Log Pre-Processing and Grammatical Inference for Web Usage Mining

Thierry Murgue

Log Pre-Processing and Grammatical Inference for *Web Usage Mining*

Thierry Murgue[1,2]

[1] EURISE, University of Saint-Étienne 23, rue du Dr Paul Michelon
42023 Saint-Étienne cedex 2 – France
[2] RIM, École des Mines de Saint-Étienne 158, Cours Fauriel
42023 Saint-Étienne cedex 2 – France
thierry.murgue@univ-st-etienne.fr

Abstract. In this paper, we propose a Web Usage Mining pre-processing method to retrieve missing data from the server log files. Moreover, we propose two levels of evaluation: directly on reconstructed data, but also after a machine learning step by evaluating inferred grammatical models. We conducted some experiments and we showed that our algorithm improves the quality of user data.
Keywords: log pre-processing, web usage mining, grammatical inference, evaluation

1 Introduction

Web Usage Mining is a complex process used in order to extract knowledge about users of a web site. It is composed of many steps as described in Fig.1, from selecting relevant data to knowing how users browse on a web site. Web Usage Mining was first introduced in 1997 [1] as the "discovery of user access patterns from Web servers". To select, to clean and to format available data are real challenges. The kind of information which is the most used in this context is the Web server log files. Already in 1995, Catledge and Pitkow used logs for the characterization of "Browsing Strategies" [2]. Other types of data can be used in this topic of research, such as client data (mouse gestures, keyboard events,...) or user physical behavior (eye movements, arm gestures,...), but they are really hard to obtain. As a consequence, most work in this research field depends on server log files. Due to some network architectures and features (*cache* and *proxy*), log data are often irrelevant and noisy. Pre-processing is therefore a crucial issue for learning. Machine learning methods for Web Usage Mining include frequent sequences learning [3, 4] and more structural models such as Hidden Markov Models (Hmms) [5, 6]. More recently, researchers have worked on grammatical models: n-grams [7], and stochastic automata with classical grammatical inference methods [8].

In this paper, we propose a method for handling the problem of noisy and irrelevant data in log files. We present an algorithm to reconstruct the data and we evaluate it. We use stochastic inference to learn users behaviors and we show that reconstructed data leads to better models.

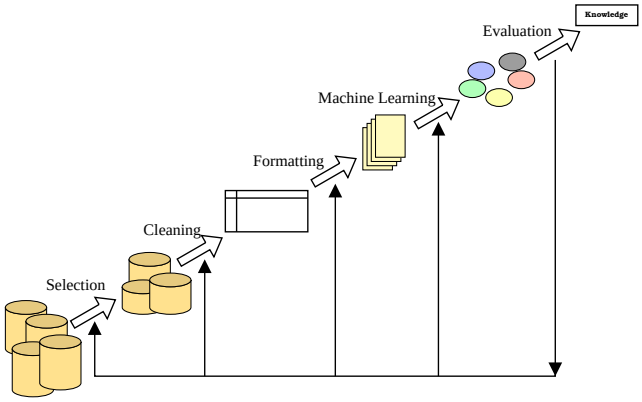


Fig. 1. Web Usage Mining process

We will first present the problem we want to deal with. The description of an algorithm to reconstruct the log data in order to obtain reliable ones follows: the method depends on some heuristics which try to detect improbable paths on navigation. Two sets of experiments show that we can retrieve some pieces of information which were lost due to caches. Finally, we conclude and propose alternative protocols in order to test log processing methods.

2 Our Problem

In order to mine user behavior, we have to extract from log files each visit of users to the web site: a visit is considered as a set of pages requested in a single semantic goal (without any embedded objects such as pictures and so on). We will see that they are missing and erroneous data in server log files. Before starting to explain our problem, we have to say more about logs.

The World Wide Web Consortium (W3C) recommends to use the *common log file* format to record required fields [9].

Table 1. a common log record

F1	F2	F3	F4	F5	F6	F7
161.3.6.51	-	-	[30/Oct/2001:20:13:27 +0100]	"GET / HTTP/1.1"	403	293

Generally, a log record is composed of (see Table 1):

- `remotehost` field (F1): the address of the machine which makes the request.
- `rfc931` field (F2): the remote login name of the user. Actually, for obvious security reasons, this field is almost always blank.

– `authuser` field (F3): the username as which the user has authenticated himself. This field is only filled when an authentication process is used.
– `date` field (F4): data and time when the server receives the request.
– `request` field (F5): the request line exactly as it came from the client.
– `status` field (F6): a return code informing the client of the request processing status.
– `bytes` field (F7): the content-length of the answer transferred.

Servers record chronologically each hit into the log files: a hit can be a real page request, but also an error or an embedded document request which is not related to the navigation of the user. First of all, we have to filter the logs, by deleting all useless items such as error reporting, non-supported requests (for example, in our preliminary version we did not process any page with parameters as `index.php?id=12&it=la`). The result of filtering raw data is a sequence of logs, which represents only a part of user navigation: actually, many requests never reach the server.

In order to reduce network traffic and to speed up the transfer of data on the Internet, the almost totality of browsers (mozilla-firefox, opera, IE,...) implement a local cache. In [10], caching is defined as "a mechanism that attempts to decrease the time it takes to retrieve a resource by storing a copy of the resource at a closer location". Consequently, when a user requests a page:

1. the cache checks if it has already been viewed by the user (typically when user has hit *back button*);
2. (a) if not, the client sends the request to the server;
   (b) else, the cache produces the answer but the server never receives the request and so can't record it.

This is the first problem due to the network features: it implies that some data is missing in server log files.

Another place where a cache can be, and it is called *global* cache in this context, is on a *proxy* server: a proxy is a machine somewhere in-between client and server; its goal is to increase the security and to make easier the administration of a machines set (a domain) by allowing only connection to external web servers from the proxy. Thus, a machine from the protected domain can request a page only to the proxy:

1. the proxy checks in its own cache, if it has already been viewed by one of the users using the proxy;
2. (a) if not, the proxy forwards the request to the server: in this case, the *from field* recorded into the logs is the proxy address and not the client address;
   (b) else, the cache produces the answer but the server never receives the request and so can't record it.

In this configuration, external servers receive requests only from the proxy, so in the log files, we can't distinguish where the request comes from.

So, to retrieve the exact visits of each user, we have to deal with different kinds of noise: erroneous data, when the from address is a proxy and not the real user's machine and missing data, when the cache instead of the server supplies the answer.

3 Reconstructing data: algorithm WhichSession

In order to retrieve the missing data and correct the wrong data, we suppose that the web site corresponding to the server is known: we model the web site by a graph where nodes represent pages and links represent hypertext links into the page. The heuristics we use detect inconsistent sequences of pages in the user navigation: the algorithm WhichSession is described below.

Algorithm 1: WhichSession

```
Data: h(0)  //page we want to class
foreach k ∈ opened_sessions do
    if h_k(1) → h(0) then
        Store(k, h(0))  /*add h(0) in session k and exits */
min_back_length = ∞;
//the number of backward steps, we have to follow to find a linked page
foreach k ∈ opened_sessions do
    for 2 ≤ i ≤ |h_k| do
        if h_k(i) → h(0) then
            if min_back_length > i then
                min_back_length = i ;
                min_session = k; break;
if min_back_length < ∞ then
    /*we found an inconsistency into log records, we have to include a part of
      the history in order to have a linked path into the web site */
    Add_Missing_Values(min_session, min_back_length);
    Store(min_session,h(0));
else
    /*there's no available page in histories: add a new session */
    l = New_Session();
    Store(l,h(0));
```

The symbol $\rightarrow$ is used to indicate that there is a hypertext link from the first page to second one. Some notations are to be defined: for the situation at the current time, there are some sessions opened and we want to store the current log record $h(0)$ into the right session, the one corresponding to the unique user

who requests the page. We have to select the best session, by scrolling through the history of opened sessions (h_k is the history of the session k, $h_k(i)$ is the ith page of h_k), and choosing the one which has the minimum backward steps required to find a link to the current page.

4 Artificial data experiments

4.1 Artificial Data presentation

In order to have reliable data to test our reconstructing method, we generated artificial logs according to possible paths in a web site. We select a page and we compute a path with some "navigation errors" that could typically have been produced by hitting the *back button* or no taking the shortest path. The site is composed of 105 different pages, the average length of generated paths is 10.9 pages. At this step, we obtain a reference complete set of logs l_0 without erroneous data. Then we simulated some caches from which we obtained l_1 and finally rebuilt the data l_2 from l_1 using algorithm WhichSession.

4.2 Data evaluation

We first carried out a set of experiments on these data by computing two ratings between l_0 and l_1 on one hand, and between l_0 and l_2 on the other hand: one measures the difference between the number of detected visits ($d_\#$), the other the Levenshtein's distance (d_L) [11] for each visit. In order to do that, we have to re-assign logs into their reference visit because of the lack of detected visits.

As an example, if we have a set of three visits (v_i) with two pages (A and B) into the reference artificial data such as:

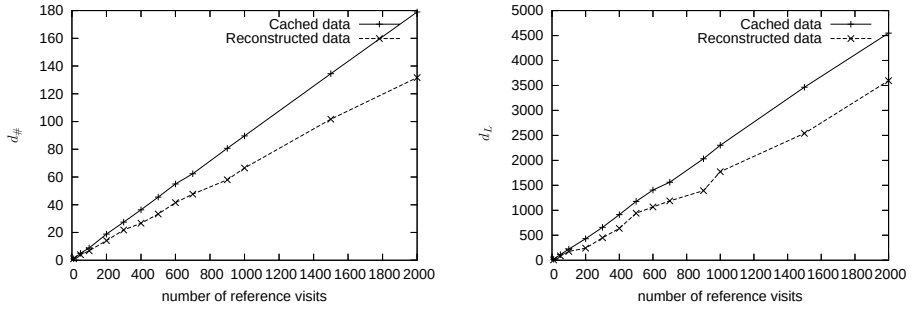
reference visits l_0	cached ones l_1	reconstructed ones l_2
$v_0 = [A, B, A]$	$v_0 = [A, B, B, A]$	$v_0 = [A, B]$
$v_1 = [B, B, A]$	$v_1 = [B]$	$v_1 = [B, B, A]$
$v_2 = [A, B]$		$v_2 = [A, B]$

One can compute $d_\#(l_0, l_1) = 1$, $d_\#(l_0, l_2) = 0$. By comparing date of logs (it is possible because we have generated the data), we can reorganize l_1 into $v_0 = [A, B]$, $v_1 = [B, A]$ and $v_2 = [A, B]$ and then compute $d_L(l_0, l_1) = 3$, $d_L(l_0, l_2) = 1$.

Results of the experiment are shown in Fig. 2.

We can see that the two ratings are better when the data are reconstructed by our method. Actually, reconstruction permits to retrieve correct missing data: that helps to detect the end of a visit, and gives better knowledge about the navigation.

This protocol allows testing the method only if we know exactly the initial visits without any kind of noise. With real data, we can't have this *a priori* knowledge, so we decided to learn grammatical models and to test them.



(a) $d_{\#}$ (b) d_L

Fig. 2. $d_{\#}$ and d_L behaviors

4.3 Models evaluation

Data for this set of experiments are the same as for the preceding one: we generated them. Here, instead of computing ratings on cached and reconstructed visits, we used them to learn grammatical models. We learn stochastic models which can predict the next most probable symbol of a sequence: here sequences represent visits and symbols are pages. We use the MDI [12] algorithm to learn such a model for the cached data and such a model for the reconstructed ones.

We generated also a sample set of ideal visits: for each visit step, we compared the real next page in the visit with the most probable one proposed by the model. We count a success if the pages represent the same one, and a failure otherwise.

Results are shown in Fig. 3.

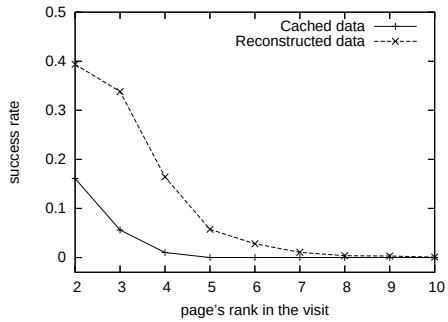


Fig. 3. Success rate in learning task on cached and reconstructed data

As presented in [4], it is a hard problem to predict a page after the few first ones, so usually authors do not try to check after the 7th page and their success rate really depends on the number of site pages. In our case, on different data, we can see that without any reconstruction, after the 5th page, we can not predict anything. But, with our method we learn better models and we can predict with a higher probability the next page. At each fixed position in the visit, we do better prediction, and for a fixed success rate, we can predict at a further position.

5 Real data problem

Experiments on artificial data can help us to test the reconstruction of data, but the main problem is to mine user behavior, so we have to experiment also on real data.

For that, we took data from a real server log file ($\approx$ 137000 logs, 100 pages, mean of 11.4 links per page) [13]. Then, we extracted visits from these data and also from reconstructed ones by our method. Here the protocol is quite the same as for artificial data: we learned one model for each kind of data (raw and reconstructed), and we evaluated models.

The main problem in this context is that we do not have perfect data to use as a sample set. We decided here to test with two different sets: one which is composed of a part of the log file (not used in learning step) for testing the model on raw data, and the other is the reconstruction of this same part by our method for testing model on reconstructed data.

We describe in Fig. 4 the results.

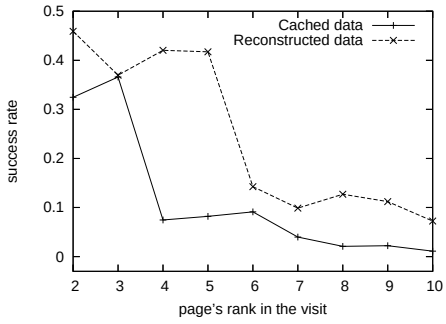


Fig. 4. Success rate in learning task on raw and reconstructed data

When we use reconstruction, results are better. Actually, we can view the two types of data as two different kinds of answers for the problem: learning model on raw data comes to predict the next page requested for the servers, taking into account not only the user behavior but also the disruptive caching

features; the model on reconstructed data can be viewed as a true user behavior model without any other kind of perturbation.

6 Conclusion

In this paper, we showed on artificial data that reconstructing log files data can avoid some perturbations due to caching features. With both types of evaluation, ratings on data and prediction evaluation, results are better with reconstruction.

On the real data, there remains a problem: which data can be used to evaluate models? We do not have perfect data to test the quality of the reconstruction: testing on reconstructing sample set introduces a small bias. In an optimal context, we have to have complete data and cached ones in order to test exactly the predictions model.

Results given by grammatical inference are really interesting: they are as good as other types of models shown in state of the art for the usage mining. We want to continue in this field by showing that these models which permit long term dependence are really good for this task.

References

1. Cooley, R., Srivastava, J., Mobasher, B.: WEB Mining: Information and Pattern Discovery on the World Wide Web. In: Proceedings of ICTAI'97. (1997)
2. Catledge, L.D., Pitkow, J.E.: Characterizing Browsing Strategies in the World-Wide Web. Computer Networks and ISDN Systems **27** (1995) 1065–1073
3. Frias-Martinez, E., Karamcheti, V.: A Prediction Model for User Access Sequences. In: WEBKDD Workshop: Web Mining for Usage Patterns and User Profiles. (2002)
4. Géry, M., Haddad, H.: Evaluation of Web Usage Mining Approaches for User's Next Request Prediction. In: Proc. of WIDM'03, New Orleans (2003) 74–81
5. Pitkow, J., Pirolli, P.: Mining Longest Repeating Subsequences to Predict World Wide Web Surfing. In: Proceedings of USITS'99. (1999)
6. Bidel, S., Lemoine, L., Piat, F., Artières, T., Gallinari, P.: Statistical Machine Learning for Tracking Hypermedia User Behaviour. In: MLIRUM - UM Workshop, Pittsburgh (2003)
7. Borges, J., Levene, M.: Data Mining of User Navigation Patterns. In: WEBKDD. (1999) 92–111
8. Karampatziakis, N., Paliouras, G., Pierrakos, D., Stamatopoulos, P.: Navigation Pattern Discovery Using Grammatical Inference. (In: Proc. of ICGI04) 17–56
9. Luotonen, A.: The Common Log File Format (1995) http://www.w3.org/Daemon/User/Config/Logging.html.
10. Pitkow, J.: In Search of Reliable Usage Data on the WWW. In: Proceedings of the Sixth International WWW Conference, Santa-Clara, CA (1997) 451–463
11. Levenshtein, V.I.: Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics - Doklady **10** (1966) 707–710 Translated from Doklady Akademii Nauk SSSR, Vol. 163 No. 4 pp. 845-848, August 1965.
12. Thollard, F., Dupont, P.: Entropie relative et algorithmes d'inférence grammaticale probabiliste. In: Conférence sur l'apprentissage automatique, CAP'99, Paris (1999) 115 – 121
13. Eurise: Web site. http://eurise.univ-st-etienne.fr (2002)