Web Log Mining by an Improved AprioriAll Algorithm

WANG Tong¹ HE Pi-lian²

Abstract—This paper sets forth the possibility and importance about applying Data Mining in Web logs mining and shows some problems in the conventional searching engines. Then it offers an improved algorithm based on the original AprioriAll algorithm which has been used in Web logs mining widely. The new algorithm adds the property of the UserID during the every step of producing the candidate set and every step of scanning the database by which to decide whether an item in the candidate set should be put into the large set which will be used to produce next candidate set. At the meantime, in order to reduce the number of the database scanning, the new algorithm, by using the property of the Apriori algorithm, limits the size of the candidate set in time whenever it is produced. Test results show the improved algorithm has a more lower complexity of time and space, better restrain noise and fit the capacity of memory.

Keywords—Candidate Sets Pruning, Data Mining, Improved Algorithm, Noise Restrain, Web Log

I. INTRODUCTION

Web mining deals with the data related to the Web, they may be the data actually present in Web pages or the data concerning the Web activities. The Web can be viewed as the largest unstructured data source available, although the data on the Web sites, which composed them, is structured. This presents a challenging task for effective design of and access to Web pages. Web data can be classified into the following categories [1]:

1) Content of actual Web pages is that to be presented to the users.
2) Intrapage structure includes the HTML / XML code for the pages.
3) Interpage structure is the actual linkage structure between Web pages.
4) Data usage describes how Web pages are accessed.
5) User profiles include demographic and registration information obtained about users. These could also include information found in cookies.

This paper mainly to show the improved AprioriAll algorithm, which has lower complexity of time and space than that of the original one, can be used in Web Mining widely.

II. MINING WITH WEB LOGS

A. Web Logs

Web usage mining looks at logs of Web access. General access pattern tracking is a type of usage mining that looks at Web pages visit history [4]. Web Servers record access information as a click-stream-data into log files. Whenever a Web page is clicked, corresponding data will be generated and recorded. There are valuable information in the profile, such as the access patterns of users, the types of explores and operating system being used, as well as how long a user was on line. Parts of log file of Microsoft Internet Information Server 5.0 is shown as following and its format fits into the format addressed by W3C extension norm[3].

The logs can be examined from either a client perspective or a server perspective. When it is evaluated from a server side, mining uncovers information about the sites where the service resides. It can be used to improve the design of the sites. By evaluating a client’s sequence of clicks, information about a user (or group of users) is detected. This could be used to perform prefetching and caching of pages.

B. The problems be suffer with the conventional search engines

Almost all Web sites have searching function, and the search engines which they are using are confronting with the following troubles[2]:

1) Over abundance: Most of the data on the Web are of no interest to most people. In other words, although there is a lot of data on the Web, an individual query will retrieve only a very small subset of it.
2) Limited coverage: Search engines often provide results from a subset of the Web pages. Because of the extreme size of the Web, it is impossible to search the entire Web at any time a query is submitted. Instead, most search engines create indices that are updated periodically. When a query is requested, only the index is directly accessed.
3) Limited query: Most search engines provide access based on simple keyword-based searching. More advanced search engines may retrieve or order pages based on other properties such as popularity of pages.

4) Limited customization: Query results are often determined only by the query itself. However, as with traditional IR systems, the desired results are often dependent on the background and knowledge of the user as well.

III. THE IMPROVEMENT OF SEQUENTIAL PATTERNS ALGORITHM BASED ON USERS

A. The base principle

The effectiveness of a set of Web pages depends not only on the content of individual Web pages, but also on the structure of the pages and their ease of use. The most common data mining technique used on click-stream-data is that of uncovering traversal patterns. A traversal pattern is a set of pages visited by a user in a session. Knowledge of frequently references of contiguous pages can be useful to predict future references and thus for prefetching and caching purposes. Following it, knowledge of backward traversals can be used to improve the design of a set of Web pages by adding new links to shorten future traversals. The use of such performance improvements as user side caching actually alter the sequences visited by a user and impact any mining of the web logs data at the server side. The maximal profit is used primarily to reduce the number of meaningful patterns discovered. That is the rules, which have been expressed by other more complex rules, can be deleted. The problem of the click-stream-data is that the data of Web logs is asynchronous. Although a record will be written into a Web log file whenever a user click a Web site, monitoring the access pattern of a user is not an easy thing, because the information in a Web log file is recorded in the time order. In order to traverse users click-stream, we can not analyze the Web logs on time order only, users property should be taken into account too, that means we should traversal the Web logs according to the order of timestamp as well as that of UserID.

A sequential pattern (as applied to Web usage mining) is defined as an ordered set of pages that satisfies a given support and is maximal (i.e., it has no subsequence that is also frequent). Support is defined not as the percentage of sessions with the pattern, but rather the percentage of the customers who have the pattern. Since a user may have many sessions, it is possible that a sequential pattern should span a lot of sessions. It also needs not be contiguously accessed pages. A k-sequence is a sequence of length k (i.e., it has k pages in it).

B. The original AprioriAll algorithm

The first step is sort step to put the data in the correct order, that is ordered by UserID and timestamp, the remaining steps are somewhat similar to those of the Apriori algorithm. The difference between the Apriori and the AprioriAll is that when generate the candidate sets, the AprioriAll use full join, that means all the items in L_{k+1} can be joined and put into C_{k}. On the contrary, in the Apriori, all the items in L_{k+1} can only be forth joined. As for a Web user can navigate back or forth, the AprioriAll but not the Apriori be used on Web log mining. The sort step creates the actual customer sequences, which are the complete reference sequences from one user (across transactions). During the first scan it finds large 1-itemset. Obviously, a frequent 1-itemset is the same as a frequent 1-sequence. In subsequent scans, candidates are generated from the large itemsets of the previous scans and then are counted. In counting the candidates, however, the modified definition of support must be used. In the algorithm we show that AprioriAll is used to perform this step.

The original algorithm[1]

Input

\[ D=\{S_1,S_2,\ldots,S_n\} \quad \text{//Database of sessions} \]
\[ S \quad \text{//Support} \]

Output: sequential patterns

SEQUENTIAL PATTERNS ALGORITHM

D=sort D on UserID and time of first page reference in each session;
Find \( L_1 \) in D;
\( L=\text{AprioriAll (}D, S, L_1\); Find maximal reference sequences from \( L \);

The following is an example to show the processing steps.
EXAMPLE The XYZ Corporation maintains a set of five Web pages: [A B C D E]. The following sessions (listed in timestamp order) have been created:
\[ D=\{S_1=\{U_1, A, B, C, E\} \quad S_2=\{U_2, A, C\} \quad S_3=\{U_3, B, C, E\} \quad S_4=\{U_4, A, C, D, E\} \quad S_5=\{U_5, A, C, D, E, C\}\} \]

We have added to each session the UserID. Suppose the support threshold is 50%.

In this example, user \( U_3 \) actually has two transactions (sessions). To find his sequential patterns, We must think of his sequence as the actual concatenation of those pages in \( S_1 \) and \( S_2 \). Also, since support is measured not by transactions but by users, a sequence is large if it is contained in at least one customer’s sequence. After the sort step, we have that
\[ D=\{S_1=\{U_1, A, B, C, E\} \quad S_2=\{U_2, A, C\} \quad S_3=\{U_3, B, C, E\} \quad S_4=\{U_4, A, C, D, E\} \quad S_5=\{U_5, A, C, D, E, C\}\} \]

We find \( L_1=\{<A> \quad <B> \quad <C> \quad <D> \quad <E>\} \)

There are 5 members.

\[ L_2=\{<A, B> \quad <A, C> \quad <A, D> \quad <A, E> \quad <B, C> \quad <B, D> \quad <B, E> \quad <C, D> \quad <C, E> \quad <D, E> \}\]

There are 20 members.

\[ L_3=\{<A, B, C> \quad <A, B, D> \quad <A, B, E> \quad <A, C, D> \quad <A, C, E> \quad <A, D, E> \quad <B, C, D> \quad <B, C, E> \quad <B, D, E> \quad <C, D, E> \}\]

There are 32 members.

International Scholarly and Scientific Research & Innovation 1(4) 2007 1047 scholar.waset.org/1999.4/10434
The candidate sets generated in every step is crossed each other, and the others should not be account. On the other hand, the candidate sets and large sets refers to the UserID. In the user property is forced to take into account, every element in the candidate sets sharply, as well as the number of scanning database are high. The property of the Apriori algorithm, which is that every element’s subset of the items in $L_k$ must be in the $L_{k-1}$, can be used to prune.

In the following improved algorithm, on the one hand, the user property is forced to take into account, every element in the candidate sets and large sets refers to the UserID. In every step to generate a candidate set, the elements in the previous large set which have the same UserID should be crossed each other, and the others should not be account. On the other hand, the candidate sets generated in every step is pruned in the light of the property of Apriori and the result is called $C_k$. By these way, we can reduce the size of every candidate set sharply, as well as the number of scanning database, so the complexity of the time and space can be reduced. The following is the description of this improved algorithm.

The improved algorithm

Input

$\{U_1, U_2, \ldots, U_n\}$ // The set of users

$D=\{t_1, t_2, \ldots, t_k\}$ // Database of sessions with UserID

S // Support

Output: sequential patterns

**SEQUENTIAL PATTERNS ALGORITHM**

D=sort D on UserID and time of first page reference in each session;

$L_1$ with UserID=[large 1-itemssets];

For (k=2; $L_{k-1}$#null; k++) do

Begin

$C_k=$Apriori-gen($L_{k-1}, U_i$); // new candidate set

For all transaction $t_i$ D do

begin

$C_i$=subset($C_k, t_i$);

For all candidate c $C_i$ do

begin

If has frequent-subset(c, $L_{k-1}$)

Cj=subset(c, $L_{k-1}$);

Then return False;

Else True;

The following table outlines the steps taken by the improved algorithm.

$C_1=$\{$A_1, A_2, B_1, B_2, C_1, D_1, E_1, E_2, E_3, E_4\}$ There are 5 members.

$L_1=$\{$A_1, A_2, B_1, B_2, C_1, D_1, E_1, E_2, E_3, E_4\}$ There are 14 members.

$C_2=$\{$A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4, C_1, C_2, C_3, C_4, D_1, D_2, D_3, E_1\}$ There are 25 members.

$C_3=$\{$A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, B_5, C_1, C_2, C_3, C_4, C_5, D_1, D_2, D_3, D_4, D_5, E_1\}$ There are 11 members.

$L_3=$\{$A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, B_5, C_1, C_2, C_3, C_4, C_5, D_1, D_2, D_3, D_4, D_5, E_1\}$ There are 12 members.

$C_4=$\{$A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, B_5, C_1, C_2, C_3, C_4, C_5, D_1, D_2, D_3, D_4, D_5, E_1\}$ There are 4 members.

$C_i=$\{$A_1, A_2, A_3, A_4, A_5, B_1, B_2, B_3, B_4, B_5, C_1, C_2, C_3, C_4, C_5, D_1, D_2, D_3, D_4, D_5, E_1\}$

Obviously, the number of elements in every candidate set here is smaller than that of the original algorithm.
IV. TEST AND RESULT

In order to compare the improved algorithm to the original one, we used a piece of Web logs of www.baidu.com.cn, which about 2.5G, to do the test. The result showing in TABLE I:

<table>
<thead>
<tr>
<th></th>
<th>CPU</th>
<th>Memory</th>
<th>Time cost 1</th>
<th>Memory</th>
<th>Time cost 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>PIII</td>
<td>1G</td>
<td>512K</td>
<td>1G</td>
<td>75' 35''</td>
</tr>
<tr>
<td>improved</td>
<td>PIII</td>
<td>1G</td>
<td>15' 28''</td>
<td>1G</td>
<td>13'16''</td>
</tr>
</tbody>
</table>

The principle for improving is to reduce the number of elements in every candidate set, but any change of the large sets is not allowed. There are three aspects to make this algorithm better than the original one. Firstly, when the candidates are being produced, instead of dealing with all the items of the previous large set, only the elements which having the same UserID are crossed. Secondly, by pruning, the number of elements in the candidate sets is decreased once more. Thirdly, while deciding whether an element is large or not, it only need to scan the turples in the database with the same UserID, by this way, there are only a few records in the database being scanned.

Although the cost of time and space consuming are reduced greatly, but in the real processing, the capacity of the memory should be taken into account either. For the sake of the huge number of the Web logs, there is no memory in the world has enough space to be loaded. This has become one of the reasons for losing. The huge number of the Web logs records produces great press on the CPU too. By thinking of no relationship between the patterns of different users, three methods can be used. The first one is loading all Web log records into SQL Server database during preprocessing, and using the power of the DBMS to store, group, sort and query the data. TABLE II and TABLE III show the Data Directory.

<table>
<thead>
<tr>
<th>Fields Name</th>
<th>Data Type</th>
<th>Length</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>OrderNum1</td>
<td>long</td>
<td>4</td>
<td>PK</td>
</tr>
<tr>
<td>IP</td>
<td>Varchar</td>
<td>50</td>
<td>Not null</td>
</tr>
<tr>
<td>UserID</td>
<td>Varchar</td>
<td>50</td>
<td>Not null</td>
</tr>
</tbody>
</table>

The second one is that, in order to match the capacity of the memory, the data of only one user are loaded for each time. The third one is that, multi-computers can be connected into a net, in which, every computer only process the data of one user on each time and these computers are running on parallel. By these measures, not only the neck of the memory problem, but also the problem of the huge data of Web logs has been ruled out. When a computer is dealing with the data of one user, a small support is used, in order not to lose any patterns of users. But on this way, some noise may be introduced too. On the next step, all of the result of the previous step are collected together into a new data set $D_g$, and use the original support to match the all users’ patterns and lead to the final result. For the support return to the original one, the noise which introduced by the previous step are all ruled out. Moreover, in the case of new data be appended into Web logs, only the new data should be dealt singly and whose result can be appended into $D_g$. Next time, we need only to scan the $D_g$ again. By this way, the result produced by previous processing can be used either, this means time be saved.

REFERENCES