Web Users Clustering

Tadeusz Morzy, Marek Wojciechowski, Maciej Zakrzewicz

Poznan University of Technology
Institute of Computing Science
ul. Piotrowo 3a, 60-965 Poznan, Poland
{morzy,marek,mzakrz}@cs.put.poznan.pl

Abstract. Web log mining is a new subfield of data mining research. It aims at discovery of trends and regularities in web users' access patterns. This paper presents a new algorithm for automated segmentation of web users based on their access patterns. The results may lead to an improved organization of the web documents for navigational convenience.

1 Introduction

The problem of web log mining consists in automated analyzing of web access logs in order to discover trends and regularities (patterns) in users' behavior. The discovered patterns are usually used for improvement of web site organization and presentation. The term of *adaptive web sites* has been proposed to denote such automatically transformed web sites [10].

One of the most interesting web log mining methods is *web users clustering* [11]. The problem of web users clustering (or segmentation) is to use web access log files to partition a set of users into clusters such that the users within a cluster are more similar to each other than users from different clusters. The discovered clusters can then help in on-the-fly transformation of the web site content. In particular, web pages can be automatically linked by artificial hyperlinks. The idea is to try to match an active user's access pattern with one or more of the clusters discovered from the web log files. Pages in the matched clusters that have not been explored by the user may serve as navigational hints for the user to follow.

Consider a web access log file, where each URL request is stored together with a timestamp and a user IP address. An example of such dataset is given in Figure 1a with its transformed form given in Figure 1b, where an ordered list of URL requests is stored for each web user. Assume that the problem is to cluster the users' access sequences into two clusters, containing the users having similar access histories. The basic question here is: how to measure the similarity of two user's access sequences? It seems that e.g. the users 150.254.32.101 and 150.254.32.105 are similar since they both contain the same subsequence '$url_7 \rightarrow url_8$' . But what can we say about the similarity of the user's sequences e.g. 150.254.32.102 and 150.254.32.103? We argue that these two user's sequences also can be considered similar, since there is a user's sequence (150.254.32.104) which contains both the subsequence that is common to 102 (‘$url_2 \rightarrow url_9$’) and the subsequence that is common to 150.254.32.103 (‘$url_6 \rightarrow$

$url_{11} \rightarrow url_{14}$’). In general, we assume that two sequences are similar if either they contain the identical subsequences, or there exists a *connecting path* (Figure 1c) through a set of other sequences. In our example, the best solution would be to put the users 150.254.32.101 and 150.254.32.105 into one cluster and the users 150.254.32.102, 150.254.32.103, 150.254.32.104 into the other. Notice that the similarity between two user access sequences always depends on the presence of other sequences contributing to connecting paths. Because of that, we do not formalize the similarity measure between two sequences. Instead, we directly introduce a criterion function that represents the quality of clustering. A cluster in our approach is a set of user access sequences such that within the set several subsequences occur much more frequently than outside the set. Quality of clustering depends on how frequently subsequences which are typical for a given cluster occur in other clusters. We do not consider all possible subsequences that can be extracted from web access log files, because their number is likely to be very large. We concentrate on subsequences that frequently occur in the files, called sequential patterns [2]. There exist a number of fast algorithms for discovering sequential patterns and we can use any of them as the preprocessing step.

We propose a heuristic algorithm for discovering an arbitrary number of possibly overlapping clusters that hold the web users, whose behavior is similar to each other. We refer to our clustering method as to *partial* clustering, because we allow the users who are not similar to any other not to be covered by any cluster, and we allow a web user to belong to more than one cluster.

time	**user_IP**	**item**
03:18	150.254.32.101	url_1
03:18	150.254.32.104	url_2
03:19	150.254.32.102	url_3
03:21	150.254.32.101	url_4
03:21	150.254.32.102	url_2
03:27	150.254.32.101	url_5
04:01	150.254.32.103	url_6
04:03	150.254.32.104	url_6
04:06	150.254.32.105	url_7
04:06	150.254.32.101	url_7
04:11	150.254.32.101	url_8
04:15	150.254.32.105	url_8
04:16	150.254.32.102	url_9
04:17	150.254.32.102	url_{10}
04:17	150.254.32.103	url_{11}
04:17	150.254.32.104	url_9
04:20	150.254.32.105	url_{12}
04:22	150.254.32.103	url_{13}
04:24	150.254.32.104	url_{11}
04:25	150.254.32.103	url_{14}
04:26	150.254.32.104	url_{14}
04:28	150.254.32.105	url_{15}
04:29	150.254.32.105	url_5
04:30	150.254.32.105	url_{16}

Fig. 1a. Example dataset

user_IP	**sequence**
150.254.32.101	$url_1 \rightarrow url_4 \rightarrow url_5 \rightarrow url_7 \rightarrow url_8$
150.254.32.102	$url_3 \rightarrow url_2 \rightarrow url_9 \rightarrow url_{10}$
150.254.32.103	$url_6 \rightarrow url_{11} \rightarrow url_{13} \rightarrow url_{14}$
150.254.32.104	$url_2 \rightarrow url_6 \rightarrow url_9 \rightarrow url_{11} \rightarrow url_{14}$
150.254.32.105	$url_7 \rightarrow url_8 \rightarrow url_{12} \rightarrow url_{15} \rightarrow url_5 \rightarrow url_{16}$

Fig. 1b. Transformed form of the example dataset

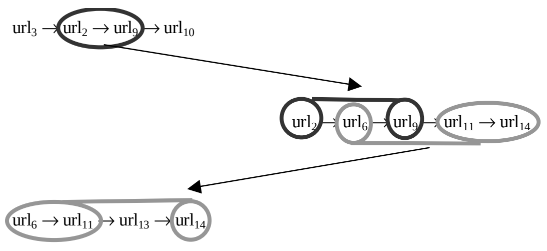
$url_3 \rightarrow url_2 \rightarrow url_9 \rightarrow url_{10}$
$url_2 \rightarrow url_6 \rightarrow url_9 \rightarrow url_{11} \rightarrow url_{14}$
$url_6 \rightarrow url_{11} \rightarrow url_{13} \rightarrow url_{14}$

Fig. 1c. The idea of *connecting path*.

Our clustering algorithm is agglomerative in nature. It starts with a number of small clusters and merges them together to reach the given number of resulting clusters. In the initial set of clusters, each cluster corresponds to one frequent pattern and contains all sequences supporting it. Clusters are iteratively merged according to the set similarity measure called Jaccard coefficient [8] applied to their contents.

2 Related Work

Many clustering algorithms have been proposed in the area of machine learning [6] [8] and statistics [9]. Those traditional algorithms group the data based on some measure of similarity or distance between data points. They are suitable for clustering data sets that can be easily transformed into sets of points in n-dimensional space, which makes them inappropriate for categorical data.

Recently, several clustering algorithms for categorical data have been proposed. In [7] a method for hypergraph-based clustering of transaction data in a high dimensional space has been presented. The method used frequent itemsets to cluster items. Discovered clusters of items were then used to cluster customer transactions. [5] described a novel approach for clustering collections of sets, and its application to the analysis and mining of categorical data. The proposed algorithm facilitated a type of similarity measure arising from the co-occurrence of values in the data set. In [4] an algorithm named CACTUS was presented together with the definition of a cluster for categorical data. In contrast with the previous approaches to clustering categorical data, CACTUS gives formal descriptions of discovered clusters. Unfortunately, none of the algorithms employs sequential dependencies to generate clusters.

The most similar approach to ours is probably the approach to document clustering proposed in [3]. The most significant difference between their similarity measure and ours is that we look for the occurrence of variable-length subsequences and concentrate only on frequent ones.

Most of the research on sequences of events concentrated on the discovery of frequently occurring patterns. The problem was introduced in [2]. The class of patterns considered there, called sequential patterns, had a form of sequences of sets of items. The statistical significance of a pattern (called support) was measured as a percentage of data sequences containing the pattern.

3 Problem Formulation

Definition 3.1. Let $L = \{l_1, l_2, ..., l_m\}$ be a set of literals called items. Each item represents a single document. A *sequence* $S = <X_1\ X_2\ ...\ X_n>$ is an ordered list of items such that each item $X_i \in L$. Let the database D be a set of sequences.

Definition 3.2. We say that the sequence $S_1 = <Y_1\ Y_2\ ...\ Y_m>$ *supports* the sequence $S_2 = <X_1\ X_2\ ...\ X_n>$ if there exist integers $i_1 < i_2 < ... < i_n$ such that $X_1 = Y_{i1}$, $X_2 = Y_{i2}$, ..., $X_n = Y_{in}$. We also say that the sequence S_2 is a *subsequence* of the sequence S_1 (denoted by $S_2 \subset S_1$).

Definition 3.3. A *frequent pattern* is a sequence that is supported by more than a user-defined minimum number of sequences in *D*. Let *P* be a set of all frequent patterns in *D*.

Definition 3.4. A *cluster* c is an ordered pair <*Q*,*S*>, where $Q \subseteq P$ and $S \subseteq D$, and *S* is a set of all database sequences supporting at least one pattern from *Q*. We call *Q* a *cluster description*, and *S* a *cluster content*. We use a dot notation to refer to a cluster description as to *c.Q* and to a cluster content as to *c.S*.

In order to measure, how much a sequence matches a cluster, we introduce the *error function*. The function returns *0* when a sequence matches the cluster perfectly.

Definition 3.5. Given the set of clusters *C*, the *error function err*(*c,s*) for a cluster *c* and a sequence *s* is defined as follows:

$$err(c,s) = \left|\{p \in c.Q : s \text{ does not support } p\}\right| + \left|\{p \in x.Q : x \in C \wedge x \neq c \wedge s \text{ supports } x\}\right|$$

In other words, the error of placing a sequence *s* in a cluster *c* is equal to the number of patterns, which define *c* but are *not* supported by *s*, plus the number of patterns, which define the other clusters but *are* supported by *s*. The perfect match of a sequence *s* and a cluster *c* occurs when *s* supports all patterns from *c* and none of the patterns from any other cluster.

Problem statement. Given a database $D = \{s_1, s_2, ..., s_k\}$ of data sequences, and a set $P = \{p_1, p_2, ..., p_m\}$ of frequent patterns in *D*, the problem is to divide *P* into a set of *n* clusters $c_1, c_2, \ldots, c_n$, such that:

$$\forall_{i,j,i \neq j} \; c_i.Q \cap c_j.Q = \emptyset \,,$$

minimizing the following error criterion:

$$E = \sum_{i=1}^{n} \sum_{s \in c_i.S} \left(err(c_i, s)\right)^2 \,.$$

4 Pattern-Oriented Partial Clustering

In this section, we describe a new clustering algorithm POPC for clustering large volumes of web access sequential data. The algorithm implements the general idea of agglomerative hierarchical clustering. However, instead of starting with a set of clusters containing one sequence each, our algorithm uses previously discovered frequent patterns and starts with clusters containing web access sequences supporting the same pattern. We assume that frequent patterns have already been discovered.

Before we present the clustering algorithm itself, we provide a formal definition of a *union* of two clusters, which is the result of merging two clusters together. We also give a formula that serves as *inter-cluster similarity* used to determine the order in which clusters are merged.

Definition 4.1 A *union* c_{ab} of the two clusters c_a and c_b is defined as follows:

$$c_{ab} = union(c_a, c_b) = < c_a.Q \cup c_b.Q \,,\, c_a.S \cup c_b.S >$$

Definition 4.2 *Inter-cluster similarity* between two clusters c_a and c_b is a Jaccard coefficient applied to cluster contents:

$$f(c_a, c_b) = \frac{|c_a.S \cap c_b.S|}{|c_a.S \cup c_b.S|} \ .$$

The above similarity function reflects co-occurrence of patterns describing two clusters (the size of the intersection of the cluster contents represents the number of sequences containing at least one pattern from each of the two cluster descriptions). It returns values from the range of *<0;1>*, where the value of *1* means that the clusters are identical while the value of *0* means that the clusters exhibit no similarity at all. The inter-cluster similarity measure was chosen so that it reduces the number of sequences supporting patterns associated with other clusters.

4.1 Algorithm POPC

The algorithm for partial clustering based on frequently occurring patterns is decomposed into two following phases: *Transformation Phase* and *Merge Phase*.

Transformation Phase. In this phase, the database is transformed into a pattern-oriented form, which is more suitable for evaluating unions and intersections of cluster contents (used in the subsequent phases). For each frequent pattern we keep an ordered list of web access sequences supporting the pattern. Each web access sequence is represented by its identifier, e.g. an IP address of a web client. Sequences that do not support any frequent pattern are ignored.

Each pattern, together with the list of sequences supporting it, constitutes a cluster whose description is a set that contains the pattern as its only element. The cluster's content is made up of a set of web access sequences from the list.

The proposed database representation simplifies evaluation of inter-cluster similarities. There is no need to refer to the original database in subsequent phases of the algorithm. Moreover, the size of the transformed database reduces as clusters are being merged together. When the process is finished, the database contains the result of clustering (descriptions and contents of the discovered clusters).

Merge Phase. Figure 2 presents the Merge Phase of the clustering algorithm. First, the *m* patterns are mapped into *m* clusters, forming an initial set of clusters C_1, where each cluster is described by exactly one pattern. In the next step, the similarity function values are evaluated for all possible pairs of clusters. The similarity values are stored in a form of a matrix M_1. Next, the algorithm iteratively merges together pairs of clusters according to their similarity values and cluster contents' sizes. In each iteration *k*, the two most similar clusters $c_a, c_b \in C_k$ are determined, and replaced by a new cluster c_{ab} = *union*(c_a, c_b). If there are several pairs of clusters having maximal similarity values, then the two clusters having the smallest contents are merged. The actual merging is done by the function called *cluster*, described in detail later in the paper. When the new cluster is created, the matrix containing similarity values has to be re-evaluated. This operation is performed by means of the function called *simeval*, described later in the paper.

```
$C_1 = \{c_i: c_i.Q=\{p_i\}, c_i.S=\{s_j: s_j \in D \wedge s_j$ *supports* $p_i\}\}$;
$M_1 =$ *simeval*$(C_1, \varnothing)$;    $k=1$;
***while*** $|C_k| > n$ and exist $c_a,c_b \in C_k$ such that $f(c_a,c_b) > 0$ ***do begin***
    $C_{k+1} =$ *cluster*$(C_k, M_k)$;    $M_{k+1} =$ *simeval*$(C_{k+1}, M_k)$;    k++;
***end;***
Answer $=C_k$;
```

Fig. 2. Merge Phase

The Merge Phase stops when the number of clusters reaches *n* (the required number of clusters) or when there is no such pair of clusters $c_a,c_b \in C_k$ whose similarity is greater than *0*. The latter condition implies that the algorithm may discover a larger number of clusters than requested by a user. In this case, the number of discovered clusters (as well as the fraction of the original database covered by them) depends on the number and strength of frequent patterns used for clustering. If the quality of clustering is unsatisfactory, the clustering should be repeated with a higher number of frequent patterns (patterns satisfying a lower frequency threshold).

Similarity Matrix Evaluation: simeval. Similarity matrix M_l stores the values of the inter-cluster similarity function for all possible pairs of clusters in an l-th algorithm iteration. The cell $M_l(x,y)$ represents the similarity value for the clusters c_x and c_y from the cluster set C_l (see example in Figure 3). The function simeval computes the values of the similarity matrix M_{l+1}, using both the similarity matrix M_l and the current cluster contents. Notice that in all iterations except the first one, the similarity matrix need not be completely re-computed. Only the similarity values concerning the newly created cluster have to be evaluated. Due to diagonal symmetry of the similarity matrix, for k clusters, only $(k^2-k)/2$ similarity function values need to be computed before the first iteration, and only (k-1) in the subsequent ones.

In each iteration, the size of the matrix decreases since two rows and two columns corresponding to the clusters merged to form a new one are removed and only one column and one row are added for a newly created cluster.

-	$f(c_2, c_1)$	$f(c_3, c_1)$
$f(c_1,c_2)$	-	$f(c_3, c_2)$
$f(c_1,c_3)$	$f(c_2, c_3)$	-

$f(c_1,c_2) = f(c_2, c_1)$
$f(c_1,c_3) = f(c_3, c_1)$
$f(c_2, c_3) = f(c_3, c_2)$

Fig. 3. Structure of the similarity matrix for three clusters

Cluster Merging: cluster. In each iteration, the number of processed clusters decreases by one. The similarity-based merging is done by the function called cluster. The function cluster scans the similarity matrix and finds pairs of clusters, such that their similarity is maximal. If there are many pairs of clusters that reach the maximal similarity values, then the function cluster selects the one with the smallest size of the union of their contents. Notice that no access to the original database is required to perform this phase of the algorithm. The function cluster takes a set of clusters C_k as one of its parameters and returns a set of clusters C_{k+1} such that $C_{k+1} = (C_k \setminus \{c_a, c_b\}) \cup \{c_{ab}\}$, where $c_a,c_b \in C_k$ are clusters chosen for merging and $c_{ab} = \text{union}(c_a,c_b)$.

4.2 Sequence Classification

Having discovered the clusters, we can use the clustering model, represented by cluster descriptions, to classify new web users. For each new web user access path, we look for frequent patterns supported by it. If the access path supports a pattern from a cluster's description, it is assigned to that cluster. If the access path supports patterns from descriptions of more than one cluster it is mapped to all of them. Taking into account the number of patterns in each cluster's description that are supported by a given web user, we could compute membership probability of his access path in each cluster (based on the error function *err*).

5 Experimental results

To assess the performance and results of the clustering algorithm, we performed several experiments on a *Pentium II* client platform, running *MS Windows98,* cooperating with *Oracle 8.0.5* relational database management system on 2-processor Sun SPARCserver 630MP, with 128 MB of main memory, running Solaris. Experimental data sets were created by synthetic data generator *GEN* from Quest project [1]. *GEN* generates textual data files containing sets of numerical items.

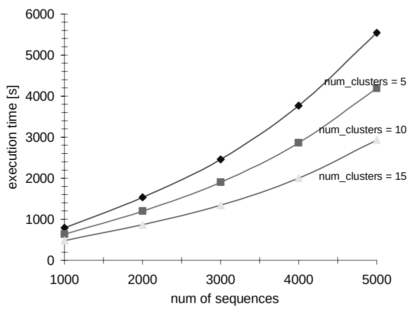

Fig. 4. Execution time for different database sizes

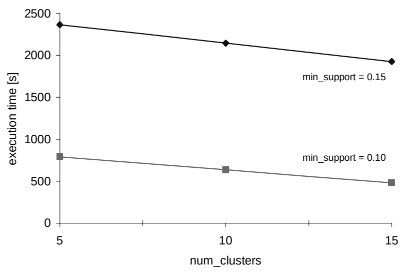

Fig. 5. Execution time for different number of resulting clusters

Figure 4 shows the performance of the clustering algorithm for different database sizes. The execution time grows almost linearly for the increasing number of source sequences. This behavior is caused by the fact that the number of sequences supporting given frequent patterns (for the same support threshold) grows linearly as the size of the database increases.

Figure 5 illustrates the influence of the number of requested clusters on the execution time of our algorithm. We observe that the execution time depends almost linearly on the number of iterations of the algorithm (each iteration merges one pair of clusters). We could expect the execution time of each iteration to decrease, because the number of clusters reduces. However, we should notice that the size of clusters in fact increases, and it compensates the above feature.

We have also evaluated the quality of results produced by our heuristic algorithm. Using our implementation of a combinatory algorithm to find the ideal clustering, we compared the ideal solution with the heuristic one and we computed the quantity of incorrectly clustered sequences. Due to NP-complexity of the combinatory algorithm we were able to perform the test for a small number of initial clusters only (10-15). However, the test showed that our heuristic algorithm correctly clusters c.a. 91% of database sequences, what seems to be a very good result.

6 Conclusions and Future Work

We considered the problem of clustering web access sequences. Due to the limitations of the existing clustering methods, we introduced a new algorithm, which uses frequent patterns to generate both clustering model and cluster contents. The algorithm iteratively merges smaller, similar clusters until the requested number of clusters is reached. In the absence of a well-defined metric space, we propose the inter-cluster similarity measure based on co-occurrence to be used in cluster merging.

An important feature of the algorithm is that it does not only divide the web users into clusters but also delivers a classification model that can be used to classify future web users. Since the model is formed by a set of frequent patterns to be contained, the classification of a new web user access path simply consists in checking if it contains patterns from any of the clusters' descriptions. If the new user access path contains patterns from different clusters, then it belongs to many clusters with different membership probabilities.

References

1. Agrawal, R.; Mehta, M.; Shafer, J.; Srikant, R.; Arning, A.; Bollinger, T.: The Quest Data Mining System. In Proc. of the 2nd KDD Conference (1996)
2. Agrawal R., Srikant R.: Mining Sequential Patterns. Proc. of the 11th ICDE Conf. (1995)
3. Broder A., Glassman S., Manasse M., Zweig G.: Syntactic clustering of the Web. Proceedings of the 6th International WWW (1997)
4. Ganti V., Gehrke J., Ramakrishnan R.: CACTUS-Clustering Categorical Data Using Summaries. Proc. of the 5th KDD Conference (1999)
5. Gibson D., Kleinberg J.M., Raghavan P.: Clustering Categorical Data: An Approach Based on Dynamical Systems. Proc. of the 24th VLDB Conference (1998)
6. Hartigan J.: Clustering Algorithms. John Wiley and Sons (1975)
7. Han E., Karypis G., Kumar V., Mobasher B.: Clustering based on association rules hypergraphs. Proc. Workshop on Research Issues on Data Mining and Knowledge Discovery (1997)
8. Jain A.K., Dubes R.C.: Algorithms for Clustering Data. Prentice Hall (1988)
9. Kaufman L., Rousseeuw P.: Finding Groups in Data. John Wiley and Sons (1989)
10. Perkowitz, M., Etzioni, O.: Adaptive web sites: an AI challenge, Proc. of the 15th Int'l Joint Conf. on Artificial Intelligence (1997)
11. Yan, T.W., Jacobsen, M., Garcia-Molina, H., Dayal, U.: From User Access Patterns to Dynamic Hypertext Linking, proc. of Fifth International WWW Conference (1996)