentioned above. Next, the BestCandidate function (Table IV) will choose the most likely session to contain l_j. In this function, candidateSet[1] is scanned first, since

Table III. Modified Session Identification Heuristic.

```
type logEntry {
   ip : string,        request : URI
   time : seconds,     referrer : URI
   agent : string,     method : string
   status : string     protocol : string
}

type session {
   count : integer
   list : array of logEntry
   Class : integer
}
```

```
1.  Let H denotes the set of active sessions.
2.  Let L denotes the time-ordered Web 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) then
11.       create new session s'
12.       add l_j to s'.list and increment s'.count.
13.       add s' to H.
14.    else
15.       assign = bestCandidate(candidateSet, l_j)
16.       if (assign is NULL) then
17.          create new session s'
18.          add l_j to s'.list and increment s'.count.
19.          add s' to H.
20.       else
21.          add l_j to assign.list
22.          increment assign.count
23. end;
```

they are the ones that will most likely contain l_j. If no matching session is found, then candidateSet[2] is scanned, followed by candidateSet[3], and finally, candidateSet[4]. We use a 30-minute session timeout and the session contiguity conditions to determine whether l_j belongs to a given session.

One potential pitfall of our approach is that it may inadvertantly group together sessions that belong to different users. For example, requests that come from different machines but with the same domain name (e.g. lnx02.cs.umn.edu and lnx03.cs.umn.edu) could be grouped

Table IV. The getCandidates function.

```
function getCandidates(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;
14. return candidateSet
```

```
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  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
```

together even though they belong to different users. This is because our session identification heuristics are capable of distinguishing between the two sessions as long as they do not overlap each other in terms of their set of requested pages or their request times. Unfortunately, we found that both conditions can be violated by sessions due to users from our university. We have attempted to correct this problem by ignoring

candidateSet[2] and candidateSet[3] for clients with hostnames that end with umn.edu or other known non-proxy hostnames. We verify this step by checking all the sessions that still contain multiple IP adddresses.

3.2. Feature Vector Construction

After the server sessions have been created, the next step is to derive the properties of each session. Each session is broken up into several episodes, where each episode corresponds to a request for an HTML file. (We will use the terms episode and request interchangeably throughout this paper.) We associate each episode with a tuple, (p_i, p_j), where p_i and p_j are the requested and referred HTML pages. A session that does not contain any request for HTML files will have a single episode, $(-, -)$ associated with its last log entry.

Table V presents a summary of attributes that can be derived from the server sessions. The values of these attributes may change as the session length increases. The computation of temporal attributes such as $totalTime$, $avgTime$ and $stdevTime$ is illustrated in Fig. 4. $totalTime$ is approximated by the interval between the first and the last log entry of the session while the $avgTime$ and $stdevTime$ attributes are computed based on the intervals between two successive HTML requests in the session.

The *width* and *depth* attributes are computed from the graph representing all the HTML requests in a particular session. 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. For example, if a session contains the following requests, { (/A,-), (/A/B,/A), (/A/B/C,/A/B)}, then the session's *width* is 1 while its *depth* is 3. As another example, a session that contains requests for {(/A,-) (/A/B,/A), (/C,-) (/D,-) } will have a *width* of 3 and a *depth* of 2.

$MultiIP$ and $MultiAgent$ are two binary features that are used to indicate whether a session contains multiple IP addresses or user agents. The rest of the attributes in Table V are self-explanatory.

3.3. Session Labeling

We use the common robot detection techniques described in Section 2.1 to determine the class label of each session. The session labeling algorithm is summarized in Table VII:

1. If a session s contains a request for the robots.txt file, then the session is declared to be a robot session (denoted as $s.Class = 1$).

Table V. Summary of attributes derived from the Web server sessions. These attributes are used for class labeling (denoted as Classify) and representing the access features of a session (denoted as 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) requested.	Feature
5	robots.txt	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	$totalTime$	approximated total time of the session.	Feature
12	$avgTime$	average time between two HTML requests.	Feature
13	$stdevTime$	standard deviation of time between requests	Feature
14	$Night$	for requests made between 12am to 7am.	Feature
15	$Repeated$	fraction of repeated 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 HTML requests).	Ignore
24	referrer = "-"	% of requests with unassigned referrer	Classify
25	$MultiIP$	indicates whether session contains multiple IP	Feature
24	$MultiAgent$	indicates whether session contains multiple agents	Feature

2. We label the rest of the sessions according to their user agents. In order to do this, we have divided the user agents into 4 categories: Type 1 (known robots), Type 2 (known browsers), Type 3 (possible robots) and Type 4 (possible browsers). This categorization is done semi-automatically. Type 3 agents contain agent names that would suggest that they are likely to be robots, while Type 4 agents are possible browsers or helper applications invoked by the browsers. Table VI shows some examples of the different types of agents. In our experiments, we label sessions with Type 3 agents as

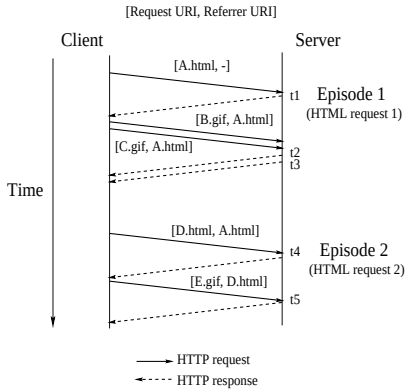

Figure 4. There are two requests in this session. $t1$, $t2$, $t3$, $t4$ and $t5$ are the timestamps recorded in the server logs. The total time of the session is $t5 - t1$, while the period between the two requests is $t4 - t1$.

Table VI. Examples of different types of user agents.

User agent	Type
ArchitextSpider	Type 1
Mozilla/4.0 (compatible; MuscatFerret/2.0; http://www.webtop.com/)	Type 1
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

robots ($s.Class = 1$) while those with Type 4 agents as non-robots ($s.Class = 0$).

3. For sessions that contain multiple user agents, we use a labeling scheme that favors non-robot as opposed to robot sessions. There are two main reasons for doing this. First, we observed that the majority of these multi-agent sessions contain combinations of Type 2 and Type 3 agents, or Type 3 and Type 4 agents. Further analysis showed that these sessions are due to Web users who invoke a helper application while surfing the Web. In the following example, Go!Zilla is a download manager program used by the Web user to retrieve a zipped file:

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)"

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 several multi-agent type sessions with combinations of Type 1 and Type 2 agents. A typical example of a Type 1 user agent found in this combination is "Java1.1" [4]. Upon further investigation, we found that these multi-agent sessions are created by Web browsers that are accessing the HTML pages containing Java applets. In fact, our session labeling algorithm will classify all multi-agent sessions as non-robots except for those with combinations of Type 1 and Type 3 agents.

4. 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 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 client is a Web robot, as long as the number of requests is large. Otherwise, the session is more likely created by a Web browser. This is because a Web browser can generate requests with unassigned referrer values when a user submits a URI from the address window or clicks on a bookmark entry (these are known as user-input clicks). When the number of requests is large, the probability that a Web browser generates only user-input clicks is low. Thus, by selecting an appropriate threshold t on the minimum number of requests, one can potentially identify new robot sessions.

3.4. Classification

After deriving the session features, classification models are built using the C4.5 decision tree algorithm (Quinlan, 1993). There are two main objectives we would like to achieve: (1) to find a good model for predicting Web robot sessions based upon their access features, and (2) to be able to detect robotic activities as soon as possible, with reasonably high accuracy.

Table VII. 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 |Agents| = 1 then
6.         if Type1 ∈ AgentTypes or Type3 ∈ AgentTypes
7.             then s.Class = 1
8.             else s.Class = 0
9.    else     session contains multiple agent types
10.        if (Type2 ∈ AgentTypes or Type4 ∈ AgentTypes)
11.            then s.Class = 0
12.            else s.Class = 1
13.    if s.Class = 0 and |s.list| > t then
14.        if HEAD(s) = 100% or UnassignReferrer(s) = 100%
15.            then s.Class = 1
16. end;
```

There are various metrics we can use to evaluate the performance of a classifier. 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 by using techniques such as bagging and boosting. This issue is described further in Section 4.1. In the area of information retrieval, recall and precision are two popular metrics for evaluating the performance of 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 (Van Rijsbergen, 1979).

Given the objective of detecting Web robot activities as early as possible, it is necessary to build the classification models incrementally (after each request) and determine the minimum number of requests needed to identify robot sessions accurately. In order to do this, we need to generate the data sets for each request. For example, the data set for one request is generated from all the sessions because each session

has at least one episode. If a session contains only a single request, we use all the information within the session to create its feature vector. However, if the session contains more than one request, we should use only the session information up to its first HTML request. As another example, for the data set with two requests, we ignore all the sessions that have only one request, and consider only those sessions with at least two HTML requests. Sessions having more than two requests are then truncated to ensure that their feature vectors contain only the session information up to their second HTML request. This procedure is repeated until there are very few sessions left to create a new data set (we have created data sets up to 7 requests in our experiments).

3.5. Identifying Mislabeled Sessions

The overall performance of a classifier depends primarily on the choice of the learning algorithm and the quality of the training data. Evaluation of the various learning algorithms on this data set will not be discussed here because it is not the main focus of this paper. Instead, we turn our attention to the problem of identifying mislabeled sessions in order to improve the quality of the training data as well as to assess the effectiveness of our session labeling scheme. In particular, we are interested to know the main reasons for the sample mislabeling - are they due to poor choice of labeling heuristics, insufficient information in the data, or misleading information in the data? In this section, we describe a technique for identifying mislabeled sessions using an ensemble filtering approach (Brodley and Friedl, 1999). The idea here is to build multiple classifiers from the training instances and use their misclassification errors to identify the mislabeled samples. In our approach, the misclassification errors are also weighted according to the classifiers' accuracies.

Our technique uses the classification models built from all the attributes given in Table V. This includes the attributes that are used to determine the class label of the session (e.g. robots.txt and % HEAD request). Since the C4.5 algorithm prunes the decision trees during model building in order to avoid overfitting, the leaf nodes of the trees contain a probability distribution for each class. We denote these probabilities as $P(0|X)$ and $P(1|X)$, where $P(i|X)$ is the probability that the sample X belongs to class i.

Suppose there is an ensemble of k classifiers, $C_1, C_2, \cdots, C_k$, built from the training data. Let $t(X)$ be the true class of sample X according to the session labeling heuristics, while $c_m(X)$ is the class label assigned to X by the classifier C_m. Furthermore, let A_m denotes the accuracy of classifier C_m.

Using the above definitions, we define the false positive $FP_m(X)$ and false negative $FN_m(X)$ scores for each sample X and classifier C_m according to the following formulas:

$$FP_m(X) = \begin{cases} 0 & \text{if } t(X) = c_m(X), \\ A_m \times |P(c_m(X)|X) - P(t(X)|X)| & \text{if } t(X) \neq c_m(X) \text{ and } t(X) = 0. \end{cases} \quad (3)$$

$$FN_m(X) = \begin{cases} 0 & \text{if } t(X) = c_m(X), \\ A_m \times |P(c_m(X)|X) - P(t(X)|X)| & \text{if } t(X) \neq c_m(X) \text{ and } t(X) = 1. \end{cases} \quad (4)$$

The overall false positive or false negative score of a sample X is given by $FP(X) = \sum_{m=1}^{k} FP_m(X)$ and $FN(X) = \sum_{m=1}^{k} FN_m(X)$. A large $FP(X)$ score indicates that the session is currently assigned to be a non-robot even though the classification models suggest that it is actually a robot. If we examine the log entries for these false positive sessions, we can verify whether they are indeed non-robots or are being mislabeled. Later, we will show that many of the sessions with high false positive scores are indeed mislabeled due to the presence of camouflaging and previously unknown robots.

Meanwhile, sessions with large $FN(X)$ scores are robot sessions that are often misclassified as non-robots. Since our session labeling algorithm is stricter towards robots than non-robots, there is less opportunity for a non-robot to be mislabeled as robots. Thus, the false negative sessions are more likely due to misclassification by the learning algorithm rather than due to mislabeling of the training samples.

4. Experimental Evaluation

4.1. Experimental Data Set

The experiments described in this section were performed on Web server logs recorded from January 1st until January 31st, 2001 at the University of Minnesota Computer Science department. We have consolidated the logs for the two Computer Science department Web servers, at `http://www.cs.umn.edu` and `http://www-users.cs.umn.edu`. Log entries that correspond to redirection requests from one server to the other are removed to prevent duplication. The consolidated Web logs contain a total of 1,639,119 entries. After preprocessing, 180,602 sessions are created, and the distribution of robot and non-robot sessions are shown in Table VIII.

The overall data set is then partitioned into two groups: G1, which contains all the training instances of known (Type 1 and Type 2) user

Table VIII. Session distribution for various number of requests.

# Request	# Type 1 Agents	# Type 2 Agents	# Type 3 Agents	# Type 4 Agents
1	8040	165835	2805	3922
2	3075	49439	550	1201
3	2050	30277	247	803
4	1627	21436	159	639
5	1415	15618	100	523
6	1091	12108	74	424
7	908	9775	59	379

Table IX. Summary of the dataset description.

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.

where:
G1 : contains Type 1 (known Robot) and Type 2 (known Browser) agents.
G2 : contains Type 3 (possible Robot) and Type 4 (possible Browser) agents

agents, and G2, which contains all the instances of Type 3 and Type 4 user agents. Classification models are built using samples from G1 only (i.e. clean data, denoted as E0), or a mixture of G1 and G2 (i.e. noisy data, E1). We repeat the sampling and model building procedure 10 times for each data set. A summary of the dataset description is given in Table IX.

In order to account for the unequal sizes of robot and non-robot sessions, we weighted the training and test sets to ensure that robot and non-robot sessions have equal representation. For instance, suppose a (training or test) data set contains 100 robots and 1000 nonrobots, then each robot session is duplicated 10 times to ensure that both classes have equal distribution during model building. This procedure is called stratification by oversampling. We have also conducted similar experiments using the full unweighted data set, along with other stratification strategies (e.g. stratification by undersampling). The results for the stratification by undersampling are quite similar to the one presented in this paper, while the results for the unweighted data are

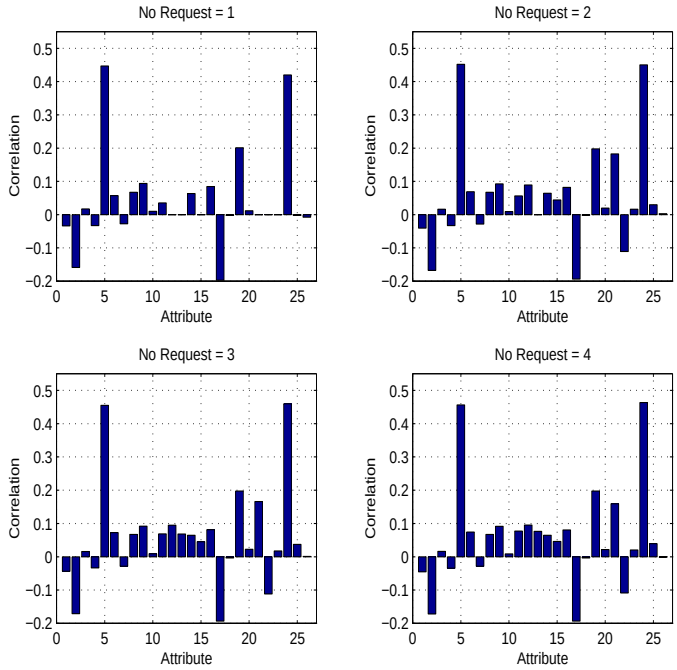

Figure 5. Correlation between each access attribute and the robot class label for various number of requests. The horizontal axis corresponds to the attribute Ids given in Table V.

less conclusive[5]. We have omitted the discussion of these results due to the lack of space.

4.2. Correlation Analysis

Our initial task is to determine whether there exists an attribute that correlates highly with sessions due to Web robots. Figure 5 shows the correlation of each attribute as the number of requests increases from 1 to 4:

1. As expected, the attributes used to determine the class labels (i.e. robots.txt, % HEAD and % unassigned referrer) have very strong positive correlation with the robot sessions. Although their correlations are relatively large, none of them exceeds 0.5. This suggests that using any of these attributes alone are insufficient to detect Web robots effectively. More importantly, the values of these attributes can be easily manipulated by robot designers to hide their true identities. For example, a robot designer can easily create a Web robot that does not use the HEAD request, does not access the robots.txt file, but assigns a referrer value to each of its request messages.

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

3. After two requests, the $avgTime$, $width$ and $depth$ attributes become more significant. The width is positively correlated with robot sessions, whereas the depth attribute is negatively correlated. This confirms our previous observation that many robots, especially the search engine ones, use a breadth-first search strategy to retrieve files from a Web site. Also, notice that the $MultiIP$ attribute is positively correlated, due to robots that parallelize their retrieval operations.

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 tend to have more irregular periods between HTML 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 robot sessions. 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[6]. Thus, it is surprising that the $Night$ attribute is positively correlated with robot sessions. Upon closer examination, we found that this is because most robot sessions are long, spanning into the 12am to 7am time window (which was used to determine the value of 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 $Night = 1$.

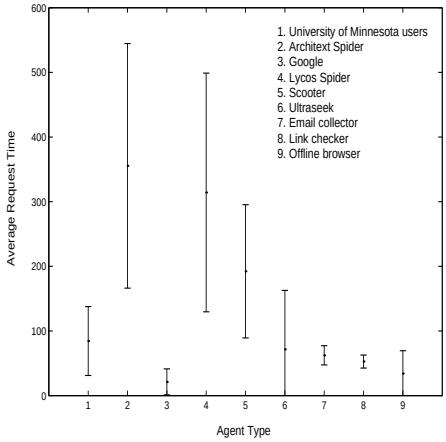

Figure 6. The mean *avgTime* and *stdevTime* attribute values for various Web clients.

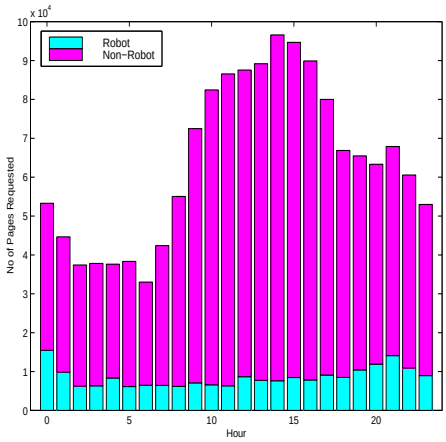

Figure 7. Summary of hourly Web traffic at the University of Minnesota Computer Science department Web server.

4.3. Classifier Performance

Figure 8 shows the classification accuracies of the various models induced from our data sets. The results show that after four requests, we can obtain models with an average accuracy close to 90%. Also, their precision and recall (Figure 9) consistently stay above 82 % and 95% respectively, after more than three requests. While the addition of noisy data (for E1) does not degrade the classifier precision by much, the recall decreases by as much as 5%. The small difference between E0 and E1 curves for large number of requests can be explained by the small number long sessions with Type 3 and Type 4 agents in the data sets (Table VIII).

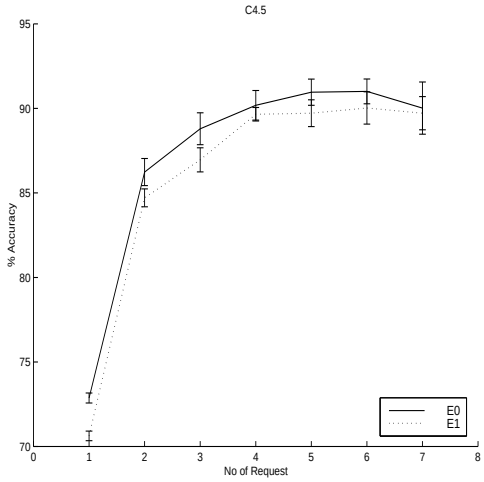

Figure 8. Accuracies of classification models for different number of requests.

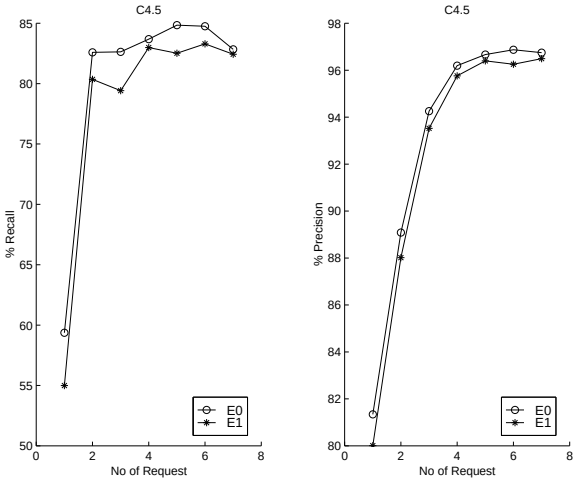

Figure 9. Recall and Precision of classification models for different number of requests.

There is a dramatic improvement in all three performance measures when the number of requests increases from one to two. This is due to the fact that attributes such as *avgTime*, *width*, *depth* and *Repeated* become meaningful after one request.

C4.5 also provides an auxiliary program to generate classification rules from the decision trees. Table X presents some of the high-confidence rules for the robot classes. Indeed, most of the rules seem to agree with our initial correlation analysis. For the data set with one request, the best rules for predicting robot sessions tend to have many attributes in their antecedent. These rules tend to overfit the data, which explains

Table X. Some of the high-confidence decision rules produced for different number of requests.

# Requests	Induced rules [Confidence]
1	$Night = 1$, $totalPages > 3$, % Image ≤ 0.0026, % Binary Exec ≤ 0.9285, % HTML ≤ 0.0526, % Ascii ≤ 0.9, $MultiAgent = 0$, % Other $\leq 0.0312 \longrightarrow$ **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%]

the low recall of the results. Additionally, most of these rules predict robots based on the absence of requests for image files.

For the data set with two requests, the *avgTime*, *width* and *depth* attributes help to improve the accuracy of predicting non-robot sessions. In the example rule given in Table X, 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. Also, as the number of requests increases, shorter rules involving the width attribute begin to emerge.

4.4. Finding Mislabeled Data

We have also conducted a series of experiments to identify sessions that are often misclassified by the classification models. We use the technique described in Section 3.5 to identify sessions with high false positive and false negative scores. Figure 10 shows a summary of the log entries for some of the sessions with the highest false positive scores (i.e. sessions that are predicted to be robots but labeled as non-robots). We observe that most of these sessions are indeed mislabeled due to camouflaging or previously unknown robots.

1. The first session contains a user agent that is very similar to a conventional browser. However, the session seems to cover a significant portion of the Web site without focusing on any specific topic, which is why there is a strong possibility that this session belongs to a Web

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	/~wijesek/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	/~ssparikh/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	/~tjiang/resume.html	Mozilla/4.0 (compatible; MSIE 5.01; Windows 98; DigExt)
rfx-64-6-194-38.users.reflexcom.com	15:03:45	/~littau/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.

robot. Upon resolving the hostname of the session, our suspicion is confirmed because NEC Research, which owns the domain name, is known to have a Scientific Bibliography Search Engine.

2. The second example also looks highly suspicious despite having a user agent declared as Microsoft Internet Explorer. It is very unlikely that a user will be able to view all four HTML pages in one second. Unfortunately, we were unable to resolve their IP address to determine the origin of the client.

3. The third example is interesting because the pages retrieved by the session are mostly resume files. Our classifiers were able to detect the large width of the traversal to infer that this session is very likely due to accesses by a 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 due to a resume hunter robot. However, it is interesting to note that the domain name of the client belongs to a broadband Internet Service Provider. Thus, the traditional techniques of inspecting the user agent and IP address field of a Web client may not be able to detect this robot.

On the other hand, there are very few mislabeled non-robot sessions with high false negative scores (cf. Figure 11). Only one of the four sessions in this figure is clearly mislabeled. `SilentSurf` (i.e. the fourth session) was initially thought to be a Type 3 robot. However, many of our classification models have identified it to be a non-robot. Upon

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	/~ssparikh	SilentSurf/1.1x [en] (X11; I; $MyVersion)
63.87.244.21	4:45:05	/~ssparikh/images/headsil.jpg	SilentSurf/1.1x [en] (X11; I; $MyVersion)
63.87.244.21	4:45:06	/~ssparikh/images/back/ivy.gif	SilentSurf/1.1x [en] (X11; I; $MyVersion)

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

further analysis, we found that `SilentSurf` is in fact an anonymizer Web site that changes the user agent of a browser into a robot-like value. The rest of the false negative sessions in this figure could be due to misclassifications by our models. These sessions contain mostly robots whose navigational patterns are very similar to human users (e.g. offline browsers) and download utility programs that have very short session lengths (which makes it difficult to predict correctly their class labels).

5. Conclusion

While our initial correlation analysis suggests that many of the attributes used to label the Web sessions (e.g. robots.txt, % HEAD requests and % unassigned referrers) are good robot predictors, these attributes alone are often insufficient to detect Web robots. Instead, we have strived to build accurate predictive models for Web robots by using access features derived from the navigational patterns of the various Web clients. The results of our experiments on the Computer Science department Web server logs suggest that Web robots can be detected with more than 90% accuracy after 4 requests. These classification models can be used to detect camouflaging robots that have very similar navigational patterns as other known robots.

Throughout this paper, we assume that the navigational patterns of Web robots are significantly different than those of human users. In fact, we have observed that Web robots traversing a Web site with the same information need will exhibit similar navigational patterns. This suggests that perhaps we may need to construct different classification

models for each type of robot in order to achieve better predictive accuracy. We hope to explore this possibility in our future work.

Our session labeling procedure may introduce some errors into the classification models. We have observed that the labeling errors are more prevalent for robot sessions compare to non-robot sessions since our session labeling algorithm is biased towards non-robots. Another future direction of this research will be to consider some of the existing analytical models for quantifying the errors due to mislabeled examples (Brodley and Friedl, 1999).

We also need to refine the features used for building the classifiers. For example, we can incorporate the Web content and structure information into the session features. Further investigation is also needed to study the effect of other types of navigational patterns not captured by our data.

Acknowledgements

This work was partially supported by NSF grant # ACI−9982274 and by Army High Performance Computing Research Center contract number DAAH04−95−C−0008. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by AHPCRC and the Minnesota Supercomputing Institute.

Notes

[1] An auction aggregator collects information from various on-line auction sites and lists the integrated results at their own Web site. As a result, prospective buyers can buy products from auction sellers without visiting the site where the auction was originally posted. This is of great concern to many auction site operators because buyers and sellers may eventually stop visiting their Web site and use the services of aggregator sites instead.

[2] The anonymizer Web site SilentSurf is located at `http://www.noproxy.com`, while the Turing Machine Web site is located at `http://anonymizer.com`.

[3] For example, the Web Robot Database located at `http://info.webcrawler.com/mak/projects/robots/active/`.

[4] 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.

[5] A full discussion of these results are reported in our technical report.

[6] The observed traffic pattern is very similar to the e-commerce traffic observed by Rosenstein in (Rosenstein, 2000).

References

Balabanovic, M. and Shoham, Y. (1995). 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*.

Brodley, C. and Friedl, M. (1999). Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167.

Clark, D. (2000). Shopbots become agents for business change. *IEEE Computer*, pages 18–21.

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

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

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

Graham, L. (2000). Keep your bots to yourself. *IEEE Software*, 17(6):106–107.

Gray, M. (1993). Measuring the growth of the web. http://www.mit.edu/people/mkgray/growth/.

Jackson, S. (1998). Building a better spider trap. http://www.spiderhunter.com/spidertrap.

Kephart, J. and Greenwald, A. (1999). Shopbot economics. In *Agents*, 378–379.

Kolar, C., Leavitt, J., and Mauldin, M. (1996). Robot exclusion standard revisited. http://www.kollar.com/robots.html.

Koster, M. (1994a). Guidelines for robot writers. http://info.webcrawler.com/mak/projects/robots/guidelines.html.

Koster, M. (1994b). A standard for robot exclusion. http://info.webcrawler.com/mak/projects/robots/norobots.html.

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

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

Pirolli, P., Pitkow, J., and Rao, R. (1996). Silk from a sow's ear: Extracting usable structures from the web. In *CHI-96*, Vancouver.

Quinlan, J. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

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

Van Rijsbergen, C. J. (1979). *Information Retrieval*. Butterworths, London.

Yoon, M. (2000). Web robot detection. http://www.arsdigita.com/doc/robot-detection.