Alternative Approach to Tree-Structured Web Log Representation and Mining

Fedja Hadzic, Michael Hecker
DEBII, Curtin University
Perth, Australia
{fedja.hadzic,michael.hecker}@curtin.edu.au

Abstract*—More recent approaches to web log data representation aim to capture the user navigational patterns with respect to the overall structure of the web site. One such representation is tree-structured log files which is the focus of this work. Most existing methods for analyzing such data are based on the use of frequent subtree mining techniques to extract frequent user activity and navigational paths. In this paper we evaluate the use of other standard data mining techniques enabled by a recently proposed structure preserving flat data representation for tree-structured data. The initially proposed framework was adjusted to better suit the web log mining task. Experimental evaluation is performed on two real world web log datasets and comparisons are made with an existing state-of-the-art classifier for tree-structured data. The results show the great potential of the method in enabling the application of a wider range of data mining/analysis techniques to tree-structured web log data.

Keywords: web usage mining, tree-structured web logs

I. INTRODUCTION

Web mining is the use of data mining techniques to automatically extract useful information and knowledge patterns from the web documents and services [1]. The problem differs depending on the type of information mined and the knowledge sought after in the application. Hence, within the general problem one can distinguish between *web content mining* – extracting information and analysing the structure of the documents on the web, *web usage mining* (also referred to as web log mining) - mining for user access and browsing patterns, and *web structure mining* – mining the structure of the hyperlinks within the web [2, 3]. The work presented in this paper is focused on the web usage mining problem, and more specifically to mining of web log data represented in tree-structured form.

The type of information commonly sought after during the analysis of web logs is: users' browsing patterns, analysis of users visiting the same web pages, most frequent usage paths, paths occurring frequently inside user interactions with the site, detecting users who visit certain pages frequently, etc. [4]. This information is often extracted using the frequent pattern mining algorithms where the user interactions with a web site are represented as separate records in transactional or sequential databases. Many methods have been developed for this purpose, and frequent itemset and sequence mining algorithms are commonly used to efficiently extract interesting associations, that reveal common user activities and interests [5-7]. Besides these advances in web log mining, a number of works focused on web log data represented in a more complex form so that the navigational pattern and the overall structure of the web site can be described in the data [8-10]. One example is the LOGML language proposed in [11] which allows for a more detailed and informative representation of web logs, using an XML template. The set of requested web pages and traversed hyperlinks in a web log file are represented as a web graph (tree) which is a subset of the web site graph of the web server being analysed. In the context of tree mining, a tree-structured pattern will contain more descriptive information in comparison with an itemset or sequence pattern. This was demonstrated in [12] where it is shown that representing web logs in a tree-structured form allows one to mine for substructures which are more informative, because they often reveal more information regarding the structure of the web site and users' navigational patterns.

In this work we take an alternative approach for web log mining, based on our recently proposed [13] structure preserving flat data representation for tree-structured data such as XML. The conversion process is based on the extraction of a database structure model within which each tree instance can be matched to generate the flat data representation that captures the structural properties. The work demonstrated the effectiveness of the method when the XML schema is well defined and the attributes/values in each instance follow the same order with respect to the schema. Decision tree learning performed very well when applied on a synthetic dataset for classifying instances according to their structural properties. However, when tested on web log data, while it had satisfactory classification accuracy, the task was fairly complex and took long time to execute. This is because the structure of instances of web log data can vary greatly among sessions and having a database structure model that captures all of them can be quite inefficient as well as classification accuracy can suffer due to the many irrelevant nodes present. Hence in this work, the approach of database structure model extraction and flat data format generation is modified to suit better for web log data. The user can supply the minimum frequency threshold for the part of the structural characteristics to be considered as part of the database structure model extracted and hence as part of the training set. In web log data this interprets to only the frequently occurring navigational patterns within user sessions to be taken into account. The not so common navigational patterns within a session are therefore ignored, while the behavior that can be classified as relevant is still captured. The experimental evaluation is performed using two real world web log datasets and the comparisons are made with existing structural classifier. The results indicate the suitability of the method as an alternative approach to web log mining. Furthermore, given that many more data mining/analysis techniques exist for flat data representation, the conversion approach in itself can potentially enable a wider range of techniques to be applied for mining of tree-structured web log data. This will hasten the application process, as any already developed data mining method can be directly applied, rather than individually adapting each of the methods to tree-structured data.

The rest of the paper is organized as follows. In Section 2 we overview some of the related works. The definitions of concepts essential for understanding the proposed method are described in Section 3. Section 4 overviews the basics of the structure preserving flat data representation for tree-structured data and the extensions proposed in this work. Experiments are provided in Section 5. In Section 6 we summarize the paper and indicate areas of future investigation.

II. Related Works

Web usage mining aims to discover information such as: users' browsing patterns, analysis of users visiting the same web pages, most frequent usage paths, paths occurring frequently inside user interactions with the site, detecting users who visit certain pages frequently, etc. [3, 4, 14]. Due to the nature of the web log data, many of the aims can be satisfied using frequent pattern mining approaches, where the pattern can correspond to either an itemset, sequence or a graph structured item [11]. One can then form association rules from the extracted patterns. For example, the WEBMINER system has been developed to discover association patterns and sequential patterns from server access logs [15]. Many approaches have been developed that apply (modifications of) association rule and sequential pattern discovery techniques to analyse web logs [9, 16-18]. Other techniques have also proved useful for web usage analysis, such as statistical analysis, clustering, classification and dependency modelling. For example, WebPersonalizer system [19] provides a framework for mining web log data and provides recommendations to users based on their similarities to previous users. It uses a combination of association rule mining, clustering, sequence mining, and classification. A unifying probabilistic framework for clustering individuals where the data is in non-vector form has been proposed in [20]. A general expectation-maximization procedure for clustering is adopted and the framework has been successfully applied to cluster individuals based on their web navigation behaviour. The discovery of recurrent patterns embedded in sequences of web page requests is also of importance for web usage mining. In [21] the authors proposed the use of the Bayes error rate framework for analysing problems of this nature in a Markov context. In [22], a different approach is adopted that combines OLAP and data mining techniques to analyse web logs for prediction, classification and time-series analysis. As in general data mining, pre-processing of the web usage patterns is an important issue and some approaches have been discussed in [4]. For a more detailed overview of the existing web log mining techniques, please refer to [5-7].

Most of the techniques discussed above mine web logs that are represented in a relational or sequential database, and hence, usually adopt a frequent itemset or sequence mining based approach for discovering interesting associations. We will now examine some approaches that represent web logs using a graph structure in order to extract more informative patterns. These approaches are motivated by the observation that additional information such as navigational patterns and subsessions are harder to realize from the traditional representation of web logs, and as such have proposed alternative ways for representing web log data and mining it. For example, in order to capture navigational patterns, the work presented in [23] models the web log data as a directed weighted graph where the weights indicate the probabilities that reflect the user interaction with the web site. They refer to this representation as hypertext probabilistic grammar and have tailored the association rule framework for analysing such representations. They have extended their work in [8] by making use of entropy as an estimator of statistical properties of the hypertext grammar. The MiDAS algorithm [9] map web log data onto a concept hierarchy and then discover sequences of hits by extracting pattern trees where a node contains all the properties of a hit (URL, frequency, timestamp) and the links represent the relationships between the nodes. A notion of the aggregate tree is proposed in [24], used for pre-processing of a sequential web log file. It eliminates duplicates and merges all sequences with common prefixes together so that each node in the tree is annotated with a number of sequences having the same prefix up to and including this node. This tree is then scanned using an analytical procedure to discover the navigational behaviour of users. In [10], the granularity of user sessions is increased by adding information about sub-sessions within web page access sessions, in order to predict the following set of pages to be visited by a user. The sub-session information is represented using a tree structure and the frequent path within the tree indicates the set of pages likely to be visited from a given node (page) in the tree.

In the abovementioned work, the main motivation for representing web log data as a tree structure is to add granularity to the information that can be extracted from web logs and inferences that can be made, as well as simplify the mining process to some extent. In order to enable easier, more meaningful and standardized representation of web log data which can capture information related to the structure of the web site and navigational patterns (sub-sessions), LOGML language was developed [11]. It is an XML 1.0 application for describing log reports of web servers. The set of requested web pages and traversed hyperlinks in a web log file are represented as a web graph (tree) which is a subset of the web site graph of the web server being analysed. In [12], an algorithm for mining ordered embedded subtrees was presented and it was demonstrated how it can be useful for extracting additional information from web log data represented in tree-structured form, in comparison when itemset mining or sequence mining approach was applied.

III. Problem Background

As the focus of this paper is on tree-structured web log data, this section first defines some tree related concepts and then proceeds with an example tree representation of web server log sessions. To lay the necessary ground for describing the proposed method in Section 4, we conclude the section with a definition of commonly used string encoding for trees.

A graph consists of a set of nodes (or vertices) that are connected by edges. Each edge has two nodes associated with it. A path is defined as a finite sequence of edges. A rooted tree has its top-most node defined as the root that has no incoming edges. In a tree there is a single unique path between any two nodes. A node *u* is said to be a parent of node *v*, if there is a directed edge from *u* to *v*. Node *v* is then said to be a child of node *u*. Nodes with the same parent are called siblings. A *rooted ordered labelled tree* can be denoted as $T = (v_{0,} V,L,E)$, where (1) $v_0 \in V$ is the root vertex; (2) V is the set of vertices or nodes; (3) L is a labelling function that assigns a label $L(v)$ to every vertex $v \in V$; (4) $E = \{(v_1,v_2)|\ v_1,v_2 \in V$ AND $v_1 \neq v_2\}$ is the set of edges in the tree, and (4) for each internal node the children are ordered form left to right. The problem of *frequent subtree mining* can be generally stated as: given a database of trees *Tdb* and minimum support threshold (σ), find all subtrees that occur at least σ times in *Tdb* [25]. A *pre-order traversal* of a tree can be defined as follows: If ordered tree *T* consists only of a root node *r*, then *r* is the pre-order traversal of *T*. Otherwise let T_1, T_2, ..., T_n be the subtrees occurring at *r* from left to right in *T*. The pre-order traversal begins by visiting *r* and then traversing all the remaining subtrees in pre-order starting from T_1 and finishing with $T_{n.}$ For an extensive overview of the frequent subtree mining field the interested reader is referred to [25-27].

To illustrate how a user session can be represented as a tree consider two simple sessions in Fig. 1 that have been logged from our institutes' web server. Each session starts from the home page www.debii.curtin.edu.au, denoted with '/' in Fig. 1. For faster processing the strings are commonly mapped to unique integers, and in Table 1 we present the mapping used in this example (note that '.html' is ignored).

Session 1:
/
/about.html
/about/objectives.html
/about/mission-and-vision.html
/research.html
/research/topics.html
/research/topics/50-data-mining-and-semantic-technologies.html
/research/topics/51-business-intelligence.html
/phd-a-msc.html

Session 2:
/
/research.html
/research/topics.html
/research/publications.html
/research/projects.html
/research/seminars.html
/contact-us.html

Figure 1. Example user sessions

TABLE I. INTEGER MAPPING FOR WEB PAGES FROM FIG. 1

ID	Web Page
0	Home page
1	about
2	objectives
3	mission-and-vision
4	research
5	Topics
6	50-data-mining-and-semantic-technologies
7	51-business-intelligence
8	phd-a-msc
9	publications
10	projects
11	seminars
12	contact-us

The access sequence of web pages from Fig. 1 can be represented in a tree-structured way as shown in Fig. 2. The order of pages accessed is reflected by the pre-order traversal of the tree. The corresponding tree structure is more informative than just a sequence of pages accessed as it captures the structure of the web site, and navigational patterns over this website. By organizing them in such a way, specific pages can be considered under the same context. An example of this is the two pages being grouped under the 'topics' parent node with label 5 in the tree of session 1, and four pages under the 'research' parent node with label 6 in the tree of session 2. The session 1 has come from an IP within the university and is an example of most likely a student gaining some general information about the institute and then looking at the postgraduate study related information. The second session came from an IP external to the university (most likely from another university within Perth), where the user was interested in some research topic and after finding out a public seminar being held, contacted the institute for more information.

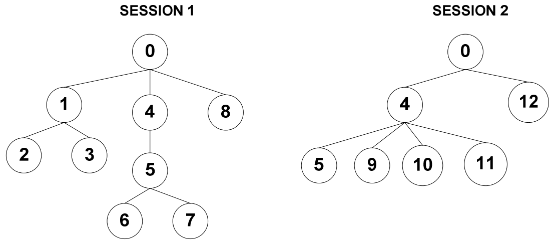

Figure 2. Tree representation of user sessions (Fig. 1)

A common way of representing trees is using the pre-order (depth-first) string encoding (φ) as described in [12].

The pre-order string encoding [12] of a tree *T*, denoted as $\varphi(T)$, can be generated by adding vertex labels in a pre-order traversal of a tree $T = (v_0, V, L, E)$, and appending a backtrack symbol (for example '-1', '-1'$\notin L$) whenever we backtrack from a child node to its parent node. For example, the pre-order string encoding of the two trees from Fig. 2 are, φ(session 1)='0 1 2 -1 3 -1 -1 4 5 6 -1 7 -1 -1 -1 8 -1' *and* φ(session 2)='0 4 5 -1 9 -1 10 -1 11 -1 -1 12 -1'.

IV. Method Description

This section starts first with description of the flat representation of tree-structured data (henceforth referred simply as table) initially proposed in [13]. We then discuss its main characteristics and some of the limitations when applied to web log data. This leads to the adjustment of the method described at the end of the section.

A. Basis of the approach

The first row of a (relational) table consists of attribute names, which in a tree database are scattered through independent tree instances (transactions). One way to overcome this problem is to first assume a structure according to which all the instances/transactions are organized. Each of the transactions in a tree database should be a valid subtree of this assumed structure, which we refer to as the database structure model (*DSM*). This *DSM* will become the first row of the table, and while it does not contain the attribute names, it contains the most general structure where every instance from the tree database can be matched to. The *DSM* needs to ensure that when the labels of a particular instance from the tree database are processed, they are placed in the correct column, corresponding to the position in the *DSM* where this label was matched to. Hence, the labels (attribute names) of this *DSM* will correspond to pre-order positions of the nodes of the *DSM* and sequential position of the backtrack ('-1') symbols from its string encoding.

The process of extracting a *DSM* from a tree database consists of traversing the tree database and expanding the current *DSM* as necessary so that every tree instance can be matched against *DSM*. Let the tree database consisting of *n* transactions be denoted as $Tdb = \{tid_0, tid_1, \ldots, tid_{n-1}\}$, and let the string encoding of the tree instance at transaction tid_i be denoted as $\varphi(tid_i)$. Further, let $|\varphi(tid_i)|$ denote the number of elements in $\varphi(tid_i)$, and $\varphi(tid_i)_k$ ($k = \{0, 1, \ldots, |\varphi(tid_i)|-1\}$) denote the k_{th} element (a label or a backtrack '-1') of $\varphi(tid_i)$. The same notation for the string encoding of the (current) *DSM* is used, i.e. $\varphi(DSM)$. However, rather than storing the actual labels in $\varphi(DSM)$, 'x' is always stored to represent a node in general. The process of extracting the *DSM* from *Tdb* can be explained by the pseudo code in Fig. 3 [13].

To illustrate the complete conversion process using *DSM* please refer back to Fig. 2. We use the string encoding to represent the *DSM* and since the order of the nodes (and backtracks ('-1')) is important we label the nodes and backtracks sequentially according to their occurrence in the string encoding. For nodes (labels in the string encoding), x_i is used as the attribute name, where *i* corresponds to the pre-order position of the node in the tree, while for backtracks, b_j is used as the attribute name, where *j* corresponds to the backtrack number in the string encoding. Hence, from our example in Fig. 2, the string encoding of extracted *DSM* is $\varphi(DSM)$ = 'x_0 x_1 x_2 b_0 x_3 b_1 x_4 b_2 x_5 b_3 b_4 x_6 x_7 x_8 b_5 x_9 b_6 b_7 b_8 x_{10} b_9'. To fill in the remaining rows (instances) every string encoding of each tree from Fig. 2 is scanned and when a label is encountered it is placed to the matching column (i.e. under the matching node (x_i) in *DSM*), and when a backtrack ('-1') is encountered, a value 'yes' is placed to the matching column (i.e. matching backtrack (b_j) in *DSM*). This is easily done by tracking the level of the instance tree and the *DSM* as they are being matched. The remaining entries are assigned values of 'no' (indicating non existence). The resulting table is shown in Table 2 (*DSM*: row 1, session 1: row 2, session 2: row 3).

```
Database Structure Model (DSM) Extraction
Input: tree database Tdb
Output: DSM
inputNodeLevel = 0 // current level of $\varphi(tid_i)_k$
DSMNodeLevel = 0 // current level of $\varphi(T(h_{max}, d_{max}))_k$
$\varphi(DSM) = \varphi(tid_0)$ // set default DSM  (use 'x' instead of labels)
for i = 1 to n – 1  // n = |Tdb|
  for each $\varphi(tid_i)_k$ in $\varphi(tid_i)$
    for each p = 0 to (|$\varphi(DSM)$|-1)
      if  $\varphi(tid_i)_k$ = -1 then inputNodeLevel– –
                      else inputNodeLevel++
      if $\varphi(DSM)_p$= 'b_i' then DSMNodeLevel– –
                      else DSMNodeLevel++
      if inputNodeLevel ≠ DSMNodeLevel
        if  $\varphi(tid_i)_k$ = -1 then
          while inputNodeLevel ≠ DSMNodeLevel
            p++
            if  $\varphi(DSM)_p$ = -1 then DSMNodeLevel– –
                                  else DSMNodeLevel++
          endwhile
         else
       while inputNodeLevel ≠ DSMNodeLevel
             append 'x' at position p+1 in $\varphi(DSM)$
   k++
   p++
             if  $\varphi(tid_i)_k$ = -1 then inputNodeLevel– –
                                   else inputNodeLevel++
           endwhile
    endfor
  endfor
endfor
return DSM
```

Figure 3. Pseudo code for *DSM* extraction

TABLE II. Flat representation of trees in Fig. 2

x_0	x_1	x_2	b_0	x_3	b_1	x_4	b_2	x_5	b_3	b_4	x_6	x_7	x_8	b_5	x_9	b_6	b_7	b_8	x_{10}	b_9
0	1	2	y	3	y	n	n	n	n	y	4	5	6	y	7	y	y	y	8	y
0	4	5	y	9	y	10	y	11	y	y	12	n	n	n	n	n	n	y	n	n

One can see that the database structure model *DSM* determines what a valid instance of a particular tree characteristic is, and takes the exact positions of tree instance within the *DSM* into account. This implies that if two subtrees have the same labels and are structurally the same, but their nodes have been matched against different nodes in *DSM* (i.e. they occur at different positions) they would be considered as instances with different characteristics. For example, in Fig. 2, the subtree with encoding '4 5' would be considered to occur in both sessions as defined within the current tree mining framework. However, within the flat representation they have been matched to different nodes within the *DSM*. This key difference is caused by the fact that not all the instances of a tree database may follow the same order or will have all the elements of the document input structure (e.g. XML schema) available. However, if all the instances in a tree database always follow the same structure and node layout as the input document structure (hence *DSM*), then there would be no such difference. This was experimentally demonstrated in [13]. Additionally, an example scenario was discussed when this distinguishing characteristic may be desired, as the different occurrence may indicate different context. Another motivation of the method is that it can overcome some complexity issues caused by existence of the nodes in every record, which are mainly there to contextualize the information [13]. Because the method enables the use of other data mining methods that do not rely on generation of frequent subtrees discriminative class characteristics are easier detected.

In [13] the method was successfully applied for a structural classification problem, and the extraction of the *DSM* as just explained was necessary in order to capture all of the differing characteristics of a class. However, when tested on web log data, while satisfactory classification accuracy was achieved, it was evident that web log data posed a challenging classification problem for the proposed method. As there are many different ways of navigating a website and the number of web page visits within a session can vary greatly, extracting *DSM* within which each user session can be undesired. It becomes much larger than necessary to capture the majority of trees in the database and running time and classification accuracy are negatively affected.

B. Adapting the approach to web log data

In this work, the approach is modified to be better applicable for web log data. Essentially the user can supply the minimum frequency threshold for the part of the structural characteristics to be considered as part of the *DSM* extracted and hence as part of the training set. In web log mining, this interprets to taking only the frequent enough navigational patterns into account, while ignoring the not so common navigational patterns from a user session.

During the *DSM* extraction phase, we store the number of times that a tree instance has matched a node or backtrack in the progressively built *DSM.* Occurrence of each node attribute within the *DSM* implies the existence of a specific backtrack attribute to ensure structural validity of the pre-order string encoding (e.g. x_2 implies b_0 and x_7 implies b_7 from Table 2). Hence, after the whole *DSM* is extracted, it is safe to simply remove any nodes and backtracks that do not satisfy the minimum frequency set by the user. This will result in *DSM* reflecting the structure that was exhibited by as many instances as the user specified minimum frequency threshold. Another modification needs to occur when building the flat data representation by matching every tree instance to *DSM*, since we still want parts of the sessions that exhibit frequent navigational patterns to be part of the dataset to be mined. Essentially, each instance is matched against the *DSM*, and the current levels of the tree instances and *DSM* is tracked. Whenever the levels match, the label or 'y' is stored in the corresponding column of the table, and when levels do not match, either the *DSM* or the tree instance encoding is traversed until the levels match. Any non matched entries in the table are assigned the 'n' value indicating non existence in the currently processed tree instance.

The conversion process can be formulized as follows. Let the tree database consisting of *n* instances be denoted as *Tdb* = $\{tid_0, tid_1, \ldots, tid_{n-1}\}$, and let the string encoding of the tree instance at transaction tid_i be denoted as $\varphi(tid_i)$. The *DSM* is extracted from *Tdb* using the procedure explained earlier. Further, let $|\varphi(tid_i)|$ denote the number of elements in $\varphi(tid_i)$, and $\varphi(tid_i)_k$ ($k = \{0, 1, \ldots, |\varphi(tid_i)|\text{-}1\}$) denote the k_{th} element (a label or a backtrack '-1') of $\varphi(tid_i)$. The flat data format or table F_T (*C*, *R*) (*C* = columns, *R* = rows) is set up where $C = \{c_0, c_1, \ldots, c_{m-1}\}$ ($m = |C| = |\varphi(DSM)|$), and $R = \{r_0, r_1, \ldots, r_{p-1}\}$ ($p = |R| = n+1$ (i.e. extra column for attribute names). The value in column number *x* and row number *y* is denoted as $F_T\ (c_x,\ r_y)$. Hence, to set the attribute names $F_T\ (c_i,\ r_0) = \varphi(DSM)_k$ where $i = k = \{0, 1, \ldots, (|\varphi(DSM)|\text{-}1)\}$. The process of populating the entries from F_T using *Tdb* is explained by the pseudo code in Fig. 4.

```
Tree database Tdb to flat data format $F_T$ conversion
Input: Tdb, DSM
Output: $F_T$
// set up the attribute name row in $F_T$
$F_T (c_i,r_0)=\varphi(DSM)_k \;\forall\; i=k=\{0, \ldots, (|\varphi(DSM)|-1)\}$
inputNodeLevel = 0 // current level of $\varphi(tid_i)_k$
DSMNodeLevel = 0 // current level of $\varphi(DSM)_k$
// populate $F_T$
for $i = 0$ to $n - 1$  // $n = |Tdb|$
  for each $\varphi(tid_i)_k$ in $\varphi(tid_i)$
    for $p = 0$ to $(|\varphi(DSM)|-1)$
      if $\varphi(tid_i)_k = -1$ then inputNodeLevel– –
                               else inputNodeLevel++
      if $\varphi(DSM)_p$= 'b' then DSMNodeLevel– –
                               else DSMNodeLevel++
      if inputNodeLevel = DSMNodeLevel
        if $\varphi(tid_i)_k = -1$ then $F_T (c_p,r_{i+1})$ = 'y'
                               else $F_T (c_p,r_{i+1}) = \varphi(tid_i)_k$
      else // level mismatch
        if $\varphi(tid_i)_k = -1$ then // traverse $\varphi(DSM)$ until match
          while inputNodeLevel ≠ DSMNodeLevel
            $F_T (c_p,r_{i+1})$ = 'n'
            p++
            if $\varphi(DSM)_p = -1$ then DSMNodeLevel– –
                                   else DSMNodeLevel++
          endwhile
```

```
                $F_T(c_p,r_{i+1}) = \varphi(tid_i)_k$
              else // traverse $\varphi(tid_i)$ until match
            while $inputNodeLevel \neq DSMNodeLevel$
              $k$++
              if $\varphi(tid_i)_k = -1$ then $inputNodeLevel$– –
                                  else $inputNodeLevel$++
              endwhile
              $F_T(c_p,r_{i+1}) = \varphi(tid_i)_k$
      endfor
    endfor
  endfor
  return $F_T$
```

Figure 4. Pseudo code for tree database to flat format conversion

V. Experimental evaluation

The purpose of the experiments performed, is to demonstrate that by using the proposed tree database transformation, one can still discover useful knowledge from tree-structured web log data using techniques developed for flat data format. As case in point we evaluate the C4.5 decision tree algorithm [28] and Naive Bayes classifier using Weka software [29]. The results are compared with the XRules structural classifier [30], which discovers association rules based on the algorithm for discovering frequent ordered embedded subtrees [12].

Experiment 1. The first experiment is performed using the web access trees from the computer science department of the Rensselaer Polytechnic Institute previously used in [30] for evaluating the XRules structural classifier. In this dataset the tree instances are labeled according to two classes, namely the internal and external web site access. All of the three datasets (US1924, US2430, US304) were combined and instances were replicated to make the class distribution even. The resulting dataset had 68302 instances out of which 66% was used for training and the remainder for testing. For the XRules approach we have used the default confidence parameter of 0.5 (50%), while the support threshold was varied from 0.3 to 0.01. Since different support thresholds were used, in our approach the flat data representation of the dataset is done separately for each support threshold, as the extracted database structure model (*DSM*) varies, and hence the number of attributes used during decision tree learning. Since XRules approach will not generate any rules that do not satisfy the minimum support threshold, this was reflected in the constraint of the C4.5 for minimum instances per leaf in the decision tree. For the Bayesian classifier we could not impose such a constraint, and hence the default classifier from Weka is used on the flat data generated by frequency adjusted *DSM*. Being a completely different approach, we will exclude the Bayesian classifier from discussion, but will show its results as another example of a standard classifier enabled for application to tree-structured data.

The comparison of predictive accuracy (%), for different support values is shown in Fig. 5. Comparing C4.5 and XRules, one can see that using the C4.5 algorithm achieves better accuracy in all the cases. For thresholds of 0.3 and 0.2 the rules generated within the XRules approach do not satisfy the minimum support and confidence threshold for this dataset, and a default rule was used. Similarly, the C4.5 decision tree only consisted of 2 rules namely b_5 = ‘y’ (class 1) and b_5 = ‘n’ (class 0), for support = 0.3. These rules only correspond to the existence of a particular node in the structure and not the actual value that the node has, as any specific value did not occur frequently enough to satisfy the minimum support threshold. For support value of 0.3, the extracted *DSM* was ‘x_0 x_1 x_2 x_3 x_4 b_0 b_1 x_5 b_2 b_3 x_6 b_4 x_7 b_5 b_6’, and hence through the rule(s) the existence of node x_7 was checked indirectly by checking b_5 and used for classification. For the support value of 0.2 one can see from Fig. 5 that the same classification accuracy was detected. However, the C4.5 decision tree has generated 2428 rules based on the value of the first node accessed in a session. This is because the at such lower support threshold the specific rules were used for classification. However, it is clear that the simple two rules detected at support of 0.3 for checking the existence or non-existence of a node is preferred, which was only captured by the storage of backtrack (b_i) attributes within the *DSM*. This indicates a useful property of the approach that needs to be further explored.

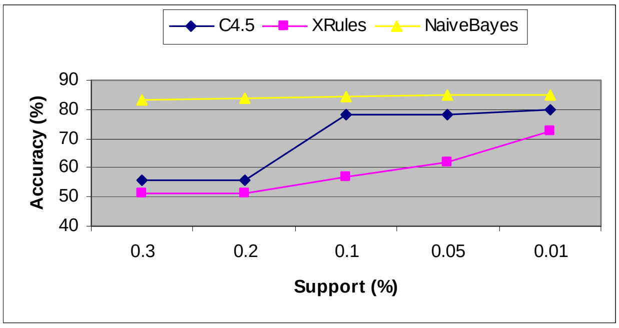

Figure 5. Accuracy on combined CSLogs data

As for the remainder of support values, the frequent subtree miner [12] within the XRules classifier typically created more rules initially in comparison to the number of leaf nodes in C4.5 decision tree, but after applying the confidence threshold these reduced significantly below the number of leaf nodes in C4.5. Please note that the XRules approach, consists of several phases namely the XMiner engine based on [12] which discovers the rules to be used for classification which then need to be evaluated by finding the default class in training and accuracy in testing, as discussed in [30]. In Fig. 6 we show the time taken by the C4.5 algorithm and the XMiner engine of XRules. In addition, the confidence measure used in XRules greatly reduces the number of rules that need to be generated, and the methods are based from different classes of data mining techniques. As such, these are not completely compatible, but the time performance of C4.5 can be considered satisfactory for reasonable support thresholds. At 0.01 support, the time taken is much larger because a leaf node can capture many instances and the *DSM* extracted according to the frequency is much larger (221 attributes as opposed to 63 at support

0.05). Hence many more tests need to be performed for suitable split-nodes in the decision tree.

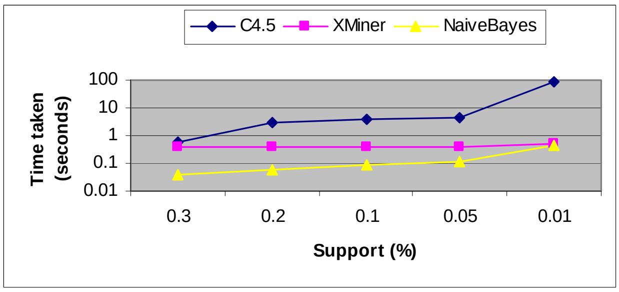

Figure 6. Time performance on combined CSLogs data

Experiment 2. For the second experiment we have taken apache2 (v2.2.3) web server log files from our own website (debii.curtin.edu.au) for a 4 month period in its native (default) format. All "access" log files, which contain attempts to access pages on this website were taken, while messages stored in the normal error message log were excluded. We have removed all requests/lines that do not request a page via a GET or POST request (e.g. HEAD or other requests will be ignored). Furthermore, we have excluded all automatic inclusion requests that appear each and every time a particular page is opened. For example, a page may include a number of images, scripts, stylesheets (css) and others. In the log file they would appear as single separate requests and hence would interfere and could be seen as requests (clicks) that have been done manually. Hence, we only retain all requests where someone has entered our website and has navigated from one page to another by clicking on forms and hyperlinks. Similar to the CSlogs dataset [12, 30] we have classified requests to our website in "internal" (within the university) and "external" (outside the university). As the amount of requests in these two classes is different, we have taken the larger proportion (which is external) and duplicated entries from "internal" to match the "external" ones in numbers. We have defined a session as visiting a webpage from the same computer with no more than 30 minutes in between page visits. We have also ignored sessions where less than three web pages are navigated to, as there is really no navigational pattern inherent in such session. In order to avoid storing automated requests, we have also excluded any request or sequence of requests that are performed by bots (e.g. search bots from google, yahoo or bing) by using the robots.txt as our anchor. Every time a sequence of requests is commenced with a request to access robots.txt, this is excluded as usual (bots request this file only to determine their permissions to index or crawl the website). Furthermore, we excluded any request that is well known to be an attack to our content management system (CMS), usually results in HTTP GET requests with 403 (forbidden) replies (as we have mod_security installed, they can be easily detected).

The grouped user sessions were converted to trees as was explained with the illustrative example in Section 3. The resulting dataset had 18840 instances out of which 66% was used for training and the remainder for testing. The same parameter setting as for the first experiment was applied to both approaches. The comparison of predictive accuracy (%), for different support values is shown in Fig. 7. As one can see the XRules approach has clearly outperformed the C4.5 decision tree for this dataset for all support thresholds. The performance of XRules is particularly better when higher support thresholds are used, and as they are lowered we see that the accuracy of the C4.5 decision tree is approaching that of XRules. The time performance is shown in Fig. 8.

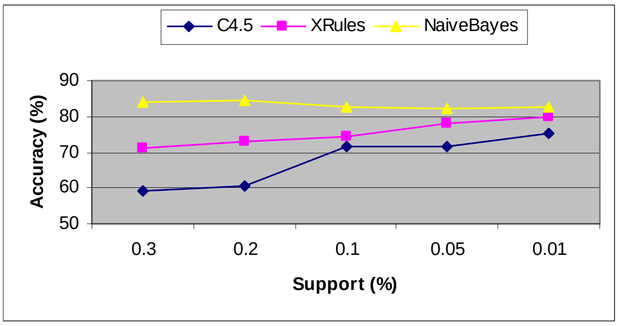

Figure 7. Accuracy on DEBII Logs data

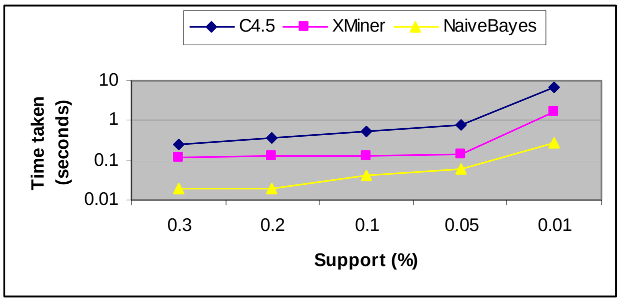

Figure 8. Time performance on DEBII Logs data

The accuracy results comparison between C4.5 and XRules are quite contrasting to the previous results on CSLogs data, and one needs to investigate the differences between the dataset to determine the inconsistency in results. It could be due the fact that the data from the second experiment was collected over a longer period of time, where there is a greater variation among navigational pattern due to potential modifications of the web site. For example, as more web pages are added/removed the particular access to web page may not occur in the same position in the navigational pattern. The general navigational patterns are captured by the database structure model, for the given frequency, and this could also explain why the frequency has to be reduced to such a great deal in order to generalize over those variations of navigational patterns during a larger time period. On the other hand, it could be due to the fact that different data mining methods are used, and perhaps better accuracy could be achieved when different classifiers are used. This was clearly the case for NaiveBayes classifier as in all experiments it had better run time and accuracy, and investigation of other classifiers is left as future work.

VI. CONCLUSIONS AND FUTURE WORK

The work presented in this paper has investigated an alternative approach to tree-structured web log representation and mining. It first converts the tree-structured data into a flat representation that preserves the structural and attribute-value information. This enables an application of a wider range of data mining/analysis techniques. The decision tree learning was taken as a case in point and the approach performed better than an existing structural classifier in one case, and worse in another. Some interesting properties were revealed as some rules were based on the existence of a node within a tree structure reflecting the general navigational pattern. This needs to be investigated further, together with the cases when the differing characteristics of the approach are desired. Furthermore, as many other techniques exist for flat data format, future work will also investigate their use on the proposed representation, with stronger focus on clustering techniques.

REFERENCES

[1] O. Etzioni, "The world wide Web: Quagmire or gold mine," Communications of the ACM, 39(11), 1996, pp. 65-68.

[2] R. Cooley, B. Mobasher, B. and J. Srivastava, "Web mining: Information and pattern discovery on the World Wide Web," Proc. Int'l Conf. on Tools with Artificial Intelligence, Newport Beach, CA, 1997, pp. 558–567.

[3] R. Kosala and H. Blockeel, "Web Mining Research: A Survey," SIGKDD Explorations, vol. 2(1), 2000, pp. 1-15.

[4] R. Cooley, B. Mobasher, and J. Srivastava, "Data preparation for mining World Wide Web browsing patterns," Journal of Knowledge and Information Systems, 1, 1, 1999.

[5] M. Eirinaki and M. Vazirgiannis, "Web Mining for Web Personalization", ACM Transactions on Internet Technology, vol. 3, no. 1, February, 2003.

[6] F.M. Facca and P.L. Lanzi, Mining interesting knowledge from Weblogs: a survey," Data and Knowledge Engineering, 2004.

[7] D. Pierrakos, G. Paliouras, C. Papatheodorou, and D. Spyropoulos, "Web Usage Mining as a Tool for Personalization: A Survey," User Modeling and User-Adapated Interaction, Springer Netherlands, vol. 13, no. 4, November, 2003, pp. 311-372.

[8] J. Borges and M. Levene, "Data mining of user navigation patterns," Proc. of W'shop on Web Usage Analysis and User Profiling, in conjunction with ACM SIGKDD Int'l Conference on Knowledge Discovery and Data Mining, San Diego, CA., 1999 pp. 31-36.

[9] A.G. Buchner, M. Baumgarten, S. Anand, M.D. Mulvenna, and J.G. Andhughes, "Navigation pattern discovery from Internet data," Proc. of the Web Usage Analysis and User Profiling Workshop*, in conj. with 5th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining* (San Diego, August), 1999, pp. 25–30.

[10] E. Menasalvas, S. Millan, J.M. Pena, M. Hadjimichael, and O. Marban, "Subsessions: a granular approach to click path analysis," Proc. IEEE Int'l Conference on Fuzzy Systems, 2002, pp. 878–883.

[11] J. Punin, M. Krishnamoorthy, and M.J. Zaki, "LOGML: Log markup language for Web usage mining," Proc. ACM SIGKDD Workshop on Mining Log Data Across All Customer Touch Points, August, San Francisco, CA, 2001.

[12] M.J. Zaki, "Efficiently mining frequent trees in a forest: algorithms and applications," IEEE Transaction on Knowledge and Data Engineering, vol. 17, no. 8, 2005, pp.1021-1035.

[13] F. Hadzic, "A Structure Preserving Flat Data Format Representation for Tree-Structured Data," Proc. 2nd Workshop on Quality issues, measure of interestingness, and evaluation of data mining models, held in conj. with PAKDD, Shenzhen, China, May 24-27, 2011.

[14] J. Srivastava, P. Desikan, and V. Kumar, "Web mining – concepts, applications and research directions," Data Mining: Next Generation Challenges and Future Directions, AAAI/MIT Press, Boston, MA, 2003.

[15] B. Mobasher, N. Jain, E. Han, and J. Srivastava, "Web mining: Pattern discovery from world wide Web transactions", Technical Report TR-96-050, University of Minnesota, Dept. of Computer Science, Minneapolis, 1996.

[16] F. Masseglia, P. Poncelet, and M. Teisseire, "Using data mining techniques on Web access logs to dynamically improve hypertext structure," ACM SIGWEB Newsletter, vol. 8(3), 1999, pp. 13–19.

[17] J. Pei, J. Han, B. Mortazavi-as, and H. Zhu, "Mining access patterns efficiently from Web logs," Proc. of the 4th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (PAKDD'00), Kyoto, Japan, 2000, pp. 396-407.

[18] S.E. Jespersen, J. Thorhauge, and T.B. Pedersen, "A hybrid approach to Web usage mining," Proc. of the 4th Int'l Conf. on Data Warehousing and Knowledge Discovery, 2002, pp. 73-82.

[19] B. Mobasher, R., Cooley, and J. Srivastava, "Automatic personalization based on Web usage mining," Communications of the ACM, vol. 43(8) (August 2000), pp. 142–151.

[20] I.V. Cadez, S. Gaffney, and P. Smyth, "A General Probabilistic Framework for Clustering Individuals and Objects," Proc. of the 6th ACM SIGKDD, Int'l Conf. on Knowledge Discovery and Data Mining, Boston, MA, USA, 2000, pp. 140-149.

[21] D. Chudova and P. Smyth, "Pattern Discovery in Sequences under a Markov Assumption," Proc. of the 8th ACM SIGKDD, Int'l Conf. on Knowledge Discovery and Data Mining,Edmond, Alberta, Canada, 2002, pp. 153-162.

[22] O.R. Zaiane, M. Xin, and J. Han, "Discovering Web access patterns and trends by applying OLAP and data mining technology on Web logs," Proc. of Advances in Digital Libraries Conference (ADL'98), April, Santa Barbara, CA, 1998.

[23] J. Borges and M. Levene, "Mining Association Rules in Hypertext Databases," Proc. of International Conference on Knowledge Discovery and Data Mining (KDD'98), August, 1998, pp. 149–153.

[24] M. Spiliopoulou, "The laborious way from data mining to Web mining," International Journal of Computer Systems, Science, & Engineering, vol. 14, 1999, pp. 113–126.

[25] F. Hadzic, H. Tan, and T.S. Dillon, Mining of Data with Complex Structures, Studies in Computational Intelligence Series, vol. 333, Springer, Berlin/Heidelberg, Germany, 2011.

[26] Y. Chi, S. Nijssen, R.R. Muntz, and J.N. Kok, "Frequent subtree mining an overview," Fundamenta Informaticae, Special Issue on Graph and Tree Mining, vol. 65, no. 1-2, 2005, pp. 161-198.

[27] H. Tan, F. Hadzic, T.S. Dillon, and E. Chang, "State of the art of data mining of tree structured information," International Journal of Computer Systems Science and Engineering, vol. 23, no 2, 2008.

[28] Quinlan, J.R. C4.5: Programs for Machine Learning, Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[29] G. Holmes, A. Donkin, and I.H. Witten, "Weka: A machine learning workbench," Proc. 2nd Australia and New Zealand Intelligent Information Systems Conference, Brisbane, Australia, 1994.

[30] M.J. Zaki and C.C. Aggarwal, "XRules: An Effective Structural Classifier for XML Data," Proc. of the 9th ACM SIGKDD Conference, Washington DC, USA, 20003.