Adaptive XML Tree Classification on evolving data streams

Albert Bifet and Ricard Gavaldà

Universitat Politècnica de Catalunya, Barcelona, Spain
{abifet,gavalda}@lsi.upc.edu

Abstract. We propose a new method to classify patterns, using closed and maximal frequent patterns as features. Generally, classification requires a previous mapping from the patterns to classify to vectors of features, and frequent patterns have been used as features in the past. Closed patterns maintain the same information as frequent patterns using less space and maximal patterns maintain approximate information. We use them to reduce the number of classification features. We present a new framework for XML tree stream classification. For the first component of our classification framework, we use closed tree mining algorithms for evolving data streams. For the second component, we use state of the art classification methods for data streams. To the best of our knowledge this is the first work on tree classification in streaming data varying with time. We give a first experimental evaluation of the proposed classification method.

1 Introduction

Pattern classification and frequent pattern discovery have been important tasks over the last decade. Nowadays, they are becoming harder, as the size of the pattern datasets is increasing, data often comes from sequential, streaming sources, and we cannot assume that data has been generated from a static distribution. If we want accuracy in the results of our algorithms, we have to consider that the distribution that generates data may vary over time, often in an unpredictable and drastic way.

Tree Mining is becoming an important field of research due to the fact that XML patterns are tree patterns and that XML is becoming a standard for information representation and exchange over the Internet. XML data is growing and it will soon constitute one of the largest collection of human knowledge. Other applications of tree mining appear in chemical informatics, computer vision, text retrieval, bioinformatics, and Web analysis. XML tree classification has been done traditionally using information retrieval techniques considering the labels of nodes as bags of words. With the development of frequent tree miners, classification methods using frequent trees appeared [21, 14, 8, 12]. Recently, closed frequent miners were proposed [7, 18, 1], and using them for classification tasks is the next natural step.

Closure-based mining on relational data has recently provided some interesting algorithmic developments. In this paper we use closure-based mining to reduce drastically the number of attributes in tree classification tasks. Moreover, we show how to use maximal frequent trees to reduce even more the number of attributes needed in tree classification, in many cases without loosing accuracy.

We propose a general framework to classify XML trees based on subtree occurrence. It is composed of a Tree XML Closed Frequent Miner, and a classifier algorithm. We propose specific methods for dealing with adaptive data streams. In [5] a new approach was proposed for mining closed patterns adaptively from data streams that change over time, and it was used for closed *unlabeled* rooted trees. In this work, we use the extension to the more challenging case of *labeled* rooted trees.

The rest of the paper is organized as follows. We discuss related work in Section 2. Section 3 gives background and Section 4 introduces our closure operator and its properties needed for our classification algorithm. Section 5 shows the tree classification framework and it introduces the adaptive closed frequent mining method. Experimental results are given in Section 6, and some conclusions in Section 7.

2 Related Work

Zaki and Aggarwal presented XRules in [21]. Their classification method mines frequent trees in order to create classification rules. They do not use closed frequent trees, only frequent trees. XRules is cost-sensitive and uses Bayesian rule based class decision making. They also proposed methods for effective rule prioritization and testing.

Kudo and Matsumoto presented a boosting method for tree classification in [14]. Their method consists of decision stumps that uses significant frequent subtrees as features and a Boosting algorithm which employs the subtree-based decision stumps as weak learners. With Maeda they extended this classification method to graphs in [13].

Other works use SVMs defining tree Kernels [8, 12]. Tree kernel is one of the convolutions kernels, and maps the example represented in a labeled ordered tree into all subtree spaces. The feature space uses frequent trees and not closed trees.

Garriga et al. [10] showed that when considering labeled itemsets, closed sets can be adapted for classification and discrimination purposes by conveniently contrasting covering properties on positive and negative examples. They formally proved that these sets characterize the space of relevant combinations of features for discriminating the target class.

Chi et al. proposed CMTreeMiner [7], the first algorithm to discover all closed and maximal frequent labeled induced subtrees without first discovering all frequent subtrees. CMTreeMiner shares many features with CloseGraph [19]. Termier et al. proposed DryadeParent [18], based on the hooking principle first introduced in Dryade. They claim that the branching factor and depth of the frequent patterns to find are key factors in the complexity of tree mining algorithm and that DryadeParent outperforms CMTreeMiner, on datasets where the frequent patterns have a high branching factor.

In [5], we proposed a new approach for mining closed frequent patterns adaptively from data streams that change over time, and we applied it to unlabelled frequent tree mining.

To the best of our knowledge this is the first work defined for classifying trees and mining labeled closed frequent trees in streaming data that evolve with time, and the first one in using closed and maximal frequent trees for feature reduction.

3 Preliminaries

Following Formal Concept Analysis usage, we are interested in (possibly infinite) sets endowed with a partial order relation. Elements of these sets are generically called *patterns*.

The set of all patterns will be denoted with $\mathcal{T}$, but actually all our developments will proceed in some finite subset of $\mathcal{T}$ which will act as our universe of discourse.

Given two patterns t and t', we say that t is a *subpattern* of t', or t' is a *super-pattern* of t, if $t \preceq t'$. Two patterns t, t' are said to be *comparable* if $t \preceq t'$ or $t' \preceq t$. Otherwise, they are incomparable. Also we write $t \prec t'$ if t is a proper subpattern of t' (that is, $t \preceq t'$ and $t \neq t'$).

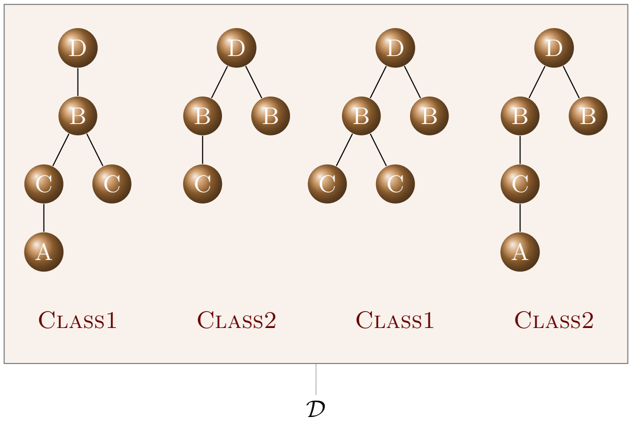
The input to our data mining process is a dataset $\mathcal{D}$ of transactions, where each transaction $s \in \mathcal{D}$ consists of a transaction identifier, *tid*, and a transaction pattern. The dataset is a finite set in the standard setting, and a potentially infinite sequence in the data stream setting. Tids are supposed to run sequentially from 1 to the size of $\mathcal{D}$. From that dataset, our universe of discourse $\mathcal{U}$ is the set of all patterns that appear as subpattern of some pattern in $\mathcal{D}$.

Figure 1 shows a finite dataset example of trees.

As is standard, we say that a transaction s *supports* a pattern t if t is a subpattern of the pattern in transaction s. The number of transactions in the dataset $\mathcal{D}$ that support t is called the *support* of the pattern t. A subpattern t is called *frequent* if its support is greater than or equal to a given threshold *min_sup*. The frequent subpattern mining problem is to find all frequent subpatterns in a given dataset. Any subpattern of a frequent pattern is also frequent and, therefore, any superpattern of a nonfrequent pattern is also nonfrequent (the *antimonotonicity* property).

3.1 Frequent Pattern Compression

We define a frequent pattern t to be *closed* if none of its proper superpatterns has the same support as it has. Generally, there are much fewer closed patterns than frequent ones. In fact, we can obtain all frequent subpatterns with their support from the set of frequent closed subpatterns with their supports. So, the



Class	Trees
Class1	$((0, D), (1, B), (2, C), (3, A), (2, C))$
Class2	$((0, D), (1, B), (2, C), (1, B))$
Class1	$((0, D), (1, B), (2, C), (2, C), (1, B))$
Class2	$((0, D), (1, B), (2, C), (3, A), (1, B))$

Fig. 1. A dataset example of 4 tree transactions

set of frequent closed subpatterns maintains the same information as the set of all frequent subpatterns.

The closed trees for the dataset of Figure 1 are shown in the Galois lattice of Figure 2.

We define a frequent pattern t to be *maximal* if none of t's proper superpatterns is frequent. All maximal patterns are closed, but not all closed patterns are maximal, so there are more closed patterns than maximal. Note that we can obtain all frequent subpatterns from the set of maximal frequent subpatterns, but not their support. So, the set of maximal frequent subpatterns maintains approximately the same information as the set of all frequent subpatterns.

4 Classification using Compressed Frequent Patterns

The pattern classification problem is defined as follows. A set of examples of the form (t, y) is given, where y is a discrete class label and t is a pattern. The goal is to produce from these examples a model $\hat{y} = f(t)$ that will predict the classes y of future pattern examples with high accuracy.

We use the following approach: we convert the pattern classification problem into a vector classification learning task, transforming patterns into vectors of attributes. Attributes will be frequent subpatterns, or a subset of these frequent subpatterns.

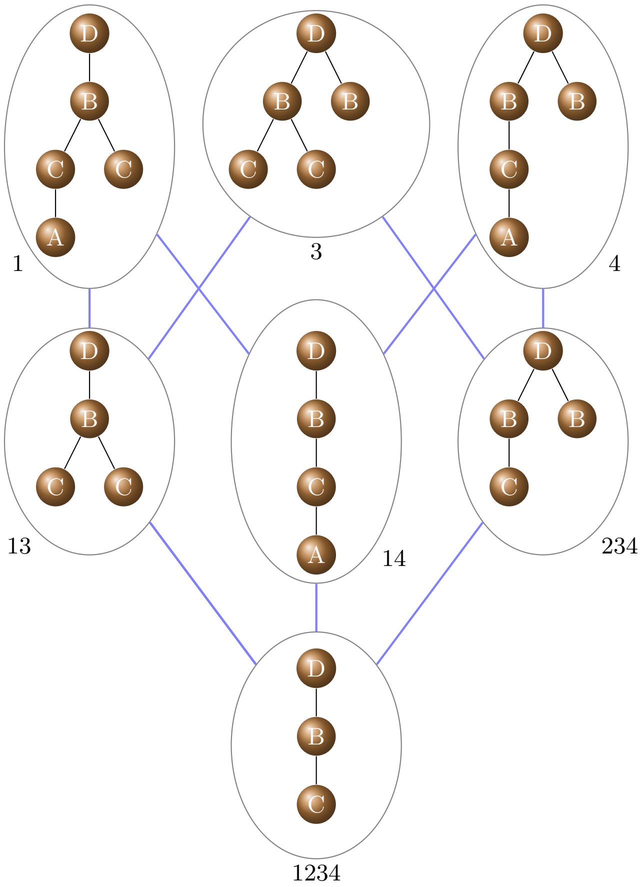


Fig. 2. Example of Galois Lattice of Closed trees

Suppose $\mathcal{D}$ has d frequent subpatterns denoted by $t_1, t_2, \ldots, t_d$. We map any pattern t to a vector x of d attributes: $x = (x_1, ..., x_d)$ such that for each attribute i, $x_i = 1$ if $t_i \preceq t$ or $x_i = 0$ otherwise.

As the number of frequent subpatterns is huge, we perform a feature selection process, selecting a subset of these frequent subpatterns, maintaining the same information, or approximate. Figures 3 and 4 show frequent trees and its conversion to vectors of attributes. Note that closed trees have the same information as frequent trees, but maximal trees loose some support information, as mentioned in Section 3.1.

4.1 Closed Frequent Patterns

Recall that if X is a set with a partial order $\leq$, a *closure operator* on X is a function $C : X \rightarrow X$ that satisfies the following for all x in X: $x \leq C(X), C(x) =$

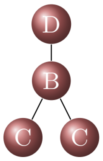
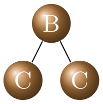
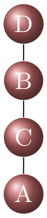
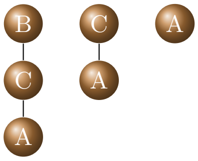
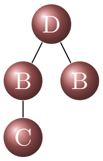
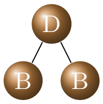
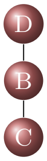
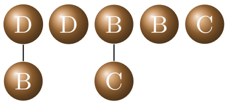
			Tree Trans.			
	Closed	Freq. not Closed Trees	1	2	3	4
c_1			1	0	1	0
c_2			1	0	0	1
c_3			0	1	1	1
c_4			1	1	1	1

Fig. 3. Frequent trees from dataset example ($min\_sup = 30\%$), and their corresponding attribute vectors.

$C(C(x))$, for all $y \in X$, $x \leq y$ implies $C(x) \leq C(y)$. A Galois connection is provided by two functions, relating two lattices in a certain way. Here our lattices are not only plane power sets of the transactions but also plain power sets of the corresponding solutions. On the basis of the binary relation $t \preceq t'$, the following definition and proposition are rather standard.

Definition 1. *The Galois connection pair is defined by:*

- *For finite* $A \subseteq \mathcal{D}$, $\sigma(A) = \{t \in \mathcal{T} \mid \forall t' \in A\, (t \preceq t')\}$
- *For finite* $B \subset \mathcal{T}$, *not necessarily in* $\mathcal{D}$, $\tau_{\mathcal{D}}(B) = \{t' \in \mathcal{D} \mid \forall t \in B\, (t \preceq t')\}$

Proposition 1. *The composition* $\Gamma_{\mathcal{D}} = \sigma \circ \tau_{\mathcal{D}}$ *is a closure operator on the subsets of* $\mathcal{D}$.

Theorem 1. *A pattern* t *is closed for* $\mathcal{D}$ *if and only if it is maximal in* $\Gamma_{\mathcal{D}}(\{t\})$.

	Frequent Trees													
	c_1		c_2				c_3		c_4					
Id	c_1	f_1^1	c_2	f_2^1	f_2^2	f_2^3	c_3	f_3^1	c_4	f_4^1	f_4^2	f_4^3	f_4^4	f_4^5
1	1	1	1	1	1	1	0	0	1	1	1	1	1	1
2	0	0	0	0	0	0	1	1	1	1	1	1	1	1
3	1	1	0	0	0	0	1	1	1	1	1	1	1	1
4	0	0	1	1	1	1	1	1	1	1	1	1	1	1

		Closed Trees				**Maximal Trees**			
Id	**Tree**	c_1	c_2	c_3	c_4	c_1	c_2	c_3	**Class**
1		1	1	0	1	1	1	0	Class1
2		0	0	1	1	0	0	1	Class2
3		1	0	1	1	1	0	1	Class1
4		0	1	1	1	0	1	1	Class2

Fig. 4. Closed and maximal frequent trees from dataset example ($min\_sup = 30\%$), and their corresponding attribute vectors.

In other words, Theorem 1 states each closed set is uniquely defined through its maximal elements. On this basis, our algorithms can avoid duplicating calculations and redundant information by just storing the maximal patterns of each closed set. We denote $\Delta(t)$ as the set of maximal patterns of each closed set of $\Gamma_{\mathcal{D}}(\{t\})$.

We can relate the closure operator to the notion of closure based on support, as previously defined, as follows: t is closed for $\mathcal{D}$ if and only if: $\Delta_{\mathcal{D}}(\{t\}) = \{t\}$.

Following the standard usage on Galois lattices, we consider now implications of the form $A \to B$ for sets of patterns A and B from $\mathcal{U}$. Specifically, we consider the following set of rules: $A \to \Gamma_{\mathcal{D}}(A)$. Alternatively, we can split the consequents into $\{A \to t \mid t \in \Gamma_{\mathcal{D}}(A)\}$.

It is easy to see that $\mathcal{D}$ obeys all these rules: for each A, any pattern of $\mathcal{D}$ that has as subpatterns all the patterns of A has also as subpatterns all the patterns of $\Gamma_{\mathcal{D}}(A)$.

Proposition 2. *Let t_i be a frequent pattern for $\mathcal{D}$. A transaction pattern t satisfies $t_i \preceq t$, if and only if it satisfies $\Delta_{\mathcal{D}}(t_i) \preceq t$.*

We use Proposition 2 to reduce the number of attributes on our classification task, using only closed frequent patterns, as they keep the same information. The attribute vector of a frequent pattern will be the same as its closed pattern attribute vector. Figure 4 shows the attribute vectors for the dataset of Figure 1.

4.2 Maximal Frequent Patterns

Maximal patterns are patterns that do not have any frequent superpattern. All maximal patterns are closed patterns. If *min_sup* is zero, then maximal patterns are transaction patterns. We denote by $M_1(t), M_2(t), \ldots, M_m(t)$ the maximal superpatterns of a pattern t. We are interested in the implications of the form $t_c \to (M_1(t) \vee M_2(t) \vee \ldots \vee M_m(t))$ where t_c is a closed pattern.

Proposition 3. *Let t_c be a closed non-maximal frequent pattern for $\mathcal{D}$. Let $M_1(t_c), M_2(t_c), \ldots, M_m(t_c)$ be the maximal superpatterns of pattern t_c. A transaction pattern t satisfies $t_c \prec t$, if and only if at least one of the maximals superpattern $M_i(t_c)$ of pattern t_c satisfies $M_i(t_c) \preceq t$.*

Proof. Suppose that pattern t_c satisfies $t_c \prec t$ but no maximal superpattern $M_i(t_c)$ satisfies $M_i(t_c) \preceq t$. Then, pattern t_c has no frequent superpattern. Therefore, it is maximal, contradicting the assumption.

Suppose, for the other direction, that a maximal superpattern $M_i(t_c)$ of t_c satisfies $M_i(t_c) \preceq t$. Then, as t_c is a $M_i(t_c)$ subpattern, $t_c \preceq M_i(t_c)$, and it holds that $t_c \preceq M_i(t_c) \preceq t$.

For non-maximal closed patterns, the following set of rules holds if t_c is not a transaction pattern:

$$t_c \to \bigvee M_i(t_c)$$

Note that for a transaction pattern t_c that it is closed and non-maximal, there is no maximal superpattern $M_i(t_c)$ of pattern t_c that satisfies $M_i(t_c) \preceq t_c$. If there are no closed non-maximal transaction patterns, we do not need to use all closed patterns as attributes, since non-maximal closed patterns may be derived from maximal patterns.

Using Proposition 3, we may reduce the number of attributes on our classification task, using only maximal frequent patterns, as they keep much of the information as closed frequent patterns.

5 XML Tree Classification framework on data streams

In this section we specialize the previous approach to the case of labelled trees, such as XML trees.

Trees are connected acyclic graphs, *rooted trees* are trees with a vertex singled out as the root, and *unranked* trees are trees with unbounded arity. We say that $t_1, \ldots, t_k$ are the *components* of tree t if t is made of a node (the root) joined to the roots of all the t_i's. We can distinguish between the cases where the components at each node form a sequence (ordered trees) or just a set (*unordered trees*). We will deal with rooted, unranked trees. We assume the presence of labels on the nodes.

An *induced subtree* of a tree t is any connected subgraph rooted at some node v of t that its vertices and edges are subsets of those of t. An *embedded subtree* of a tree t is any connected subgraph rooted at some node v of t that does not break the ancestor-descendant relationship among the vertices of t. We are interested in induced subtrees. Formally, let s be a rooted tree with vertex set V' and edge set E', and t a rooted tree t with vertex set V and edge set E. Tree s is an *induced subtree* (or simply a *subtree*) of t (written $t' \preceq t$) if and only if 1) $V' \subseteq V$, 2) $E' \subseteq E$, and 3) the labeling of V' is preserved in t. This notation can be extended to sets of trees $A \preceq B$: for all $t \in A$, there is some $t' \in B$ for which $t \preceq t'$.

Our XML Tree Classification Framework has two components:

- An XML closed frequent tree miner, for which we could use any incremental algorithm that maintains a set of closed frequent trees.
- A Data stream classifier algorithm, which we will feed with tuples to be classified online. Attributes in these tuples represent the occurrence of the current closed trees in the originating tree, although the classifier algorithm need not be aware of this.

In this section, we describe the two components of the framework, the XML closed frequent tree miner, and the data stream classifier.

5.1 Adaptive Tree Mining on evolving data streams

Using a methodology based on Galois Lattice Theory, we use three closed tree mining algorithms: an increment IncTreeMiner, a sliding-window based one, WinTreeMiner, and an algorithm that mines closed trees adaptively from data streams. It is basically an adaptation of the theoretical framework developed in [5], which deals with quite general notion of pattern and subpattern, to the case of labeled rooted trees.

For maximal frequent trees, the following properties hold:

- adding a tree transaction to a dataset of trees $\mathcal{D}$, may increase or decrease the number of maximal trees for $\mathcal{D}$.
- adding a transaction with a closed tree to a dataset of trees $\mathcal{D}$, may modify the number of maximal trees for $\mathcal{D}$.
- deleting a tree transaction from a dataset of trees $\mathcal{D}$, may increase or decrease the number of maximal trees for $\mathcal{D}$.
- deleting a tree transaction that is repeated in a dataset of trees $\mathcal{D}$ from it, may modify the number of maximal trees for $\mathcal{D}$.
- a non maximal closed tree may become maximal if
  - it was not frequent and now its support increases to a value higher or equal to min_sup
  - all of its maximal supertrees become non-frequent
- a maximal tree may become a non maximal tree if
  - its support decreases below min_sup
  - a non-frequent closed supertree becomes frequent

We could check if a closed tree becomes maximal when

- removing closed trees because they do not have enough support
- adding a new closed tree to the dataset
- deleting a closed tree from the dataset

We use three tree mining algorithms adapting the general framework for patterns presented in [5]:

- IncTreeMiner, an incremental closed tree mining algorithm,
- WinTreeMiner, a sliding window closed tree mining algorithm
- AdaTreeMiner, an adaptive closed tree mining algorithm

The batches are processed using the non-incremental algorithm explained in [3]. AdaTreeMiner is a new tree mining method for dealing with concept drift, using ADWIN [4], an algorithm for detecting change and dynamically adjusting the length of a data window. ADWIN is parameter- and assumption-free in the sense that it automatically detects and adapts to the current rate of change. Its only parameter is a confidence bound δ, indicating how confident we want to be in the algorithm's output, inherent to all algorithms dealing with random processes.

Also important for our purposes, ADWIN does not maintain the window explicitly, but compresses it using a variant of the exponential histogram technique in [9]. This means that it keeps a window of length W using only $O(\log W)$ memory and $O(\log W)$ processing time per item, rather than the $O(W)$ one expects from a naïve implementation. When windows tend to be large, this usually results in substantial memory savings.

We propose two strategies to deal with concept drift:

- AdaTreeMiner1: Using a sliding window, with an ADWIN estimator deciding the size of the window
- AdaTreeMiner2: Maintaining an ADWIN estimator for each closed set in the lattice structure.

In the second strategy, we do not delete transactions. Instead, each ADWIN monitors the support of a closed pattern. When it detects a change, we can conclude reliably that the support of this pattern seems to be changing in the data stream in recent times.

5.2 Data Stream Classifier

The second component of the framework is based on MOA. **M**assive **O**nline **A**nalysis (MOA) [11, 6] is a framework for online learning from continuous supplies of examples, such as data streams. It is closely related to the well-known WEKA project, and it includes a collection of offline and online as well as tools for evaluation. In particular, it implements boosting, bagging, and Hoeffding Trees, both with and without Naïve Bayes classifiers at the leaves.

We use bagging and boosting of decision trees, because these ensemble methods are considered the state-of-the-art classification methods.

6 Experimental evaluation

We tested our algorithms on synthetic and real data. All experiments were performed on a 2.0 GHz Intel Core Duo PC machine with 2 Gigabyte main memory, running Ubuntu 8.10.

6.1 Tree Classification

We evaluate our approach to tree classification on both real and synthetic classification data sets.

For synthetic classification, we use the tree generation program of Zaki [20], available from his web page. We generate two mother trees, one for each class. The first mother tree is generated with the following parameters: the number of distinct node labels $N = 200$, the total number of nodes in the tree $M = 1,000$, the maximal depth of the tree $D = 10$ and the maximum fanout $F = 10$. The second one has the following parameters: the number of distinct node labels $N = 5$, the total number of nodes in the tree $M = 100$, the maximal depth of the tree $D = 10$ and the maximum fanout $F = 10$.

A stream is generated by mixing the subtrees created from these mother trees. In our experiments, we set the total number of trees in the dataset to be $T = 1,000,000$. We added artificial drift changing labels of the trees every $250,000$ samples, so closed and maximal frequent trees evolve over time. We use bagging of 10 Hoeffding Trees enhanced with adaptive Naïve Bayes leaf predictions, as classification method. This adaptive Naïve Bayes prediction method monitors the error rate of majority class and Naïve Bayes decisions in every leaf, and chooses to employ Naïve Bayes decisions only where they have been more accurate in past cases.

Table 1 shows classification results. We observe that AdaTreeMiner1 is the most accurate method, and that the accuracy of WinTreeMiner depends on the size of the window.

For real datasets, we use the Log Markup Language (LOGML) dataset from Zaki et al. [16, 21], that describes log reports at their CS department website. LOGML provides a XML vocabulary to structurally express the contents of the log file information in a compact manner. Each user session is expressed in LOGML as a graph, and includes both structure and content.

The real CSLOG data set spans 3 weeks worth of such XML user-sessions. To convert this into a classification data set they chose to categorize each user-session into one of two class labels: edu corresponds to users from an "edu“ domain, while other class corresponds to all users visiting the CS department from any other domain. They separate each week's logs into a different data set (CSLOGx, where x stands for the week; CSLOG12 is the combined data for weeks 1 and 2). Notice that the edu class has much lower frequency rate than other.

Table 2 shows the results on bagging and boosting using 10 Hoeffding Trees enhanced with adaptive Naïve Bayes leaf predictions. The results are very similar for the two ensemble learning methods. Using maximals and closed frequent

Bagging	Time	Acc.	Mem.
AdaTreeMiner1	161.61	**80.06**	4.93
AdaTreeMiner2	212.57	65.78	4.42
WinTreeMiner W=100,000	192.01	72.61	6.53
WinTreeMiner W= 50,000	212.09	66.23	11.68
IncTreeMiner	212.75	65.73	4.4

Boosting	Time	Acc.	Mem.
AdaTreeMiner1	236.31	**79.83**	4.8
AdaTreeMiner2	326.8	65.43	4.25
WinTreeMiner W=100,000	286.02	70.15	5.8
WinTreeMiner W= 50,000	318.19	63.94	9.87
IncTreeMiner	317.95	65.55	4.25

Table 1. Comparison of classification algorithms. Memory is measured in MB. The best individual accuracy is indicated in boldface.

trees, we obtain results similar to [20]. Comparing maximal trees with closed trees, we see that maximal trees use 1/4 to 1/3rd of attributes, 1/3 of memory, and they perform better.

6.2 Closed Frequent Tree Labeled Mining

As far as we know, CMTreeMiner is the state-of-art algorithm for mining induced closed frequent trees in databases of rooted trees. The main difference with our approach is that CMTreeMiner is not incremental and only works with bottom-up subtrees, and our method works with both bottom-up and top-down subtrees.

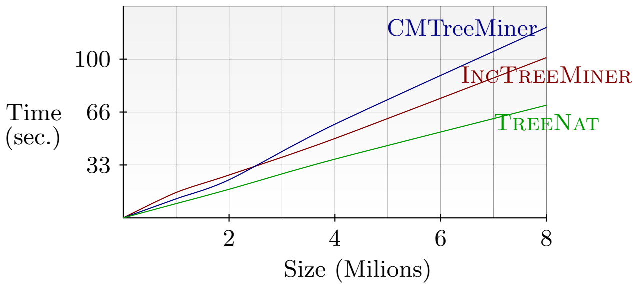


Fig. 5. Time used on ordered trees, T8M dataset

For synthetic data, we use the same dataset as in [7] and [20] for rooted ordered trees. The synthetic dataset T8M is generated by the tree generation

BAGGING		Maximal						Closed					
		Unordered			Ordered			Unordered			Ordered		
	# Trees	Att.	Acc.	Mem.	Att.	Acc.	Mem.	Att.	Acc.	Mem.	Att.	Acc.	Mem.
CSLOG12	15483	84	**79.64**	1.2	77	79.63	1.1	228	78.12	2.54	183	78.12	2.03
CSLOG23	15037	88	**79.81**	1.21	80	79.8	1.09	243	78.77	2.75	196	78.89	2.21
CSLOG31	15702	86	**79.94**	1.25	80	79.87	1.17	243	77.6	2.73	196	77.59	2.19
CSLOG123	23111	84	**80.02**	1.7	78	79.97	1.58	228	78.91	4.18	181	78.91	3.31

BOOSTING		Maximal						Closed					
		Unordered			Ordered			Unordered			Ordered		
	# Trees	Att.	Acc.	Mem.	Att.	Acc.	Mem.	Att.	Acc.	Mem.	Att.	Acc.	Mem.
CSLOG12	15483	84	**79.46**	1.21	77	78.83	1.11	228	75.84	2.97	183	77.28	2.37
CSLOG23	15037	88	79.91	1.23	80	**80.24**	1.14	243	77.24	2.96	196	78.99	2.38
CSLOG31	15702	86	**79.77**	1.25	80	79.69	1.17	243	76.25	3.29	196	77.63	2.62
CSLOG123	23111	84	79.73	1.69	78	**80.03**	1.56	228	76.92	4.25	181	76.43	3.45

Table 2. Comparison of tree classification algorithms. Memory is measured in MB. The best individual accuracies are indicated in boldface (one per row).

program of Zaki [20], used for the evaluation on tree classification. In brief, a mother tree is generated first with the following parameters: the number of distinct node labels $N = 100$, the total number of nodes in the tree $M = 10,000$, the maximal depth of the tree $D = 10$ and the maximum fanout $F = 10$. The dataset is then generated by creating subtrees of the mother tree. In our experiments, we set the total number of trees in the dataset to be from $T = 0$ to $T = 8,000,000$.

The results of our experiments on synthetic data are shown in Figures 5,6,7, and 8. We observe that as the data size increases, the running times of IncTreeMiner and CMTreeMiner become closer, and that IncTreeMiner uses much less memory than CMTreeMiner. CMTreeMiner failed in our experiments when dataset size reached 8 million trees: not being an incremental method, it must store the whole dataset in memory all the time *in addition* to the lattice structure, in contrast with our algorithms.

In Figure 9 we compare WinTreeMiner with different window sizes to AdaTreeMiner on T8M dataset. We observe that the two versions of AdaTreeMiner outperform WinTreeMiner for all window sizes.

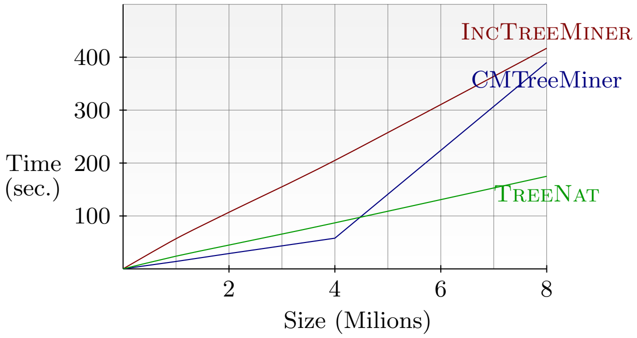


Fig. 6. Time used on unordered trees, TN1 dataset

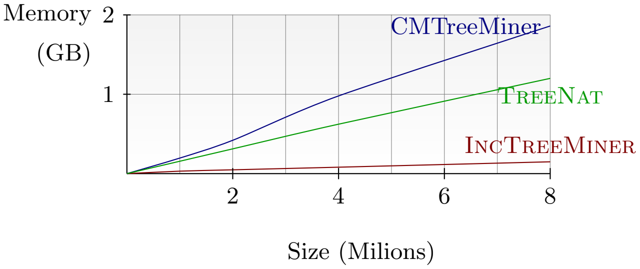


Fig. 7. Memory used on ordered trees, T8M dataset

7 Conclusions

The scheme for classification based on our methods, efficiently selects a reduced number of attributes, and achieves higher accuracy (even more in the more selective case in which we keep only attributes corresponding to maximal trees). Our approach to tree mining outperforms CMTreeMiner in time and memory consumption when the number of trees is huge, because CMTreeMiner is not an incremental method and it must store the whole dataset in memory all the time.

Song et al.[17] introduced the concept of relaxed frequent itemset using the notion of relaxed support. We can see relaxed support as a mapping from all possible dataset supports to a set of relaxed intervals. Relaxed closed mining is a powerful notion that reduces the number of closed subpatterns. We introduced the concept of logarithmic relaxed frequent pattern in [5]. Future work will be to apply this notion to our classification method by introducing an attribute for each relaxed frequent closed pattern, instead of one for each closed frequent pattern. Also, we would like to apply these classification methods to other kinds of patterns.

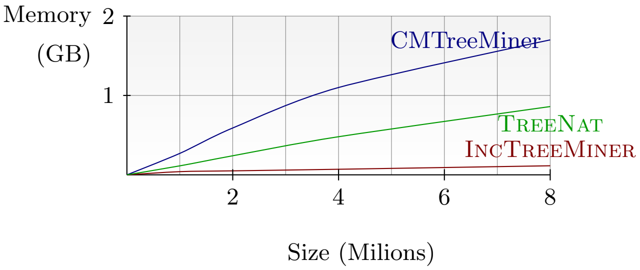


Fig. 8. Memory used on unordered trees on T8M dataset

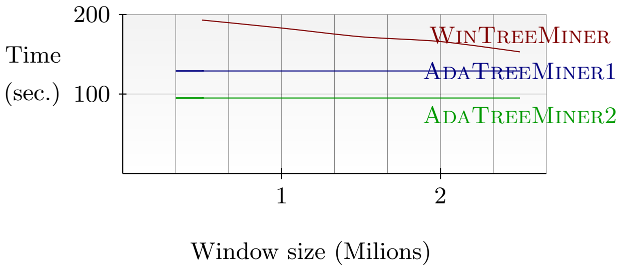


Fig. 9. Time used on ordered trees on T8M dataset varying window size

And finally, we would like to extend our work to generators [15]. In [2] the authors were interested in implications of trees of the form $G \rightarrow Z$, where G is a generator of Z. When $\Gamma(G) = Z$ for a set of trees $G \neq Z$ and G is minimal among all the candidates with closure equal to Z, we say that G is a generator of Z. Generator based representations contain the same information as the frequent closed ones. In the literature, there is no method for finding generators of trees in evolving data streams. As future work, we would like to compare classification using tree generators, with the classification methods presented in this paper.

8 Acknowledgments

Partially supported by the EU PASCAL2 Network of Excellence (FP7-ICT-216886), and by projects SESAAME-BAR (TIN2008-06582-C03-01), MOISES-BAR (TIN2005-08832-C03-03). Albert Bifet has been supported by a FI grant through the Grups de Recerca Consolidats (SGR) program of Generalitat de Catalunya.

References

[1] H. Arimura and T. Uno. An output-polynomial time algorithm for mining frequent closed attribute trees. In *ILP*, pages 1–19, 2005.

[2] J. L. Balcázar, A. Bifet, and A. Lozano. Mining implications from lattices of closed trees. In *Extraction et gestion des connaissances (EGC'2008)*, pages 373–384, 2008.

[3] J. L. Balcázar, A. Bifet, and A. Lozano. Mining frequent closed rooted trees. *Accepted for publication in Machine Learning Journal*, 2009.

[4] A. Bifet and R. Gavaldà. Learning from time-changing data with adaptive windowing. In *SIAM International Conference on Data Mining*, 2007.

[5] A. Bifet and R. Gavaldà. Mining adaptively frequent closed unlabeled rooted trees in data streams. In *14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

[6] A. Bifet, G. Holmes, B. Pfahringer, R. Kirkby, and R. Gavaldà. New ensemble methods for evolving data streams. In *15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 2009.

[7] Y. Chi, Y. Xia, Y. Yang, and R. Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *Fundamenta Informaticae*, XXI:1001–1038, 2001.

[8] M. Collins and N. Duffy. New ranking algorithms for parsing and tagging: kernels over discrete structures, and the voted perceptron. In *ACL '02*, pages 263–270, 2001.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 14(1):27–45, 2002.

[10] G. C. Garriga, P. Kralj, and N. Lavrač. Closed sets for labeled data. *J. Mach. Learn. Res.*, 9:559–580, 2008.

[11] G. Holmes, R. Kirkby, and B. Pfahringer. MOA: Massive Online Analysis. `http://sourceforge.net/projects/moa-datastream`. 2007.

[12] H. Kashima and T. Koyanagi. Kernels for semi-structured data. In *ICML*, pages 291–298, 2002.

[13] T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *NIPS*, 2004.

[14] T. Kudo and Y. Matsumoto. A boosting algorithm for classification of semi-structured text. In *EMNLP*, pages 301–308, 2004.

[15] J. Li, H. Li, L. Wong, J. Pei, and G. Dong. Minimum description length principle: Generators are preferable to closed patterns. In *AAAI*, 2006.

[16] J. Punin, M. Krishnamoorthy, and M. Zaki. LOGML: Log markup language for web usage mining. In *WEBKDD Workshop (with SIGKDD)*, 2001.

[17] G. Song, D. Yang, B. Cui, B. Zheng, Y. Liu, and K. Xie. CLAIM: An efficient method for relaxed frequent closed itemsets mining over stream data. In *DASFAA*, pages 664–675, 2007.

[18] A. Termier, M.-C. Rousset, M. Sebag, K. Ohara, T. Washio, and H. Motoda. DryadeParent, an efficient and robust closed attribute tree mining algorithm. *IEEE Trans. Knowl. Data Eng.*, 20(3):300–320, 2008.

[19] X. Yan and J. Han. CloseGraph: mining closed frequent graph patterns. In *KDD '03*, pages 286–295, New York, NY, USA, 2003. ACM Press.

[20] M. J. Zaki. Efficiently mining frequent trees in a forest. In *KDD '02*, 2002.

[21] M. J. Zaki and C. C. Aggarwal. XRules: an effective structural classifier for xml data. In *KDD '03*, pages 316–325, New York, NY, USA, 2003. ACM.