Mach Learn (2006) 62: 137–170
DOI 10.1007/s10994-006-5832-2

XRules: An effective algorithm for structural classification of XML data

Mohammed J. Zaki · Charu C. Aggarwal

Received: 1 August 2004 / Revised: 31 January 2005 / Accepted: 28 June 2005 /
Published online: 27 January 2006

Abstract XML documents have recently become ubiquitous because of their varied applicability in a number of applications. Classification is an important problem in the data mining domain, but current classification methods for XML documents use IR-based methods in which each document is treated as a bag of words. Such techniques ignore a significant amount of information hidden inside the documents. In this paper we discuss the problem of rule based classification of XML data by using frequent discriminatory substructures within XML documents. Such a technique is more capable of finding the classification characteristics of documents. In addition, the technique can also be extended to cost sensitive classification. We show the effectiveness of the method with respect to other classifiers. We note that the methodology discussed in this paper is applicable to any kind of semi-structured data.

Keywords XML/Semi-structured data · Classification · Rule induction · Tree mining

1. Introduction

The classification problem is defined as follows. We have an input data set called the *training data* which consists of a set of multi-attribute records along with a special variable called the *class*. This class variable draws its value from a discrete set of classes. The training data is used to construct a model which relates the feature variables in the training data to the

Editors: Hendrik Blockeel, David Jensen and Stefan Kramer

M. J. Zaki (✉)
Rensselaer Polytechnic Institute, Troy NY 12180
e-mail: zaki@cs.rpi.edu
Tel.: 518-276-6340
Fax: 518-276-4033

C. Aggarwal
IBM T.J. Watson Research Center, Yorktown Heights, NY 10598
e-mail: charu@us.ibm.com

class variable. The *test instances* for the classification problem consist of a set of records for which only the feature values are known while the class value is unknown. The training model is used in order to predict the class variable for such test instances.

In recent years, XML has become a popular way of storing many data sets because the semi-structured nature of XML allows the modeling of a wide variety of databases as XML documents. XML data thus forms an important data mining domain, and it is valuable to develop classification methods for such data. Currently, the problem of classification on XML data has not been very well studied, in spite of its applicability to a wide variety of problems in the XML domain.

Since XML documents are also text documents, a natural alternative for such cases is the use of standard information retrieval methods for classification. A simple and frequently used method for classification is the nearest neighbor classifier (Duda & Hart, 1973). This method works quite well for most text applications containing a small number of class labels. Other methods like Support Vector Machines (SVM) (Joachims, 2002) and Latent Semantic Analysis (Dumais *et al*., 1988) have proven to be successful in text classification. However, classifying using only the text ignores a significant amount of structural information in the XML documents. The classification behavior of the XML document may be hidden in the structural information available inside the document. In such cases, the use of IR based classifiers is likely to be ineffective for XML documents. Another approach for XML mining is to directly use association rule based classifiers such as CBA (Liu, Hsu, & Ma, 1998), CAEP (Dong *et al*., 1999) or CMAR (Li, Han, & Pei, 2001), on the XML data. Even though an XML data record has hierarchical structure, its structure can be flattened out into a set, which allows the use of an association classifier. However, this also results in loss of structural information.

In this paper, we will discuss the problem of constructing *structural rules* in order to perform the classification task. The training phase finds the structures which are most closely related to the class variable. In other words, the presence of a particular kind of structural pattern in an XML document is related to its likelihood of belonging to a particular class. Once the training phase has been completed, we perform the testing phase in which these rules are used to perform the structural classification. We will show that the resulting system is significantly more effective than an association based classifier because of its ability to mine discriminatory structures in the data.

The main contribution of this paper is to propose XRules, a structural rule-based classifier for semi-structured data. In order to do so, we also develop XMiner which mines pertinent structures for multiple classes simultaneously. We extend our classifier to the cost sensitive case, so that it can handle normal as well as skewed class distributions. We also show that our class assignment decisions are rooted in Bayesian statistics.

2. Structural rules: concepts

2.1. XML as trees

A *rooted, labeled, tree*, $T = (V, E)$ is a directed, acyclic, connected graph, with $V = \{0, 1, \ldots, n\}$ as the set of vertices or nodes, $E = \{(x, y)|x, y \in V\}$ as the set of edges. One distinguished vertex $r \in V$ is designated the *root*, and for all $x \in V$, there is a *unique* path from r to x. Further, $l : V \rightarrow L$ is a labeling function mapping vertices to a set of *labels* $L = \{\ell_1, \ell_2, \ldots\}$. The *size* of T is the number of nodes in T. In an *ordered tree* the children of each vertex are ordered (i.e., if a vertex has k children, then we can designate them as

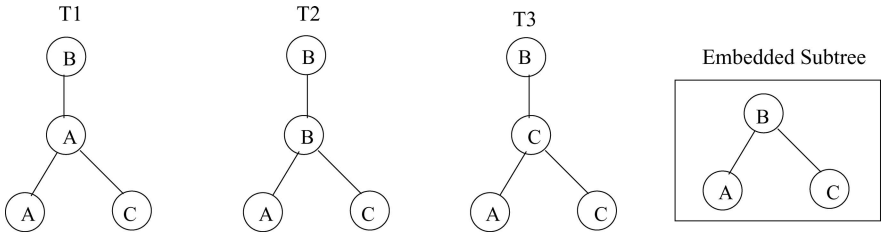


Fig. 1 Embedded subtree.

the first child, second child, and so on up to the *k*th child), otherwise, the tree is *unordered*. In this paper, we model XML documents as ordered, labeled, rooted trees i.e., child order matters, and each node has a label. This is justified since tree structured XML documents are the most widely occurring ones in real applications. We note that even if an XML document is not a tree, it can always be converted into one by using a node splitting methodology (Anderson *et al.*, 2002). Further, we do not distinguish between attributes and elements of an XML document; both are mapped to the label set.

If $x, y \in V$ and there is a path from x to y, then x is called an *ancestor* of y (and y a *descendant* of x), denoted as $x \preceq_p y$, where p is the length of the path from x to y. If $x \preceq_1 y$ (i.e., x is an immediate ancestor), then x is called the *parent* of y, and y the *child* of x. If x and y have the same parent, x and y are called *siblings*, and if they have a common ancestor, they are called *cousins*.

Subtrees Given trees $T = (V_t, E_t)$ and $S = (V_s, E_s)$, we say that T is an *isomorphic subtree* of S *iff* [1] there exists a one-to-one mapping $\varphi : V_t \to V_s$, such that $(x, y) \in E_t$ iff $(\varphi(x), \varphi(y)) \in E_s$. If φ is onto, then T and S are called *isomorphic*. T is called an *induced subtree* of $S = (V_s, E_s)$, denoted $T \preceq_i S$, iff T is an isomorphic subtree of S, and φ preserves labels, i.e., $l(x) = l(\varphi(x)), \forall x \, in V_t$. That is, for induced subtrees φ preserves the parent-child relationships, as well as vertex labels. The induced subtree obtained by deleting the rightmost leaf in T is called an *immediate prefix* of T. The induced subtree obtained from T by a series of rightmost node deletions is called a *prefix* of T.

$T = (V_t, E_t)$ is called an *embedded subtree* of $S = (V_s, E_s)$, denoted as $T \preceq_e S$ iff there exists a 1-to-1 mapping $\varphi : V_t \to V_s$ that satisfies: i) $(x, y) \in E_t$ iff $\varphi(x) \preceq_p \varphi(y)$, and ii) $l(x) = l(\varphi(x))$. That is, for embedded subtrees φ preserves ancestor-descendant relationships and labels. Embedded subtrees are thus a generalization of induced subtrees; they allow not only direct parent-child branches, but also ancestor-descendant branches. As such embedded subtrees are able to extract patterns "hidden" (or embedded) deep within large trees which might be missed by the traditional definition. As an example, consider Figure 1, which shows three trees. Let us assume we want to mine subtrees that are common to all three trees (i.e., 100% frequency). If we mine induced trees only, then there are no frequent trees of size more than one. On the other hand, if we mine embedded subtrees, then the tree shown in the box is a frequent pattern appearing in all three trees; it is obtained by skipping the "middle" node in each tree. This example shows why embedded trees are of interest. Henceforth, a reference to subtree should be taken to mean an embedded subtree, unless indicated otherwise.

Support If $T \preceq_e S$, we also say that S *contains* T or T *occurs* in S. Note that each occurrence of T in S can be identified by its unique *match label*, given by the sequence $\varphi(x_0)\varphi(x_1)\cdots\varphi(x_{|T|})$, where $x_i \in V_t$. That is, a match label of T is given as the set of

[1] if and only if

matching positions in S. Let $\delta_S(T)$ denote the number of occurrences of the subtree T in a tree S. Let d_S be an indicator variable, with $d_S(T) = 1$ if $\delta_S(T) > 0$ and $d_S(T) = 0$ if $\delta_S(T) = 0$.

Let D denote a database (a *forest*) of trees. For a tree T, we define its *absolute support* in D, denoted $\pi^A(T, D)$, as the number of trees in D that contain at least one occurrence of T, i.e.,

$$\pi^A(T, D) = \sum_{S \in D} d_S(T) = |\{S \in D | T \preceq S\}| \tag{1}$$

The *weighted support* of T is defined as $\pi^W(T, D) = \sum_{S \in D} \delta_S(T)$, i.e., total number of occurrences of T over all trees in D. The *(relative) support* of T in D, denoted $\pi(T, D)$, is defined as the fraction of trees in D that contain T, i.e.,

$$\pi(T, D) = \frac{\pi^A(T, D)}{|D|} \tag{2}$$

T is said to be *frequent* in D if $\pi(T, D) \geq \pi^{\min}$, where $\pi^{\min}$ is a user defined minimum support threshold. We denote by F_k the set of all frequent subtrees of size k, which are also called as k-(sub)trees. In some domains one might be interested in using weighted support, instead of support. Both of them are allowed in our mining approach, but we focus mainly on support.

2.2. Cost-based classification

The classification model discussed in this paper can be used for the general case of cost-sensitive classification (Domingos, 1999). In this section, we provide some definitions relevant to this topic. We assume that the training database $\mathcal{D}$ for classification consists of a set of $|\mathcal{D}|$ structures, each of which is associated with one of k class variables. Let $\mathcal{C} = \{c_1 \ldots c_k\}$ be the k classes in the data. For a structure $T \in \mathcal{D}$, we use the notation $T.c$ to refer to the class associated with T. We assume that each of these structures is an XML document that can be represented in tree format. Therefore the database $\mathcal{D}$ is essentially a forest with N components, so that each of the trees in the forest is labeled with a class variable. The class label of each structure in $\mathcal{D}$ induces a partition of the database into k disjoint parts. Let $\mathcal{D}_i = \{T \in \mathcal{D} | T.c = c_i\}$, i.e., $\mathcal{D}_i$ consists of all structures with class c_i. Clearly $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$.

The goal of classification is to learn a model, $\mathcal{R} : \mathcal{D} \rightarrow \mathcal{C}$, $\mathcal{R}(T) = c_j$ (where $T \in \mathcal{D}$ and $c_j \in \mathcal{C}$), that can predict the class label for an unlabeled test instance. We can find out how well the classifier performs by measuring its accuracy. Let $\mathcal{D}$ be some collection of structures T with known labels $T.c$. Let $\eta(\mathcal{D}) = |\{T \in \mathcal{D} | T.c = \mathcal{R}(T)\}|$ denote the number of correct predictions made by the model for examples in $\mathcal{D}$. Thus, $\eta(\mathcal{D}_i)$ gives the number of correct predictions for examples with class c_i, and $\eta(\mathcal{D}) = \sum_{i=1}^{k} \eta(\mathcal{D}_i)$ gives the total number of correct predictions made by $\mathcal{R}$ over all classes. The accuracy α of the classification model $\mathcal{R}$ on data set $\mathcal{D}$ is the ratio of correct predictions to the total number of predictions made: $\alpha(\mathcal{R}, \mathcal{D}) = \frac{\eta(\mathcal{D})}{|\mathcal{D}|}$.

Cost-sensitive accuracy For many classifier models, the accuracy is often biased in favor of classes with higher probability of occurrence. In many real applications, the cost of predicting each class correctly is not the same, and thus it is preferable to use the notion of

cost-sensitive accuracy. For each class c_i, let w_i denote a positive real number called *weight*, with the constraint that $\sum_{i=1}^{k} w_i = 1$. The *cost-sensitive accuracy*, denoted α^{cs}, is defined as the weighted average of the accuracy of the classifier on each class. Formally, we define

$$\alpha^{cs}(\mathcal{R}, \mathcal{D}) = \sum_{i=1}^{k} (w_i \times \alpha(\mathcal{R}, \mathcal{D}_i)) \tag{3}$$

There are several cost-models that one could use to compute the classification accuracy:

- The *proportional* model uses $w_i = |\mathcal{D}_i|/|\mathcal{D}|$, i.e., weights are proportional to the probability of the class in $\mathcal{D}$.
- The *equal* model uses $w_i = 1/k$, i.e., all classes are weighted equally.
- The *inverse* model uses $w_i = \frac{1/|\mathcal{D}_i|}{\sum_{j=1}^{k} 1/|\mathcal{D}_j|}$, i.e., weights are inversely proportional to the class probability.
- The *custom* model uses user-defined weights w_i.

Lemma 2.1. *For proportional model* $\alpha^{cs}(\mathcal{R}, \mathcal{D}) = \alpha(\mathcal{R}, \mathcal{D})$.

Proof: $\alpha^{cs}(\mathcal{R}, \mathcal{D}) = \sum_{i=1}^{k} (\frac{|\mathcal{D}_i|}{|\mathcal{D}|} \alpha(\mathcal{R}, \mathcal{D}_i)) = \sum_{i=1}^{k} \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \times \frac{\eta(\mathcal{D}_i)}{|\mathcal{D}_i|} = \sum_{i=1}^{k} \frac{\eta(\mathcal{D}_i)}{|\mathcal{D}|} = \frac{\eta(\mathcal{D})}{|\mathcal{D}|} = \alpha(\mathcal{R}, \mathcal{D})$ □

In this paper we will contrast the inverse cost-model with the proportional and equal model. The inverse model works well for binary classification problems with skewed class distribution, since it gives a higher reward to a correct rare class prediction.

2.3. Rule support

Rules are defined as entities which relate the frequent structures on the left hand side to the class variables on the right. Such rules are able to relate the complex structural patterns in the data to the class variable. Formally, a *structural rule* is an entity of the form $T \Rightarrow c_i$, where T is a structure, and c_i is one of the k classes.

This rule implies that if T is a substructure of a given XML record x, then the record x is more likely to belong to the class c_i. The "goodness" of such an implication is defined by two parameters which we refer to as support and strength.

Let D be any collection of trees with class labels drawn from C. The *global support* of $T \Rightarrow c_i$ in the database $\mathcal{D}$, is defined as the joint probability of T and c_i, i.e., the percentage of the trees in the database containing T and having class label c_i. Formally

$$\pi(T \Rightarrow c_i) = P(T \wedge c_i) = \frac{\pi^A(T, \mathcal{D}_i)}{|\mathcal{D}|} = \pi(T, \mathcal{D}_i) \times \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \tag{2.4}$$

The last step follows from Equation (2). The *local support* of a rule $T \Rightarrow c_i$ is simply its relative frequency in $\mathcal{D}_i$, given as $\pi(T, \mathcal{D}_i)$

.

2.4. Rule strength

The strength of a structural rule can be measured by different measures; we focus on three: confidence, likelihood ratio, and weighted confidence, as defined below.

2.4.1. Confidence

The *confidence* of the structural rule $T \Rightarrow c_i$ is defined as the conditional probability of class c_i given T, i.e., the ratio of the number of trees containing T and having class label c_i, to the number of trees containing T in the entire database. Formally, we define

$$\rho(T \Rightarrow c_i) = P(c_i|T) = \frac{P(T \wedge c_i)}{P(T)} = \frac{\pi^A(T, \mathcal{D}_i)}{\pi^A(T, \mathcal{D})} \tag{5}$$

Let us assume that we have k classes ($k \geq 2$), and let $\bar{\mathcal{C}}_i = \mathcal{C} - \{c_i\}$ be the set of all classes other than c_i. We define $\bar{D}_i = \mathcal{D} - \mathcal{D}_i$ to be the set of trees in $\mathcal{D}$ with their classes taken from $\bar{\mathcal{C}}_i$. Our approach for multi-class problems (with $k > 2$) is to treat them as a binary class problem as follows: we compare each class c_i with the rest of the classes taken as a group to form a negative class $\bar{\mathcal{C}}_i$. That is, we compare $\rho(T \Rightarrow c_i)$ with $\rho(T \Rightarrow \bar{\mathcal{C}}_i)$. Using the observation that $\mathcal{D} = \mathcal{D}_i + \bar{D}_i$, we can rewrite Equation (5) as:

$$\rho(T \Rightarrow c_i) = \frac{\pi^A(T, \mathcal{D}_i)}{\pi^A(T, \mathcal{D}_i) + \pi^A(T, \bar{D}_i)} \tag{6}$$

It is clear that $\rho(T \Rightarrow c_i) = 1 - \rho(T \Rightarrow \bar{\mathcal{C}}_i)$.

2.4.2. Likelihood ratio

The *likelihood ratio* for a rule $T \Rightarrow c_i$ is defined as the ratio of the relative support of T in examples with class c_i, to the relative support of T in examples having negative class $\bar{\mathcal{C}}_i$. Formally, it is defined as follows:

$$\gamma(T \Rightarrow c_i) = \frac{\pi(T, \mathcal{D}_i)}{\pi(T, \overline{\mathcal{D}}_i)} = \frac{\pi^A(T, \mathcal{D}_i)}{\pi^A(T, \overline{\mathcal{D}}_i)} \times \frac{|\bar{D}_i|}{|\mathcal{D}_i|} \tag{7}$$

Lemma 2.2. *Likelihood ratio for a rule is related to its confidence by the formula:*

$$\gamma(T \Rightarrow c_i) = \frac{\rho(T \Rightarrow c_i)}{\rho(T \Rightarrow \overline{\mathcal{C}}_i)} \times \frac{|\overline{\mathcal{D}}_i|}{|\mathcal{D}_i|}$$

Proof: From Equation. (5), we get $\pi^A(T, \mathcal{D}_i) = \rho(T \Rightarrow c_i) \times \pi^A(T, \mathcal{D})$ (similarly for $\pi^A(T, D_i)$). Plugging into Equation (7),

$$\gamma(T \Rightarrow c_i) = \frac{\rho(T \Rightarrow c_i) \times \pi^A(T, \mathcal{D})}{\rho(T \Rightarrow \overline{\mathcal{C}}_i) \times \pi^A(T, \mathcal{D})} \frac{|\overline{\mathcal{D}}_i|}{|\mathcal{D}_i|} = \frac{\rho(T \Rightarrow c_i)}{\rho(T \Rightarrow \overline{\mathcal{C}}_i)} \frac{|\overline{\mathcal{D}}_i|}{|\mathcal{D}_i|}$$

□

2.4.3. Weighted confidence

We define another measure called the *weighted confidence*, which combines the above two measures, given as follows:

$$\rho^w(T \Rightarrow c_i) = \frac{\pi(T, \mathcal{D}_i)}{\pi(T, \mathcal{D}_i) + \pi(T, \overline{\mathcal{D}}_i)} \tag{8}$$

We can rewrite the Equation (8), as a weighted version of Equation (6), as follows:

$$\rho^w(T \Rightarrow c_i) = \frac{\pi^A(T, \mathcal{D}_i)/|\mathcal{D}_i|}{\pi^A(T, \mathcal{D}_i)/|\mathcal{D}_i| + \pi^A(T, \overline{\mathcal{D}}_i)/|\overline{\mathcal{D}}_i|}$$

In other words, while confidence uses absolute supports, weighted confidence uses relative supports (i.e., weighted by class probability). By the next lemma, weighted confidence can also be thought of as a normalized likelihood measure.

Lemma 2.3. *The weighted confidence of a rule is related to its likelihood by the formula:*

$$\rho^w(T \Rightarrow c_i) = \frac{\gamma(T \Rightarrow c_i)}{\gamma(T \Rightarrow c_i) + 1} \tag{9}$$

Proof: From Equation (7), $\pi(T, \mathcal{D}_i) = \gamma(T \Rightarrow c_i) \times \pi(T, \overline{\mathcal{D}}_i)$. Plugging into Equation (8), we get:

$$\rho^w(T \Rightarrow c_i) = \frac{\gamma(T \Rightarrow c_i) \times \pi(T, \overline{\mathcal{D}}_i)}{\gamma(T \Rightarrow c_i) \times \pi(T, \overline{\mathcal{D}}_i) + \pi(T, \overline{\mathcal{D}}_i)} = \frac{\gamma(T \Rightarrow c_i)}{\gamma(T \Rightarrow c_i) + 1}$$

□

Both ρ and ρ^w take on values between [0, 1], while γ can take values between $[0, \infty]$. Since we want only predictive rules, we need to remove any rule that lacks predictive power. Consider a rule $(T \Rightarrow c_i)$; if its (weighted) confidence $\rho = \rho^w = 0.5$ or if its likelihood ratio $\gamma = 1.0$, then T cannot distinguish between the class c_i and its negative class $\bar{C}_i$, and we prune such a rule. In general, the acceptable range of values for a user-defined minimum confidence threshold is $\rho^{\min} \in (0.5, 1]$, while the acceptable range for minimum likelihood is $\gamma^{\min} \in (1, \infty]$. In our experiments, we will study the effects of using one strength measure over another. Let δ denote the measure of strength. For convenience, for confidence we set $\delta \equiv \rho$ (note: the notation $\equiv$ denotes that the two entities are equivalent), for weighted confidence we set $\delta \equiv \rho^w$, and for likelihood $\delta \equiv \gamma$.

We use the notation $T \Rightarrow \pi, \delta c_i$ to denote a rule with support π and strength δ. Our goal is to learn a structural rule-set $\mathcal{R} = \{R^1, R^2, \ldots, R^m\}$, where each rule is of the form $R^j : T^j \Rightarrow \pi, \delta c_i^j$, with $\pi \geq \pi_i^{\min}$ (i.e., rule support is equal to or more than the minimum support for class c_i) and with $\delta \geq \delta^{\min}$. That is, rules which satisfy a user-defined level of minimum support $\pi_i^{\min}$, and a global minimum strength threshold, $\delta^{\min}$. Note that $\delta^{\min} \equiv \rho^{\min}$ for (weighted) confidence based measure and $\delta^{\min} \equiv \gamma^{\min}$ for likelihood based measure. We

set the default minimum strength values to $\rho^{\min} = 0.5$ and $\gamma^{\min} = 1.0$; in fact, these are the values used in our experiments.

2.4.4. *Bayesian interpretation of strength*

Let $\mathcal{C} = \{c_1, \ldots, c_k\}$ be the set of k classes, and let $\bar{\mathcal{C}}_i = \mathcal{C} - c_i$. As before $\mathcal{D}_i$ is the portion of data set $\mathcal{D}$ with class c_i and $\overline{\mathcal{D}}_i$ is the remaining data set, with class in $\overline{\mathcal{C}}_i$. Given k classes, in general, an unseen example T should be assigned to class c_i if the probability of class c_i given T, $P(c_i|T)$ is the greatest over all other classes c_j, i.e., assign T to class c_i if $P(c_i|T) > P(c_j|T)$ for all $c_j \in \mathcal{C} \setminus c_j$. Since we compare a class c_i against the negative class $\overline{\mathcal{C}}_i$, we assign T to class c_i if

$$P(c_i|T) > P(\overline{\mathcal{C}}_i|T) \tag{10}$$

$$\Leftrightarrow \frac{P(T|c_i)P(c_i)}{P(T)} > \frac{P(T|\overline{\mathcal{C}}_i)P(\overline{\mathcal{C}}_i)}{P(T)} \quad \text{Bayes thm.} \tag{11}$$

$$\Leftrightarrow P(T|c_i)P(c_i) > P(T|\overline{\mathcal{C}}_i)P(\overline{\mathcal{C}}_i) \tag{12}$$

The three strength measures differ in which equation they use for class prediction. For instance, confidence measure directly uses Equation (10), since by definition (Equation (5)), $\rho(T \Rightarrow c_i) = P(c_i|T)$. Thus using the confidence measure T is assigned to class c_i if $\rho(T \Rightarrow c_i) > \rho(T \Rightarrow \overline{\mathcal{C}}_i)$.

The likelihood measure uses Equation (12). Rearranging the terms in Equation (12), we get $\frac{P(T|c_i)}{P(T|\overline{\mathcal{C}}_i)} > \frac{P(\overline{\mathcal{C}}_i)}{P(c_i)}$. Substituting $P(T|c_i) = \frac{\pi^A(T,\mathcal{D}_i)}{|\mathcal{D}_i|} = \pi(T, \mathcal{D}_i)$ (and similarly for $P(T|\overline{\mathcal{C}}_i)$) we get: $\frac{\pi(T,\mathcal{D}_i)}{\pi(T,\overline{\mathcal{D}}_i)} > \frac{P(\overline{\mathcal{C}}_i)}{P(c_i)}$. By definition of likelihood (Equation (7)), we have $\gamma(T \Rightarrow c_i) = \frac{\pi(T,\mathcal{D}_i)}{\pi(T,cal\overline{D}_i)}$. Thus Bayes rule (Equation (12)) assigns T to class c_i if $\gamma(T \Rightarrow c_i) > \frac{P(\overline{\mathcal{C}}_i)}{P(c_i)}$.

The likelihood measure assigns T to class c_i if $\gamma(T \Rightarrow c_i) > \gamma^{\min}$. If we use the default value of $\gamma^{\min} = 1$, this corresponds to ignoring the ratio of class prior probabilities, i.e., setting the ratio $\frac{P(\bar{\mathcal{C}}_i)}{P(c_i)} = 1$. In general (for proportional or equal cost model) it makes sense to use the class priors, since in the absence of any information, we should predict the class of T to be the class with higher prior. However, if c_i is rare (inverse cost model), then it is better to ignore the prior, since the prior ratio is biased in favor of the class with higher probability. By setting the prior ratio to 1, we set all classes on an equal footing.

Finally, the weighted confidence measure uses Equation 12 (consider its LHS):

$$\begin{aligned}
\text{LHS} &= \frac{P(T|c_i)P(c_i)}{P(T)} = \frac{P(T|c_i)P(c_i)}{P(T|c_i)P(c_i) + P(T|\bar{\mathcal{C}}_i)P(\bar{\mathcal{C}}_i)} \\
&= \frac{1}{1 + \frac{P(T|\bar{\mathcal{C}}_i)P(\bar{\mathcal{C}}_i)}{P(T|c_i)P(c_i)}} = \frac{1}{1 + \frac{\pi(T,\overline{\mathcal{D}}_i)}{\pi(T,\mathcal{D}_i)} \times \frac{P(\bar{\mathcal{C}}_i)}{P(c_i)}} \\
&\quad \text{setting } \frac{\pi(T,\overline{\mathcal{D}}_i)}{\pi(T,\mathcal{D}_i)} = \frac{1}{\gamma(T \Rightarrow c_i)}, \text{ we get :} \\
&= \frac{\gamma(T \Rightarrow c_i)}{\gamma(T \Rightarrow c_i) + \frac{P(\bar{\mathcal{C}}_i)}{P(c_i)}}
\end{aligned}$$

Once again, ignoring class priors ratio (i.e, setting $\frac{P(\bar{C}_i)}{P(c_i)} = 1$), we obtain the definition of weighted confidence in Equation (9). Thus LHS of Equation 12 corresponds to $\rho^w(T \Rightarrow c_i)$, and by Bayes rules we assign T to class c_i if $\rho^w(T \Rightarrow c_i) > \rho^w(T \Rightarrow \bar{C}_i)$.

As described above, confidence measures strength across the entire database $\mathcal{D}$. On the other hand, likelihood measures the local tendency of the pattern to be associated with the target class; it compares the local support of the rule for the target class (c_i) with its local support for the negative class $\bar{C}_i$ (the rest of the classes). In skewed data sets, with uneven class distributions, confidence is biased in favor of the dominant class, since globally the patterns associated with this class will have higher absolute supports compared to the minority class. Likelihood and weighted confidence do not have this bias, since they ignore class priors and use local relative supports.

3. XRules: structural rule-based classification

The classification task contains two phases. The *training phase* uses a database of structures with known classes to build a classification model, in our case a set of structural classification rules, called a *rule-set*. The *testing phase* takes as input a database of structures with unknown classes, and the goal is to use the classification model to predict their classes.

3.1. Training phase

At the beginning of classification we have a database $\mathcal{D} = \bigcup_{i=1}^{k} \mathcal{D}_i$ with known classes; $\mathcal{D}_i$ is the set of structures with class c_i. Our goal is to learn a structural rule-set $\mathcal{R} = \{R^1, R^2, \ldots, R^m\}$, where each rule is of the form $R^i : T^i \overset{\pi,\delta}{\Rightarrow} c_j^i$, with $\pi \geq \pi_j^{\min}$ and with $\delta \geq \delta^{\min}$.

There are three main steps in the training phase:

- Mining frequent structural rules specific to each class, with sufficient support and strength. In this step, we find frequent structural patterns for each class and then generate those rules which satisfy a user-defined level of minimum support for a class c_i ($\pi_i^{\min}$), and a global minimum strength threshold, $\delta^{\min}$.
- Ordering the rules according to a precedence relation. Once a set of classification rules have been generated, a procedure is required to prioritize the rule set in decreasing level of precedence and to prune out unpredictive rules.
- Determining a special class called *default-class*. Since a classifier must predict a class for all possible test cases, we need to choose a default class which will be the label of a test example, if none of the rules can be used to predict a label.

3.1.1. Mining structural rules

The first step is accomplished via an efficient structural-rule mining algorithm, XMiner, which we will discuss in detail in Section 4. For the moment let us assume that XMiner can be used to find all structural rules related to any class. XMiner\ accepts as input a list of minimum support thresholds for each class, i.e., $\pi_j^{\min}, \forall j = 1, \ldots, k$. XMiner outputs a set of frequent rules for each class, $\mathcal{R}_j = \{R^1, \ldots, R^{m_j}\}$, with m_j rules, each rule having c_j as the consequent, i.e., $R^i : T^i \overset{\pi}{\Rightarrow} c_j$ and $\pi \geq \pi_j^{\min}$.

3.1.2. Pruning and ordering rules

As noted earlier, since we want only predictive rules, we need to remove any rule that lacks predictive power. For a rule $(T \Rightarrow c_i) \in \mathcal{R}_i$, if its (weighted) confidence $\rho = \rho^w = 0.5$ or if its likelihood ratio $\gamma = 1.0$, then T cannot distinguish between the class c_i and its negative class $\bar{C}_i$, and we prune such a rule from $\mathcal{R}_i$. In general, the acceptable range of values for a user-defined minimum confidence threshold is $\rho^{\min} \in (0.5, 1]$, while the acceptable range for minimum likelihood is $\gamma^{\min} \in (1, \infty]$.

The goal of precedence ordering is to derive the final combined rule-set $\mathcal{R}$ from the rule-set of each class based on a *precedence relation*, $\ll$, which imposes a total order on $\mathcal{R}$, using a method analogous to that proposed in CBA (Liu, Hsu, & Ma, 1998). Given any two rules $R^i : T^i \overset{\pi_i, \delta_i}{\Rightarrow} c^i$ and $R^j : T^j \overset{\pi_j, \delta_j}{\Rightarrow} c^j$, we say that R^i precedes R^j, denoted $R^i \ll R^j$, if:

1. The strength of R^i is greater than that of R^j, i.e., $\delta_i > \delta_j$, or
2. $\delta_i = \delta_j$, but the support of R^i is greater than that of R^j, i.e., $\pi_i > \pi_j$, or
3. $\delta_i = \delta_j$ and $\pi_i = \pi_j$, but R^i contains a smaller number of nodes than R^j, i.e., $|T^i| < |T^j|$, or
4. T^i occurs lexicographically before T^j. The lexicographic ordering of tree structures is based on a pre-order traversal of the nodes in the tree. Note that this last case will order the rules in case none of the above conditions hold, since no two rules we mine are equal.

For precedence ordering, we sort the rules across all classes using $\ll$ to derive the final ordered rule-set $\mathcal{R} = \bigcup_{i=1}^{k} \mathcal{R}_i$. In the testing phase, the ordered rules are used in various ways to predict the target class for a new structure with unknown class.

3.1.3. Determining default class

A rule $T \Rightarrow c_i$ is said to *match* a given tree S, when its antecedent, T, is a substructure of S, i.e., $T \preceq S$. A rule set $\mathcal{R}$ is said to *cover* an example tree S, if at least one rule matches S. In general, a rule set may not necessarily cover all examples (even in the training set $\mathcal{D}$). Since a classifier must provide coverage for all possible cases, we need to define a default label, denoted *default-class*, which will be chosen to be the label of a test example, if none of the rules match it.

Let $\Delta = \{S \in \mathcal{D} \mid \nexists (R^i : T^i \Rightarrow c^i_j) \in \mathcal{R} \wedge (T^i \preceq S)\}$, be the set of examples from the training set D which are not covered by the ordered rule-set $\mathcal{R}$. Let $\Delta_i = \{S \in \Delta | S.c = c_i\}$ be the set of uncovered training examples with class c_i.

A simple way to choose the default-class is to pick the majority class in Δ, i.e., default-class $= \arg\max_{c_i}\{|\Delta_i|\}$. If $\Delta = \emptyset$, then pick default-class to be the majority class in $\mathcal{D}$. The problem with this method is that it does not take into consideration the real cost of the classes (it uses the proportional cost model by default). The approach we adopt is to choose the class that maximizes the cost-sensitive accuracy of the resulting rule-based classifier. Let $\eta(\mathcal{D})$ denote the number of correct predictions for data set $\mathcal{D}$ using our rule-set R. If $\Delta \neq \emptyset$, then the default class is given as *default-class* $= \arg\max_{c_i}\{\frac{w_i|\Delta_i|}{|\mathcal{D}_i|}\}$ (see lemma below). If $\Delta = \emptyset$, then the default class is the one with maximum weight w_i (obtained by setting $\Delta_i = \mathcal{D}_i$). It is clear that such a technique is superior from the perspective of a cost-sensitive approach.

Lemma 3.1. *The cost-sensitive accuracy is maximized for default-class* $= \arg\max_{c_j}\{\frac{w_j|\Delta_j|}{|\mathcal{D}_j|}\}$.

Proof: Assume $\Delta \neq \emptyset$. The base accuracy for a given class c_i in $\mathcal{D}_i$ is given as $\alpha(\mathcal{R}, \mathcal{D}_i) = \frac{\eta(\mathcal{D}_i)}{|\mathcal{D}_i|}$. By Equation (3) the overall base cost-sensitive accuracy is given as

$$\alpha^{cs}_{old}(\mathcal{R}, \mathcal{D}) = \sum_{i \epsilon [1,k]} \frac{w_i \eta(\mathcal{D}_i)}{|\mathcal{D}_i|}$$

Assume that we pick class c_j as the default class. This affects only the accuracy of class c_j due to the addition of correct predictions for class c_j in Δ, whereas the accuracy of all $c_i \neq c_j$ remains unchanged. Therefore, we have $\alpha(\mathcal{R}, \mathcal{D}_j) = \frac{\eta(\mathcal{D}_j)+|\Delta_j|}{|\mathcal{D}_j|}$. The new overall accuracy is then given by

$$\alpha^{cs}(\mathcal{R}, \mathcal{D}) = \frac{w_j(\eta(\mathcal{D}_j) + |\Delta_j|)}{|\mathcal{D}_j|} + \sum_{i \in [1,k], i \neq j} \frac{w_i \eta(\mathcal{D}_i)}{|\mathcal{D}_i|}$$

After simplifying, we get $\alpha^{cs}(\mathcal{R}, \mathcal{D}) = \frac{w_j|\Delta_j|}{|\mathcal{D}_j|} + \alpha^{cs}_{old}(\mathcal{R}, \mathcal{D})$. Since $\alpha^{cs}_{old}(\mathcal{R}, \mathcal{D})$ remains the same no matter which class we pick as default, the overall accuracy is maximized for the class yielding the maximum value of $\frac{w_j|\Delta_j|}{|\mathcal{D}_j|}$.

If $\Delta = \emptyset$, we set $\Delta_j = D_j$. So the class yielding maximum accuracy is the one with maximum w_j. □

Corollary 3.1. *For the proportional cost model, the accuracy is maximized if the default class is the majority class in Δ (or in $\mathcal{D}$ if $\Delta = \emptyset$).*

Proof: Assume $\Delta \neq \emptyset$. Substituting $w_i = \frac{D_i|}{|D|}$ in $\frac{w_i|\Delta_i|}{|D_i|}$, the term to be maximized, we get $\frac{|\Delta_i|}{|D|}$. This is maximized for the class with the maximum value of $|\Delta_i|$, i.e., the majority class in Δ. If $\Delta = \emptyset$, then setting $\Delta_i = \mathcal{D}_i$ gives the desired result. □

As described above we prune all unpredictive rules having $\rho^{\min} = 0.5$ or $\gamma^{\min} = 1.0$. Also recall that when building a model we always compare the confidence of the rule on class c_i versus its negative class $\bar{\mathcal{C}}_i$. In some cases, the rules may be poorly related to an example. This happens when the average (weighted) confidence or likelihood of the rules which are matched by a given example are *close* to 0.5 or 1.0, respectively, for a given class c_i. This means that the rule is equally predictive of c_i as well as $\bar{\mathcal{C}}_i$, and thus not suitable for classification. If the user sets $\rho^{\min} > 0.5$ or $\gamma^{\min} > 1.0$ any example with matching rules having average (weighted) confidence in the range $[1 - \rho^{\min}, \rho^{\min}]$ or having average likelihood is in the range $[1/\gamma^{\min}, \gamma^{\min}]$, is assumed to be an ambiguous case, which cannot be accurately classified. For example, suppose the use sets $\rho^{\min} = 0.55$, then any rule with confidence in the range $[1 - 0.55, 0.55] = [0.45, 0.55]$ will be discarded. Such ambiguous cases are added to the default set Δ (essentially treating them as examples having no matching rule in $\mathcal{R}$), which is used for the final determination of the default-class as described above.

3.2. Testing phase

At the end of training, our classification model is complete. It consists of an ordered collection of predictive rules $\mathcal{R}$, and a default-class. The testing phase takes as input the classification model, and a data set $\mathcal{D}'$ of examples with unknown classes (or for evaluation purposes,

examples with known labels, which have not been used for training). The goal of the testing phase is to predict the class for each test example. There are two main steps in testing:

- Rule Retrieval: Find all matching rules for a test example.
- Class Prediction: Combine the statistics from each matching rule to predict the most likely class for the test example.

The rule retrieval step is simple; for each test example S in the database $\mathcal{D}'$, we find the set of all matching rules, called the *matching rule-set*, $\mathcal{R}(S) = \{R^i : T^i \stackrel{\delta^i}{\Rightarrow} c^i | T^i \preceq S\}$.

For predicting the class of $S \in \mathcal{D}'$, we can use several different approaches for combining the statistics of the matching rule-set $\mathcal{R}(S)$. There are two cases to be considered: First, if $\mathcal{R}(S) = \emptyset$ (i.e., there are no matching rules), the class is predicted to be the default class, i.e., $S.c = \textit{default-class}$. On the other hand, if $\mathcal{R}(S) \neq \emptyset$, then let $|\mathcal{R}(S)| = r$. Also let $\mathcal{R}_i(S)$ denote the matching rules in $\mathcal{R}(S)$ with class c_i as the consequent, and let $|\mathcal{R}_i(S)| = r_i$. Each rule in$\mathcal{R}_i(S)$ is of the form $T^j \stackrel{\delta^j}{\Rightarrow} c_i^j$, with $\delta^j \geq \delta^{\min}$. Any matching rule $T^k \in \mathcal{R}(S) - \mathcal{R}_i(S)$ is more predictive of a class other than c_i. However, XMiner finds the support of T^k for all classes (see Section 4), so we can compute the strength of T^k for the negative class $\bar{\mathcal{C}}_i$ ($T^k \stackrel{\delta^n}{\Rightarrow} \bar{\mathcal{C}}_i$). The strength of T^k for c_i, i.e., the rule $T^k \stackrel{\delta^k}{\Rightarrow} c_i$ is given as $\delta^k = 1 - \delta^n$ if $\delta \equiv \rho$ (or $\delta \equiv \rho^w$), and as $\delta^k = 1/\delta^n$ if $\delta \equiv \gamma$ (by Eqations (5), (7), (8)). Thus, for each class c_i we can find the strength of each structural rule for that class. A matching rule with $\rho > 0.5$ or $\gamma > 1.0$ corresponds to a rule with positive predictive power for c_i, while a matching rule with $\rho < 0.5$ or $\gamma < 1.0$ is more predictive of the negative class, and thus has negative predictive power for c_i.

There are several possible methods for combining evidence:

- *Average Strength*: Compute the average rule strength for each class c_i given as $\delta_i^\mu = \frac{\sum_{j=1}^{r^i} \delta^j}{r^i}$. If $\delta_i^\mu \geq \delta^{\min}$ then we classify S as having class c_i. If δ_i^μ has default $\delta^{\min}$ values (0.5 for $\rho^{\min}$ and 1.0 for $\gamma^{\min}$) for all classes, it means that the test instance cannot be easily predicted using the rules, and the class is assigned as the default class. The approach can be generalized to the case where δ_i^μ is ambiguous, i.e., when $\rho_i^\mu \in [1 - \rho^{\min}, \rho^{\min}]$ for (weighted) confidence, and when $\gamma_i^\mu \in [1/\gamma^{\min}, \gamma^{\min}]$ for likelihood. In such a case, we assign $S.c$ to be the default class.
- *Best Rule*: Find the first rule that matches S, i.e., the first rule in $\mathcal{R}(S)$. Since the rule set is ordered according to precedence $\ll$, the first rule $T \Rightarrow c_i \in \mathcal{R}(S)$ is the best or most predictive (by nature of the total order $\ll$, a matching rule after this one will either have less strength, or less support or will be more specific). We thus predict $S.c = c_i$.
- *Best K-Rules* Apply average strength method for the first K rules in $\mathcal{R}(S)$. This is a simple generalization of the case discussed above.

In our experiments we used the average strength method for combining evidence, since it gave us the best results. It is easy to see that if only the average strength method is used, then rule ordering is not required while mining the rules. We note that for average strength-based method, if the classification behavior of a test instance is ambiguous (equal to or close to default $\delta^{\min}$ values), the classifier can also output this fact as useful information to the end user. While classifiers traditionally strive for 100% coverage (i.e., they predict a label of each test case), a practical application may often benefit greatly from knowledge of the fact that certain test instances are harder to classify than others. This results in lower coverage, but a better understanding of the overall classification process.

4. XMiner

In order to determine the set of rules, XRULES first needs to mine the frequent subtrees in the data. We chose TreeMiner as a basis for XMINER, since it is a complete method, and it uses a novel vertical representation for fast subtree support counting.

Given a dataset D with k classes, and thus k partitions $\mathcal{D}_i$, one approach to mining structural rules would be to mine each $\mathcal{D}_i$ separately using TreeMiner, and then to combine the results. There are two problems with this approach: (1) XRULES needs to know the support of a tree T in each class, but T may be frequent in one class $\mathcal{D}_i$, but not in another $\mathcal{D}_j$. (2) We would need one extra scan to count such missing class supports, thus this approach is inefficient. For example, let's assume that rule R is frequent only for class c_2. While mining D_1, R will be found to be infrequent and will be discarded. However, when we mine D_2, we find R to be frequent. To compute the strength measures, we need to know R's support in all partitions; to obtain the support requires additional scans of those missed partitions. As opposed to this strategy XMINER extends TreeMiner to simultaneously mine all frequent trees related to some class, and also incorporates multiple minimum support criteria, one per class. This ensures that any tree generated is suitable for classification purposes. Like TreeMiner, XMINER utilizes the *vertical* tree representation for fast support counting and uses a depth-first (DFS) pattern search.

4.1. Node number, scope, and match label

Let $T = (V_t, E_t)$ be a tree. We assume that vertex $x \in V_t$ is synonymous with (or numbered according to) its position in the depth-first (pre-order) traversal of the tree T. We use the notation n_i to refer to the ith node according to this numbering scheme, where $i = 0 \cdots |T| - 1$ (for example, the root is vertex n_0). Let $T(n_l)$ denote the subtree rooted at node n_l, and let n_r be the rightmost leaf (or highest numbered descendant) under n_l. Then the *scope* of x is given as the interval $[l, r]$ (or $[n_l, n_r]$). Intuitively, a node's scope demarcates the range of vertices under it. Figure 2 shows a database of 3 trees, with 2 classes; for each tree it shows the node number n_i, node scope $[l, u]$, and node label (inside the circle).

Let D denote a database of trees (i.e., a forest), and let subtree $T \preceq S$ for some $S \in \mathcal{D}$. Each occurrence of T in S can be identified by its *match label*, which is given as the set of matching positions (in S) for nodes in T. More formally, let $\{t_1, t_2, \ldots, t_n\}$ be the nodes in T, with $|T| = n$, and let $\{s_1, s_2, \ldots, s_m\}$ be the nodes in S, with $|S| = m$. Then T has a match label $\{t_{i_1}, t_{i_2}, \ldots, t_{i_n}\}$, if and only if: 1) $L(t_k) = L(s_{i_k})$ for all $k = 1, \ldots, n$ (where $L(n)$ is the label for node n), and 2) branch $b(t_j, t_k) \in T$ iff s_{i_j} is an ancestor of s_{i_k} in S. Condition 1) indicates that all node labels in T have a match in S, while 2) indicates that the tree topology of the matching nodes in S is the same as T. A match label is unique for each occurrence of T in S.

4.2. Prefix group and scope lists

Let T be a k-subtree of some tree S. Let x_k refer to the last node of T. We say that two k-subtrees T_1, T_2 are in a *prefix equivalence group* iff they share a common prefix up to the $(k-1)$th node. Let P be a prefix subtree of size $k-1$. We use the notation $[P]_{k-1}$ to refer to its group, which contains all the last items (k-th node) of trees that share P as their prefix.

We use the notation $\mathcal{L}(T)$ to refer to the *scope-list* of T. Each element of the scope-list is a triple (t, m, s), where t is a tree id (tid) in which T occurs, s is the scope of x_k, and m is a

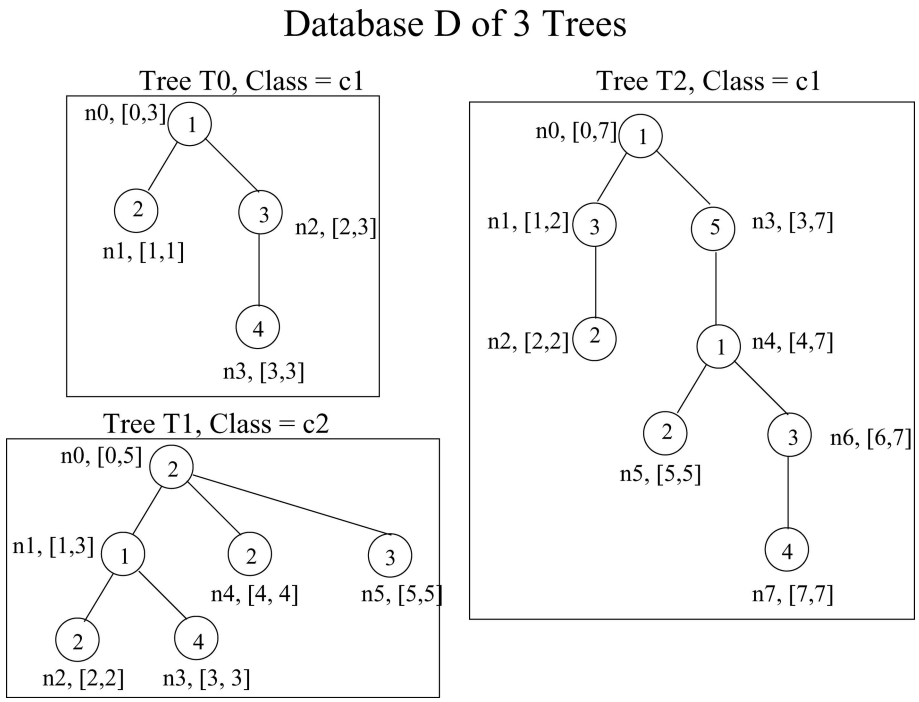


Fig. 2 A database of trees for classification.

match label for immediate prefix T. Since a subtree can occur multiple times in a tree, each tid can be associated with multiple scopes and match labels.

The initial scope-lists are created for single items i that occur in a tree T. Let $[l, u]$ be the scope of a node with label i. Since the match label of item i is simply l we omit storing m when dealing with the scope-lists of single items. We will show below how to compute pattern frequency via joins on scope-lists. Figure 3 shows the scope lists for the frequent single items (the minimum support is 100% for both classes), for the trees shown in Figure 2. Consider item 1; since it occurs at node position 0 with scope $[0, 3]$ in tree T_0, we add $(0, [0, 3])$ to its scope list. Item 1 also occurs in T_1 at position n_1 with scope $[1, 3]$, so we add $(1, [1, 3])$ to $\mathcal{L}(1)$. Finally, 1 occurs with scope $[0, 7]$ and $[4, 7]$ in tree T_2, so we add $(2, [0, 7])$ and $(2, [4, 7])$ to its scope-list. In a similar manner, the scope lists for other items are created. Item 5 is not shown, since it is not frequent for any class; it has support 50% in class c_1.

4.3. Tree mining

Figure 4 shows the high level structure of XMiner. The main steps include the computation of the frequent items and the enumeration of all other frequent subtrees via DFS search within each group. XMiner also maintains a global *class index* showing the class for each tree in the database. This index is used to quickly update the per class support for a candidate tree to check if it is frequent in any class. Figure 3 shows the class index for the example database.

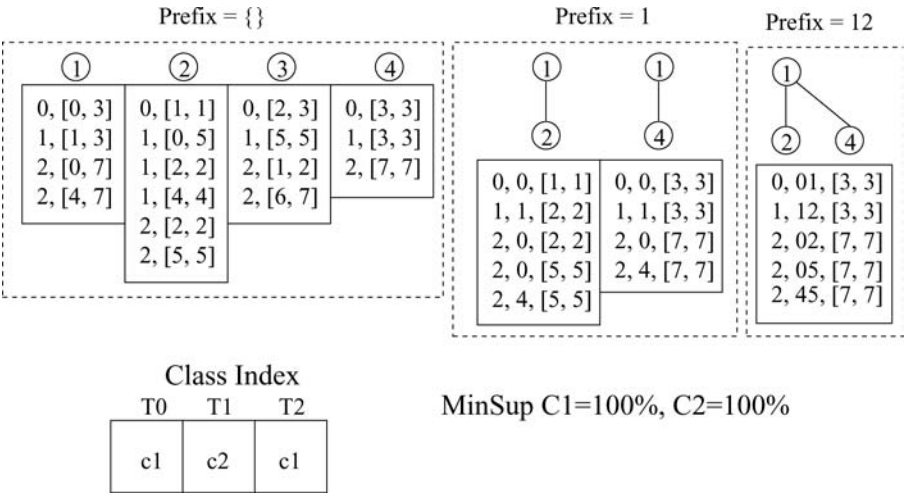


Fig. 3 Tree mining example.

XMINER $(\mathcal{D}, \pi_i^{\min} \forall i = 1 \cdots k)$:
$[P]_0 = \{$ frequent 1-subtrees for any class$\}$;
Enumerate-Xrules$([P]_0)$;

Enumerate-Xrules$([P])$**:**
for all elements $x \in [P]$ **do**
$[P_x] = \emptyset$;
for all elements $y \in [P]$ **do**
$\mathbf{R} = x \otimes y$;
$\mathcal{L}(\mathbf{R}) = \mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)$;
if for any $R \in \mathbf{R}$, R is frequent for at least one class
then $[P_x] = [P_x] \cup \{R\}$;
Enumerate-Xrules$([P_x])$;

Fig. 4 XMINER: tree mining for classification.

The input to Enumerate-Xrules is a set of elements of a group $[P]$, along with their scope-lists. Frequent subtrees are generated by joining the scope-lists of all pairs of elements (including self-joins). Before joining the scope-lists a pruning step can be inserted to ensure that all subtrees of the resulting tree are frequent. If this is true, then we can go ahead with the scope-list join, otherwise we can avoid the join. The collection of candidate subtrees is obtained by extending each tree in a group by adding one more item (the last item) from another tree in the same prefix group. We use $\mathbf{R}$ to denote the possible candidate subtrees that may result from extending tree with last node x, with the tree with last item y (denoted $x \otimes y$), and we use $\mathcal{L}(\mathbf{R})$ to denote their respective scope-lists.

The subtrees found to be frequent at the current level form the elements of groups for the next level. This recursive process is repeated until all frequent subtrees have been enumerated. In terms of memory management it is easy to see that we need memory to store intermediate scope-lists for two groups, i.e., the current group $[P]$, and a new candidate group $[P_x]$.

4.4. Scope-list joins ($\mathcal{L}(x) \cap_{\otimes} \mathcal{L}(y)$)

We now describe how we perform the scope-list joins for any two subtrees in a group $[P]$. Let $s_z = [l_z, u_z]$ denote the scope for a node z. We say the s_x is strictly less than s_y, denoted $s_x < s_y$, if and only if $u_x < l_y$. We say that s_x contains s_y, denoted $s_x \supseteq s_y$, if and only if $l_x \leq l_y$ and $u_x \geq u_y$. When we join the last elements $x \otimes y$ in a group, there can be at most two possible outcomes, i.e., we either add y as a child of x or as a sibling of x to the class $[P_x]$.

To check if the subtree, obtained when y is added as a child of x, occurs in an input tree T with tid t, it is sufficient to search if there exists triples $(t_y, m_y, s_y) \in \mathcal{L}(y)$ and $(t_x, m_x, s_x) \in \mathcal{L}(x)$, such that: i) $t_y = t_x = t$, ii) $s_y \subseteq s_x$, and iii) $m_y = m_x$.

In other words, we check 1) if x and y both occur in the same tree T with tid t, 2) if y is within the scope of x, and 3) that x and y are both extensions of the same prefix subtree, $P \preceq T$, whose match label is $m_x = m_y$. If the three conditions are satisfied, we add the triple $(t_y, \{m_y \cup l_x\}, s_y)$ to the scope-list of y in $[P_x]$. We refer to this case as an *in-scope* test.

The second pattern checks what happens when y is added as a (embedded) sibling of x. This happens when both x and y are descendants of node at position j in the prefix P, and the scope of x is strictly less than the scope of y. To check if y occurs as an embedded sibling in T with tid t, we need to check if there exists triples $(t_y, m_y, s_y) \in \mathcal{L}(y)$ and $(t_x, m_x, s_x) \in \mathcal{L}(x)$, such that: i) $t_y = t_x = t$, ii) $s_x < s_y$, and iii) $m_y = m_x$.

If the three conditions are satisfied, we add the triple $(t_y, \{m_y \cup l_x\}, s_y)$ to the scope-list of y in $[P_x]$. We refer to this case as an *out-scope* test.

Figure 3 shows the process of scope-list joins for both in-scope and out-scope tests. An example of in-scope test is when we extend item 1 by adding item 2 as a child, i.e., the tree in prefix group [1], with the branch (1, 2). Let s_i denote a scope for item i. For tree T_0 we find that $s_2 = [1, 1] \subset s_1 = [0, 3]$. Thus we add the triple (0, 0, [1, 1]) to the new scope list. In like manner, we test the other occurrences of item 2 under item 1 in trees T_1 and T_2. Note that for T_2 there are three instances of the candidate pattern: $s_2 = [2, 2] \subset s_1 = [0, 7]$, $s_2 = [5, 5] \subset s_1 = [0, 7]$, and $s_2 = [5, 5] \subset s_1 = [4, 7]$. To check if the new candidate is frequent, we derive a per class count using the class index. Since the new tree appears in tids 0,1, and 2 (we count only once per tid), using the class index we find that it occurs in classes c_1, c_2, c_1 respectively. Its support for class c_1 is 2 and for class c_2 is 1. It is thus 100% frequent locally in both classes. Figure 3 also shows an example of an out-scope test when we join branches (1, 2) and (1, 4) to obtain the tree shown on the right with prefix group [12].

Since XMiner keeps track of all embeddings, and is a complete method, its computational complexity is proportional to the number of patterns mined, as well as the maximum number of embeddings. Both of these terms can be exponential in the number of labels and/or the maximum size of a tree in the database. On the other hand, the method is more scalable, typically linear, with respect to the number of trees.

5. Empirical results

We compared our XRules structural classification approach for XML documents against an IR classifier, as well as the CBA classifier. For the IR classifier (IRC) centroids for each class

Fig. 5 URLs and their Ids.

```
1 http://www.cs.xxx.edu/
6 http://www.cs.xxx.edu/courses/
8 http://www.cs.xxx.edu/current-events/
12 http://www.cs.xxx.edu/People/
14 http://www.cs.xxx.edu/research/
...
```

were constructed using a clustering process (Aggarwal, Gates, & Yu, 1999). Then, a nearest neighbor classifier was implemented on these sets of clusters. The CBA implementation was provided to us by its authors (Liu, Hsu, & Ma, 1998). While there have been some recent improvements over CBA provided by CAEP (Dong *et al.,* 1999) and CMAR (Li, Han, & Pei, 2001), we use CBA as the representative method. In any case neither CAEP nor CMAR can handle structural patterns; they use sets. We also compare our approach with a Support Vector Machine (SVM) classifier, svm-light (Joachims 2002) (http://svmlight.joachims.org/), since SVMs have proven to be very effective in many domains; we wanted to see how they perform for structural classification. All experiments were done on a 3.2 GHz Pentium4 machine with 2GB memory and a 200 GB, 7200 rpms IDE disk.

5.1. Data sets

We evaluate our approach on both real and synthetic classification data sets. The advantage of using synthetic data sets was the additional flexibility in studying the effects of different kinds of embedded patterns and database size. On the other hand, the real data sets help to validate the approach in a practical setting.

5.1.1. Real datasets

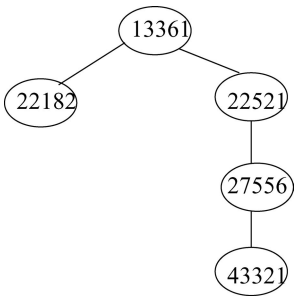
We use the Log Markup Language (LOGML) (Punin & Krishnamoorthy, 2000), to describe log reports at the CS department website. LOGML provides a XML vocabulary to structurally express the contents of the log file information in a compact manner. Each user session is expressed in LOGML as a graph, and includes both structure and content. LOGML documents have three parts: a web graph induced by the source-target page pairs in the raw logs, a summary of statistics (such as top hosts, domains, keywords, number of bytes accessed, etc.), and a list of user-sessions (subgraphs of the web graph) extracted from the logs. We used our CS department web site to populate a web graph with the help of a web crawler. The raw logs are processed by the LOGML generator and turned into one LOGML document per user-session. We use the web graph to obtain the page URLs and their node identifiers. For example, the snippet in Figure 5 shows the (node id, URL) pairs (out of a total of 56623 nodes) we extracted from the CS web graph.

For structural mining we make use of user sessions within the LOGML document. User sessions are expressed as subgraphs of the web graph, and contain complete history of the user clicks. Each user session has a name (IP or host name), a list of pages accessed along with their content, a list of edges giving source and target node pairs, and the attribute utime showing when the link was traversed. An example XML document snippet is shown in Figure 6, and the resulting user browsing graph shown in Figure 7. In general the browsing graph need not be a tree; we convert a graph into a tree by extracting only the forward edges starting from the root, avoiding cycles or multiple parents. This might result in loss of information; in general, it would be preferable to extend our classification approach using

```
<?xml version="1.0"?>
<!DOCTYPE graph SYSTEM "xgmml.dtd">
<graph name="yen.cc.strath.ac.uk" ... >
<node id="22521" label="www.cs.xxx.edu/poetry.html"
<att name="content"
value="NAZIM HIKMET His Life, His Work, Comments
Selected Poetry of Nazim Hikmet
Comtemporary Turkish Writers"/>
....
<edge source="13361" target="22182">
<att name="utime" value="19/Oct/2001:09:36:36"/>
</edge>
<edge source="13361" target="22521">
<att name="utime" value="19/Oct/2001:09:36:36"/>
</edge>
<edge source="22521" target="27556">
<att name="utime" value="19/Oct/2001:09:45:47"/>
</edge>
<edge source="27556" target="43321">
<att name="utime" value="19/Oct/2001:09:46:19"/>
</edge>
</graph>
```

Fig. 6 XML document snippet.

Fig. 7 User browsing tree.



graph mining (Yan & Han, 2002) instead of trees. However, it should be noted that graph mining is considerably more expensive than tree mining.

The real CSLOG data set spans 3 weeks worth of such XML user-sessions. To convert this into a classification data set we chose to categorize each user-session into one of two class labels: `edu` corresponds to users from an "edu" domain, (also includes "ac" academic domain), while `other` class corresponds to all users visiting the CS department from any other domain. For instance, the class of the XML document snippet shown above with host-name `yen.cc.strath.ac.uk` is `edu`. The goal of classification is to find out if we can separate users who come from edu versus other domains from their browsing behavior within the CS web site. Note that the three classifiers used in our experiments use different kinds of information from the same underlying XML user-sessions. XRULES uses only the subtrees obtained from the XML sessions, as shown in Figure 7. The original URLs or the

Table 1 Characteristics of datasets.

DB	# Sessions	edu	Other	%edu	%Other
CSLOG1	8074	1962	6112	24.3	75.7
CSLOG2	7407	1686	5721	22.8	77.2
CSLOG12	13934	2969	10965	21.3	78.7
CSLOG3	7628	1798	5830	23.6	76.4
DB	total	c_1	c_2	$\%c_1$	$\%c_2$
DS1.train	91288	41288	50000	45.2	54.8
DS2.train	67893	17893	50000	26.4	73.6
DS3.train	100000	50000	50000	50.0	50.0
DS4.train	75037	35298	39739	47.0	53.0
DS1.test	88493	38493	50000	43.5	56.5
DS2.test	72510	22510	50000	31.0	69.0
DS3.test	100000	50000	50000	50.0	50.0
DS4.test	74880	37977	36903	50.7	49.3

text is not used at all. CBA uses a flat representation of the trees, namely the node set; for our example, we obtain the set {13361, 22182, 22521, 27556, 43321}. The SVM classifier is also based on the flat node set; each node represents a binary feature, and only the presence of nodes is kept in a feature vector. For our example, SVM will use the binary feature vector "13361:1 22182:1 22521:1 27556:1 43321:1"; other features are assumed to be "0". Finally, IRC uses the entire snippet shown in Figure 6 (including the tags, text, and so on).

As shown in Table 1, we 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`. Our goal is to minimize the cost of classification inaccuracy based on the various models. We use the notation CSLOG$x - y$ to denote that we trained on CSLOGx and tested on CSLOGy. For example, CSLOG1-2 means that we learned a model from CSLOG1 and tested how well we could predict CSLOG2.

5.1.2. Synthetic datasets

We constructed a synthetic data generation program simulating website browsing behavior. Each user's browsing trail is modeled as a tree, and we group users into two classes. We also ensure that one of the classes depends on the presence of certain subtrees. Data generation happens via three steps: (i) master tree generation, (ii) subtree generation, and (iii) classification dataset generation.

Master tree generation: The program first constructs a master website browsing tree, $\mathcal{W}$, based on parameters supplied by the user. These parameters include the maximum fanout F of a node, the maximum depth D of the tree, the total number of nodes M in the tree, and the number of node labels L. We allow multiple nodes in the master tree to have the same label. The master tree is generated using the following recursive process. At a given node we first pick a label by sampling uniformly from the range 1 through L. For each node we also decide how many children to generate by sampling uniformly at random from the range 1 to F. For each node in master tree $\mathcal{W}$, we assign (random) probabilities of following

its children nodes, including the option of backtracking to its parent, such that sum of all the probabilities is normalized to be 1. The probability associated with a branch $b = (x, y)$, indicates how likely is a visitor at x to follow the link to y. As long as tree depth is less than or equal to maximum depth D this process continues recursively. The procedure stops once M nodes have been generated.

Subtree generation: Using the master tree, one can generate a subtree $T_i \preceq \mathcal{W}$ by randomly picking a subtree of $\mathcal{W}$ as the root of T_i and then recursively picking children of the current node according to the probability of following that link. Assume that $\mathcal{W}$ has l levels (*i.e.*, $1 \cdots l$). Let N_j denote all the nodes at level j in tree $\mathcal{W}$, let $w_j = 1/j$ denote the weight associated with level j, and let $P_j = \frac{w_j}{\sum_{i=1}^{l} w_i}$ denote the probability of choosing a node at level j. The root node of W_i is chosen from $\mathcal{W}$ by i) selecting a random level j with probability P_j, and ii) selecting a random node r from N_j. We add other nodes to T_i via a recursive process starting at r: Let x be the current node in T_i (x also belongs to $\mathcal{W}$), let x_j denote the jth child of x ($j = 1 \ldots k$ if x has k children), let $P(x_j)$ denote the probability of selecting child j, and let $P(x_0)$ denote the probability of backtracking to the parent of x ($P(x_i), \forall i = 0 \cdots k$ were generated while creating the master tree). We add child x_i to T according to its probability $P(x_i)$. If a child has already been visited, we select one of the other unvisited children, or we backtrack. The process stops if we have visited all children of the root node r or if we try to backtrack to the parent of x. Other strategies are also possible.

Classification dataset generation: To create a classification data set we group users into two classes, c_1 and c_2. First we generate a small pool of signature trees for class c_1, denoted T_p, using the subtree generation method from above. Second, we generate a larger collection of trees, denoted T_D. T_D will be split into two disjoint parts: a training set T'_D, and a test set T''_D. In the same manner T_p is partitioned into a training (T'_p)and testing (T''_p) pool, with one exception: T'_p and T''_p need not span T_p and they may have overlap (controlled by a parameter). Both the training and testing datasets are split into two classes as follows: For each tree in the training set, $T \in T'_D$, we check if there exists a tree $S \in T'_p$ such that $S \preceq T$. If so, T is assigned to class c_1, else it is assigned to class c_2. Similarly a tree in T''_D is assigned to class c_1 if there is a matching subtree in T''_p.

To control the effects of structure in the classification process, i.e., to emphasize structure based rules to distinguish class c_1 from c_2, a given fraction f_c, called *confusion ratio*, of trees that belong to one class (c_1) are also added to other class (c_2), after flattening out (i.e., we add the vertex set of those trees to the other class). This is called one-way addition. If we also allow members of c_2 to be added to c_1, it is called a two-way addition. This process would confuse a method based purely on set mining, since for that f_c fraction of trees a set mining method will not be able to discriminate between the two classes. By varying the confusion ratio, we can control the degree to which the datasets depend on structural rule mining.

The different synthetic data sets generated are shown in Table 1. For the DSx data sets, we trained on DSx-train and tested on DSx-test. The master tree $\mathcal{W}$ used the values $D = 10$, $F = 10$, $M = 100$, $L = 10$. We next generated $|T_D| = 100,000$ trees for the database and $|T_p| = 1000$ trees for the pool. T_D was split into training and test sets by using a $50 - 50$ split. For DS1, the training and testing pool were both of size 20, with half the trees common to both. We set $f_c = 1.0$, with one-way addition from c_1 to c_2. For DS2, the training and testing pool were identical (of size 10), and $f_c = 1.0$ from c_1 to c_2. DS3 is the same as DS2, with two-way confusion. Finally DS4 is same as DS2, but with two-way addition only half the time ($f_c = 0.5$).

Table 2 Number of rules and time.

DB	Sup	Rules	Trees	Train time (s)		Testing time (s)
				XM	Total	
CSLOG1-2	0.3%, 0.25%	28911	12231	2.1	196.0	176.3
CSLOG2-3	0.3%	19098	2755	1.5	115.3	113.9
CSLOG12-3	0.35%	29028	20745	2.9	369.9	192.6
CSLOG3-1	0.2%	31661	5056	1.9	299.1	252.0
DS1	0.3%	883	307	3.0	66.6	67.8
DS2	0.3%	1589	863	2.4	88.4	105.0
DS3	0.3%	739	234	3.1	60.8	59.2
DS4	0.3%	900	399	2.4	65.6	57.0

5.2. Example of mined patterns

Before we consider the performance of various algorithms on the different datasets, we give an example of patterns mined by XMiner from the CSLOG1 dataset using the parameters shown in Table 2. Out of the 29811 rules found, we show some top ranked structural rules for classes `edu` and `other` in Figure 8.

We find that both rules R_1 and R_2 pertain to class `edu`, and are from user sessions (most likely students) interested in the CS2 class taught by Profs. Kettnaker and Stewart at RPI. On the other hand the rules R_3 and R_4 apply to class `other`. R_3 shows that many users who come from outside edu domains are interested in the Madonna lyrics pages maintained by Kenny Zalewski; likewise R_4 characterizes users interested in the Turkish poetry links maintained by Prof. Adali at RPI. These rules are quite intuitive to understand. Further analysis of the mined rules revealed that most of the `edu` users are interested in course pages and other aspects of academics (undergraduate and graduate), where as most of the `other` users are interested in non-academic resource pages maintained by CS students and faculty, such as the Madonna (Zalewski), Turkish poetry (Adali), and Hockey (Zalewski) pages.

5.3. Comparative classification results

The IRC approach uses the actual text of the data in order to perform the classification. Therefore, it uses a greater amount of information than a purely structural classifier like XRules. IRC uses both the node content and edge information from the user-sessions. In contrast, XRules uses only the structure (tree-format) for the classification process. CBA uses the associations among different nodes visited in a session in order to perform the classification, whereas SVM treats each node as a binary feature for classification; thus each record is a (sparse) binary feature vector.

5.3.1. Weighted ROC analysis

Comparative classification results are shown in Table 3 and Figure 9. Table 3 shows the weighted accuracy results for the classifiers on different data sets. For the set of parameters used to run XRules and the number of rules mined see Table 2. Table 3 shows the accuracy for all three cost models. The best accuracy is highlighted in bold. The accuracy for a random classifier is also shown, which is computed as follows: Let D be a test dataset, let c_i be a class (out of k possible classes), and let $\mathcal{D}_i$ be the partition of D with class c_i. Then $f_i = \frac{|\mathcal{D}_i|}{|D|}$ denotes the fraction of test cases having class c_i. The *Random* classifier assigns a class

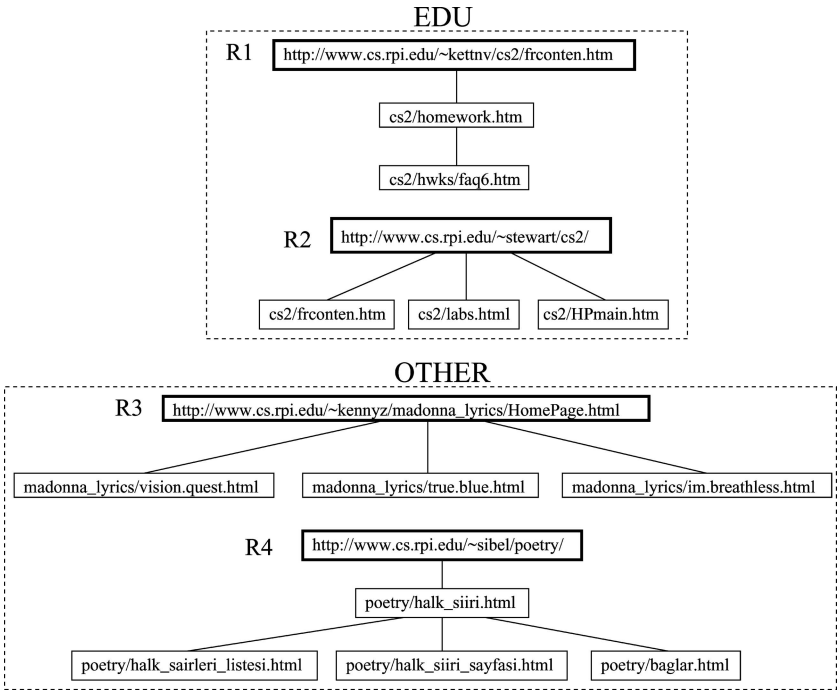


Fig. 8 Top ranking structured rules for CSLOGS1.

at random to each test case, thus we expect it to have class-specific accuracy $\alpha(\mathcal{D}_i) = f_i$ on class c_i. The weighted accuracy of the Random classifier for different cost models can then be obtained as follows: For the proportional cost model, $w_i = f_i$, thus $\alpha^{cs}(\mathcal{D}) = \sum_{i=1}^{k} w_i \times f_i = \sum_{i=1}^{k} (f_i)^2$. When $k = 2$, we get $\alpha^{cs}(\mathcal{D}) = (f_1)^2 + (f_2)^2$. For the equal cost model, $w_i = 1/k$, thus $\alpha^{cs}(\mathcal{D}) = \sum_{i=1}^{k} w_i \times f_i = \frac{1}{k} \sum_{i=1}^{k} f_i = \frac{1}{k}$. When $k = 2$, we get $\alpha^{cs}(\mathcal{D}) = 0.5$. For the inverse cost model, $w_i = \frac{1/f_i}{\sum_{j=1}^{k} 1/f_j}$ thus $\alpha^{cs}(\mathcal{D}) = \sum_{i=1}^{k} w_i \times f_i = \sum_{i=1}^{k} \frac{1/f_i}{\sum_{j=1}^{k} 1/f_j} \times f_i = \frac{k}{\sum_{j=1}^{k} 1/f_j}$. When $k = 2$, we get $\alpha^{cs}(\mathcal{D}) = \frac{2}{\frac{1}{f_1} + \frac{1}{f_2}} = \frac{2}{\frac{f_1 + f_2}{f_1 f_2}} = 2 f_1 f_2$.

Figure 9 plots each classifier as a point in the Weighted Receiver Operating Characteristic (ROC) space. ROC space is commonly used to illustrate the comparative classification performance for two-class problems. Let P be the "positive" class and N the "negative", and let $f_P = \frac{|\mathcal{D}_P|}{|\mathcal{D}|}$ and $f_N = \frac{|\mathcal{D}_N|}{|\mathcal{D}|}$ be the fraction of positive and negative classes in D, respectively. There are four possible outcomes during testing: a) Number of positive examples predicted correctly as positive (TP), b) Number of negative examples predicted incorrectly as positive (FP), c) Number of negative examples predicted correctly as negative (TN), and d) Number of positive examples predicted incorrectly as negative (FN). Note that $TP + FN = |\mathcal{D}_P|$ is the number of test examples with class P, and $TN + FP = |\mathcal{D}_N|$ is the number of test examples with class N, whereas $TP + FP$ is the number of examples predicted as positives, and $TN + FN$ is the number of examples predicted as negatives. The true positive rate (TPR) of a classifier is given as $\frac{TP}{|\mathcal{D}_P|}$, and its false positive rate (FPR) is given as $\frac{TN}{|\mathcal{D}_N|}$.

Traditional ROC analysis plots the TPR versus the FPR. We extend the traditional ROC analysis to *Weighted ROC Analysis (WROC)* as follows. We can the define the Weighted True Rate (WTR) for a classifier, which is given as the weighted average of the true positive

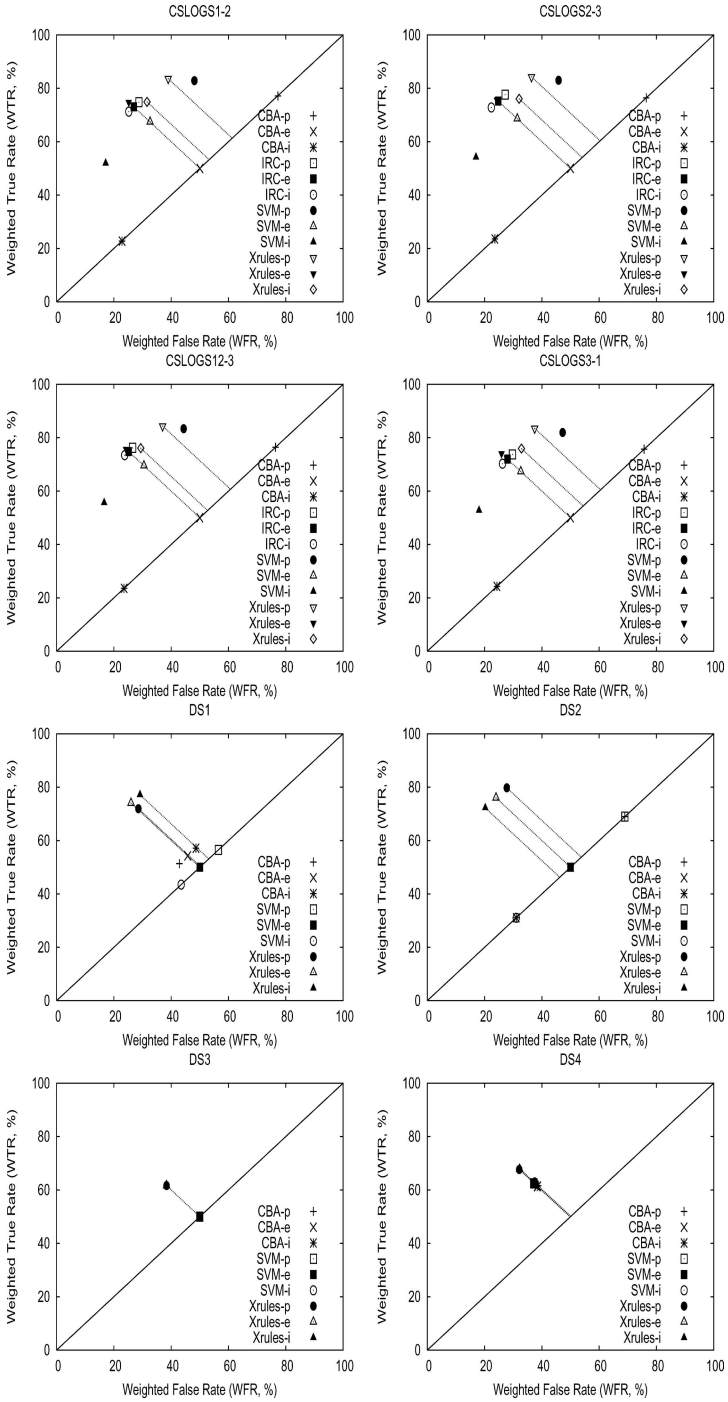


Fig. 9 Weighted ROC analysis: Weighted true rate vs. weighted false rate.

Table 3 Accuracy results and sum of distance from random.

DB	Classifier	Accuracy(%)	Proportional	Equal	Distance sum Inverse
CSLOG1-2	XRULES	**83.63**	**74.83**	**74.93**	0.97
	IRC	74.81	73.04	71.26	**0.98**
	CBA	77.23	50.0	22.77	0.0
	SVM	82.87	67.38	51.89	0.74
	Random	64.84	50.0	35.16	0.0
CSLOG2-3	XRULES	**84.29**	**75.70**	**76.07**	1.01
	IRC	77.64	75.23	72.83	**1.07**
	CBA	76.43	50.0	23.57	0.0
	SVM	83.0	68.6	54.18	0.79
	Random	69.13	50.0	30.87	0.0
CSLOG12-3	XRULES	**84.39**	**75.70**	**76.09**	1.03
	IRC	76.22	74.85	73.47	**1.05**
	CBA	76.43	50.0	23.57	0.0
	SVM	83.39	69.52	55.64	0.83
	Random	69.13	50.0	30.87	0.0
CSLOG3-1	XRULES	**83.51**	**74.09**	**75.95**	**0.97**
	IRC	73.76	72.0	70.26	0.93
	CBA	75.70	50.0	24.30	0.0
	SVM	81.95	67.35	52.75	0.74
	Random	63.21	50.0	36.79	0.0
DS1	XRULES	**71.93**	**74.02**	**77.14**	**0.99**
	CBA	51.35	54.24	57.12	0.18
	SVM	56.5	50.0	43.5	0.0
	Random	50.85	50.0	49.15	0.0
DS2	XRULES	**79.77**	**76.01**	**72.24**	**1.10**
	CBA	68.96	50.0	31.04	0.0
	SVM	68.96	50.0	31.04	0.0
	Random	57.19	50.0	42.81	0.0
DS3	XRULES	**61.63**	**61.63**	**61.63**	**0.49**
	CBA	50.0	50.0	50.0	0.0
	SVM	50.0	50.0	50.0	0.0
	Random	50.0	50.0	50.0	0.0
DS4	XRULES	**67.65**	**67.78**	**67.91**	**0.75**
	CBA	61.65	61.44	61.23	0.49
	SVM	62.5	62.65	62.82	0.54
	Random	50.01	50.0	49.99	0.0

and true negative rates:

$$WTR = w_P\left(\frac{TP}{TP+FN}\right) + w_N\left(\frac{TN}{TN+FP}\right)$$
$$= w_P\left(\frac{TP}{|\mathcal{D}_P|}\right) + w_N\left(\frac{TN}{|\mathcal{D}_N|}\right) = \alpha^{cs}(\mathcal{D}) \tag{13}$$

Thus WTR is the same as weighted accuracy. For the Random classifier, from the analysis of $\alpha^{cs}(\mathcal{D})$ for $k = 2$ given above, we know that for the proportional model $WTR = f_P^2 + f_N^2$, for the equal model $WTR = 0.5$ and for the inverse cost model $WTR = 2 f_P f_N$ Since

$1 - 2f_P f_N = (f_P + f_N)^2 - 2f_P f_N = f_P^2 + f_N^2$ (and similarly $1 - (f_P^2 + f_N^2) = 2f_P f_N$), we immediately see that for Random classifier the *WTR* for proportional costs is equal to $1 - WTR$ for the inverse model.

We can define the Weighted False Rate (WFR) for a classifier, which is given as the weighted average of the false positive and false negative rates:

$$WFR = w_P\left(\frac{FP}{FP+TN}\right) + w_N\left(\frac{FN}{TP+FN}\right) = w_P\left(\frac{FP}{|\mathcal{D}_N|}\right) + w_N\left(\frac{FN}{|\mathcal{D}_P|}\right) \tag{14}$$

Since $FP = |\mathcal{D}_N| - TN$ and $FN = |\mathcal{D}_P| - TP$, we also have

$$\begin{aligned} WFR &= w_P\left(\frac{|\mathcal{D}_N| - TN}{|\mathcal{D}_N|}\right) + w_N\left(\frac{|\mathcal{D}_P| - TP}{|\mathcal{D}_P|}\right) \\ &= w_P\left(1 - \frac{TN}{|\mathcal{D}_N|}\right) + w_N\left(1 - \frac{TP}{|\mathcal{D}_P|}\right) \\ &= w_P + w_N - \left(w_P\frac{TN}{|\mathcal{D}_N|} + w_N\frac{TP}{|\mathcal{D}_P|}\right) \\ &= 1 - \left(w_P\frac{TN}{|\mathcal{D}_N|} + w_N\frac{TP}{|\mathcal{D}_P|}\right) \end{aligned} \tag{15}$$

Notice the "inverse" relationship between WTR (Equation (13)) and WFR (Equation (15)). However, for the random classifier *WTR* and *WFR* are identical. For the random classifier, we expect $f_P|\mathcal{D}_P|$ correct positive predictions (*TP*) and $f_N|\mathcal{D}_N|$ correct negative predictions (*TN*). This means that the remaining $(1 - f_P)|\mathcal{D}_P| = f_N|\mathcal{D}_P|$ positive examples will be incorrectly predicted as negatives (*FN*), and likewise the remaining $(1 - f_N)|\mathcal{D}_N| = f_P|\mathcal{D}_N|$ negative examples will be incorrectly predicted as positives (*FP*). Plugging these values in Equation (14) for the random classifier, we have $WFR = (w_P\frac{f_P|\mathcal{D}_N|}{|\mathcal{D}_N|} + w_N\frac{f_N|\mathcal{D}_P|}{|\mathcal{D}_P|}) = w_P f_P + w_N f_N = WTR$.

Weighted ROC analysis plots the weighted true rate versus the weighted false rate (as opposed to the FPR versus TPR in ROC analysis). An ideal classifier has WTR 100% and WFR 0%, which corresponds to the top left corner of the WROC space. In general, a classifier with as high a WTR and as low a WFR is preferred, thus when comparing multiple classifiers, the more towards the top left corner a classifier is, the better it is in terms of classification performance. Also, given that $WTR = WFR$ for the Random classifier, in the WROC plots shown in Figure 9, the Random classifier is shown as the diagonal line. We expect a reasonable classifier to perform at least as good as the random classifier and thus its acceptable region of the WROC space should lie above the diagonal, and towards the top left corner.

Generally ROC analysis compares multiple classifiers by computing the area of the convex-hull of the points representing each classifier (above the main diagonal); the classifier with the most area has better overall classification performance (Provost & Fawcett, 2001). However, in our case, we only have three points for each algorithm representing the WTR-WFR values under proportional, equal and inverse costs. Instead of computing the area under the convex-hull, we can compute the distance of each classifier (i.e., each point in WROC space) from the diagonal line representing the Random classifier. We can then claim that a classifier with a larger distance has overall better classification performance. There is an obvious $\sqrt{2}/2 = 0.707$ upper limit for the maximum distance from the diagonal, which is

achieved by the ideal classifier with WTR=100% and WFR=0%. In general, the distance for a classifier can be computed using the formula below:

Lemma 5.1. *Given WTR and WFR for a classifier, the distance from the diagonal is given as* $\frac{|WTR-WFR|}{\sqrt{2}}$.

Proof: The diagonal line in the WROC plot corresponds to the line $L : y = x$, with $x \in [0, 1]$ and $y \in [0, 1]$. The point $P = (WFR, WTR)$ represents the classifier in WROC space. We need to first compute the distance of P to L. Let $Q = (x_q, y_q)$ be the point on L closest to P. Since line segment PQ is perpendicular to L, the slope of PQ is -1, and its equation is $PQ : y = -x + b$, where b is some constant. Since P lies on PQ, we have $WTR = -WFR + b$, giving us $b = WTR + WFR$. Since Q is on both PQ and L, we have $y_q = x_q$ and $y_q = -x_q + b$. Solving for x_q, we get $x_q = -x_q + b$ and $x_q = b/2$. Thus $Q = (b/2, b/2)$, and $\|PQ\|_2 = \sqrt{(WFR - b/2)^2 + (WTR - b/2)^2} = \frac{|WTR-WFR|}{\sqrt{2}}$. □

Lemma 5.2. *The distance from the diagonal for a classifier is given as* $\frac{|TPR-FPR|}{\sqrt{2}}$.

Proof: Let d be the distance from random, $d = \sqrt{\frac{(WTR-WFR)^2}{2}}$ given in Lemma 5.1. Consider the numerator term $n = WTR - WFR$. We substitute Equations (13) and (15) for WTR and WFR, to get $n = w_P(\frac{TP}{|\mathcal{D}_P|}) + w_N(\frac{TN}{|\mathcal{D}_N|}) - (1 - (w_P\frac{TN}{|\mathcal{D}_N|} + w_N\frac{TP}{|\mathcal{D}_P|}))$. After simplifying we get $n = (w_P + w_N)(\frac{TP}{|\mathcal{D}_P|} - \frac{FP}{|\mathcal{D}_N|}) = \frac{TP}{|\mathcal{D}_P|} - \frac{FP}{|\mathcal{D}_N|} = TPR - FPR$. Putting n back into the formula for d we get the desired result. □

5.3.2. CSLOGS datasets

Let us consider the accuracy comparison among the different classifiers shown in Table 3. We can see that for all data sets and all cost models, XRULES is the best classifier. For the CSLOG data sets, XRULES delivers an accuracy between 83.51% and 84.39% for the proportional model compared to IRC's accuracy from 73.76% to 77.64%, CBA's accuracy between 75.7% to 77.23%, and SVM's accuracy between 81.95% to 83.39%. Thus, the accuracy of XRULES is about 8-10% higher (in absolute difference) than that of IRC, 5–10% higher than that of CBA, and about 1–1.5% higher than SVM, for the traditional proportional model. For this model, CBA appears to be a better classifier than IRC. However, the model that CBA learns generally has only one rule. This rule always predicts a test case to be `other`. While this strategy pays off in the proportional cost model (since `other` is the majority class with 76–79% occurrence), it does not work for the equal model (50% accuracy) and fails completely for the inverse cost model (23–24% accuracy). IRC does a much better job than CBA in distinguishing one class from the other.

For the equal and inverse cost models, we find that XRULES has higher accuracy than SVM, CBA and IRC, though IRC's performance is close to that of XRULES. The accuracy of XRULES is about 0.5%–2% (absolute difference) higher than IRC, 25% higher than CBA, and about 7% higher than SVM, for the equal cost model. The situation is more pronounced for inverse model, where the accuracy of XRULES is 2.5–6% higher than IRC, 50% higher than CBA, and 25% higher than SVM. These results are in agreement with our approach of maximizing the weighted accuracy of XRULES for different cost models.

Accuracy alone, however, conveys only one aspect of classification performance. A more clear picture emerges when one considers the WROC analysis plotting WTR against WFR,

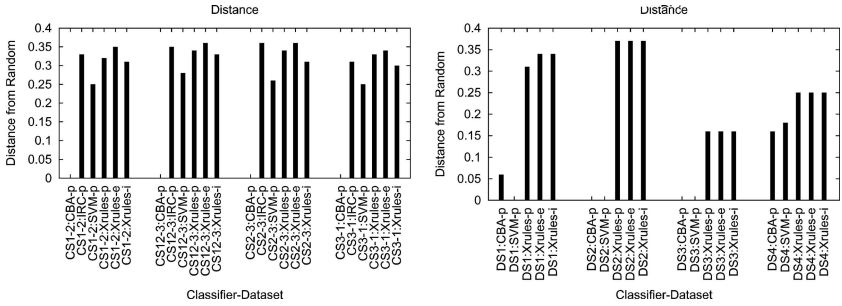


Fig. 10 Distance from random.

as shown in Figure 9. The WROC plots show the performance for all three cost models. For example, XRULES-p, XRULES-e, and XRULES-i denote the performance of XRULES under the proportional, equal and inverse cost models, respectively. Note that in Figure 9, we have also drawn the distance of the XRULES(-p/-e/-i) from the Random classifier (i.e., from the diagonal line).

Figure 10 plots the actual distance from the diagonal for different datasets and classifiers (the left figure shows the results for the CSLOGS datasets, whereas the right figure shows results for the synthetic datasets). For IRC, SVM and CBA, we show the distance only for IRC-p, SVM-p, CBA-p (i.e., for the proportional cost model), since the distance for the other cost models is the same for these classifiers. That is, for classifiers that do not explicitly model cost, their TPR and FPR remain the same regardless of the cost model, which implies, by Lemma 5.2, that their distance from random remains constant. For XRULES\ all three distances are shown (XRULES-p, XRULES-e, XRULES-i). For proper comparison using the WROC plots and distances, we have to compare classifiers (i.e., points in the WROC space) separately for each cost model. For example, we compare XRULES-p only with other classifiers for the same cost model, namely CBA-p, SVM-p, and IRC-p, and so on.

From the WROC analysis in Figure 9 and the distances in Figure 10, we can clearly see that CBA is no better than the Random classifier, since its WTR and WFR lie on the diagonal for all three cost models; thus the distance for CBA is also 0 for all the CSLOGS datasets. Looking at the areas in Figure 10, we find that XRULES is always better than SVM; the distance of SVM is 0.25 for all three costs, whereas the distance of XRULES ranges from 0.3–0.35 for the different models. Comparing against IRC, XRULES\ always has higher distance for the equal model, but has slightly lower distance for the proportional and inverse costs. In all cases though, as per Table 3, XRULES has higher accuracy. Also once we look at the sum of the distance from Random for all three cost models for each classifier, as shown in the last column of Table 3, we can see that for all datasets XRULES has higher or competitive performance against the other classifiers. However for the CSLOGS datasets, IRC represents a viable alternative; IRC has a slightly better distance score, but XRULES gives better accuracy for about the same distance score. This is interesting since IRC actually uses the text, i.e., the content, of the documents to make its classification decisions, information that is ignored by XRULES. Nevertheless, there seems to be enough information conveyed by the structure alone to make XRULES competitive with IRC. For the other classifiers that ignore content, XRULES is clearly superior. These results also point to the fact that it may be possible to improve upon both IRC and XRULES by combining both content and structure. We will explore this as part of future work.

5.3.3. *Synthetic dataset*

On synthetic data sets, which do not have content (only structure), the IR classifier does not work. So we compared only XRULES, SVM and CBA. Accuracy results are shown in Table 3, the WROC plots are shown in Figure 9, and the distances from Random are shown in Figure 10. On these synthetic datasets XRULES is clearly superior to both SVM and CBA, not only in terms of accuracy, but also in terms of the distance. In fact, for all cost models XRULES outperforms both SVM and CBA.

For DS2 and DS3, both SVM and CBA degenerate into a random classifier. SVM also performs poorly on DS1, whereas CBA achieves slightly better WTR-WFR ratios. For DS4, both SVMs and CBA tend to do better, but are clearly worse than XRULES. In terms of accuracy, XRULES tends to have 6-20% (absolute difference) higher values than CBA and SVM for the proportional model across the four datasets, 5–25% higher values for equal and 5-40% higher accuracies for the inverse model. Out of the the four datasets, DS3 is clearly the toughest for any set-based classifier. The reason is that DS3 has full two-way addition of trees from class c_1 to class c_2 and vice versa. Thus both CBA and SVM see the same set of features for the two classes and are unable to discriminate between them. On the other hand if we reduce the two way addition to only half the time, as in DS4, both of them are able to discard the incompatible examples and learn the rules from the remaining examples. Both DS1 and DS2 have only one-way addition, i.e., examples from c_1 are added to c_2, so we expect both CBA and SVM to discriminate c_2 from c_1 at least by some amount. Nevertheless, these feature/set-based methods are not able to extract the embedded structure. Thus, from the clear superiority of XRULES for structural classification tasks, we can safely assume that for those datasets whose structure conveys something about class, we can expect XRULES to be very effective.

In summary, XRULES gives consistently better (or competitive) performance than the other classifiers for all cost models and data sets. It works better than an associative classification approach like CBA, which flattens out the structure into a set representation. It outperforms the SVM classifier across the board for both synthetic and real datasets. In terms of accuracy, it even outperforms an IR based classifier, which explicitly learns over the content, but only implicitly over the structural information in the XML documents. Using WROC analysis, we found out that the IR classifier and XRULES tend to have comparable overall performance (WTR-WFR values) on the CSLOGS datasets, since they use different kinds of information to maximize their classification results, and this suggests that a joint content and structure based classification approach is likely to perform even better.

5.4. Efficiency results

Table 2 shows the number of frequent patterns (rules) mined by XMINER, and time for training and testing. The table also shows the minimum support value used for the two classes. Only for CSLOGS1-2 we used a different value for `edu` (0.3%) and for `other` (0.25%). Note that the Rules column gives the total number of rules mined. Out of these the number given under Trees column are actually trees (i.e., have at least one branch), where as the rest are plain sequences (i.e., linear trees). We can see that the number of actual trees ranges from 14.5% (for CSLOGS3-1) to 71.5% (for CSLOGS12-3). Thus it appears that the tree mining is adding real value to the mined rules, as opposed to only sequence mining. The results underscore the high efficiency of that XMINER (XM) engine. The frequent trees for classification are determined in less than 8 seconds. The total training and testing time are comparable, since in both cases we have to find the matching rules for each example. This

Table 4 Effect of strength measure.

DB	Strength	Proportional	Equal	Inverse
CSLOG1	γ	81.47	74.73	**74.93**
	ρ	**83.63**	72.33	68.21
	ρ^w	81.82	**74.83**	74.78
CSLOG2	γ	82.14	**75.70**	**76.07**
	ρ	**84.29**	73.95	70.55
	ρ^w	82.42	75.69	75.77
CSLOG12	γ	81.58	75.76	**76.09**
	ρ	**84.39**	73.69	69.14
	ρ^w	82.0	**75.70**	75.56
CSLOG3	γ	81.03	**74.09**	**75.95**
	ρ	**83.51**	73.02	71.36
	ρ^w	81.15	74.07	75.78
DS1	γ	70.44	73.68	76.93
	ρ	**71.93**	71.69	71.44
	ρ^w	70.90	**74.02**	**77.14**
DS2	γ	56.37	61.65	66.93
	ρ	**79.77**	**76.01**	**72.24**
	ρ^w	61.37	65.06	68.76
DS3	γ	**61.63**	**61.63**	**61.63**
	ρ	61.45	61.45	61.45
	ρ^w	61.45	61.45	61.45
DS4	γ	59.02	58.88	58.74
	ρ	**67.65**	**67.78**	**67.91**
	ρ^w	61.71	61.62	61.52

is needed to determine the default class in training, and to find the accuracy in testing. The run time can be improved by storing the rules in an appropriate index structure; currently XRULES performs a linear search for matching rules.

5.5. Choice of rule strength

We next study how the choice of strength measure affects the accuracy of XRULES, as shown in Table 4. The best results are in bold. For the proportional model, confidence performs better than both likelihood and weighted confidence. Its accuracy is typically 2–4% higher on CSLOG and as much as 20% higher on DSx data sets. This is in agreement based on the Bayesian interpretation in Section 2.4.4. On the other hand, with the exception of DS2 and DS4, likelihood and weighted confidence perform better than confidence with equal cost model. The weighted confidence has a slight (if insignificant) edge over likelihood (for both proportional and equal costs).

The likelihood measure has a slight edge over weighted confidence for the inverse cost model on CSLOG data sets. These results are in agreement with the discussion in Section 2.4.4. The only exceptions are DS2 and DS4 where confidence does better. The reason is that in these data sets the confusion factor complicates the decision making, since one-way (two-way) addition adds patterns from one class to the other (and vice-versa). result.

In summary, we conclude that confidence is a better measure for proportional model and either likelihood or weighted confidence is better for equal or inverse costs. The right choice of strength measure depends on the data set characteristics and cost model. If we expect

Table 5 Effect of Likelihood Ratio (CSLOG1-2).

$\gamma^{\min}$	Proportional	Equal	Inverse	#Rules	Time
1	81.47	**74.73**	74.93	28911	176.1
2	81.98	74.67	75.07	28553	173.9
3	82.89	73.83	75.78	25698	155.8
4	83.28	72.58	75.83	24190	144.1
5	**83.51**	72.14	76.39	17732	105.9
6	83.32	69.93	77.30	12006	71.4
7	83.20	69.37	77.10	10008	59.2
8	82.89	67.81	77.34	8936	54.5
9	82.44	66.27	77.55	8424	49.1
10	82.41	65.79	**77.63**	8199	47.5
15	81.81	62.62	77.42	7848	45.6
20	81.06	60.28	77.38	7636	44.3

many patterns with similar global supports but different local supports of rare classes, the likelihood/weighted confidence measure will usually provide better results.

5.6. Effect of minimum strength

Table 5 shows the effect of varying the minimum likelihood $\gamma^{\min}$ on the accuracy of prediction for CSLOG1-2. Best accuracy for each cost model is in bold. For proportional cost model, the accuracy tends to increase up to a point (83.51% for $\gamma^{\min} = 5$) and then starts to drop. The same effect is observed for inverse model, but the model continues to improve until $\gamma^{\min} = 10$. For the equal cost model, the accuracy tails off at the very beginning. Similar results were obtained for other strength measures. These results suggest that by choosing an appropriate $\gamma^{\min}$ one can get a model that can behave like ρ for the proportional model (e.g., at $\gamma^{\min} = 5$, we get 83.51% accuracy compared to 83.49% accuracy using confidence, in Table 4), and can improve the accuracy for the inverse model.

6. Related Work

Tree mining, being an instance of frequent structure mining, has obvious relation to association (Agrawal *et al.,* 1996) and sequence (Agrawal & Srikant, 1995) mining. Frequent tree mining is also related to tree isomorphism (Shamir & Tsur, 1999), tree pattern matching (Cole, Hariharan, & Indyk, 1999) and tree inclusion (Kilpelainen & Mannila, 1995). Both subtree isomorphism and pattern matching deal with induced subtrees, while we mine embedded subtrees. Further, we are interested in enumerating all common subtrees in a collection of trees, and using them for building a classifier.

With the advent of XML as a data representation and exchange standard, there has been active work in indexing and querying XML documents, which are mainly tree (or graph) structured; to efficiently answer ancestor-descendant queries various node numbering schemes similar to ours have been proposed (Li & Moon, 2001; Zhang *et al.,* 2001;, Abiteboul, Kaplan, & Milo, 2001). The major difference between these works and ours is that instead of answering user-specified queries based on regular path expressions, we are interested in finding all frequent tree patterns among the documents.

Tree mining has attracted a lot of attention recently. We developed TreeMiner (Zaki, 2002) to mine labeled, embedded, and ordered subtrees. The notions of scope-lists and rightmost extension were introduced in that work. (Asai *et al.* (2002) presented FreqT, an apriori-like algorithm for mining labeled ordered trees; they independently proposed the rightmost candidate generation scheme. Wang and Liu (Wang & Liu, 2000) developed an algorithm to mine frequently occurring subtrees in XML documents. Their algorithm is also reminiscent of the level-wise A priori (Agrawal *et al.,* 1996) approach, and they mine induced subtrees only. There are several other recent algorithms that mine different type of tree patterns, which include FreeTreeMiner (Chi, Yang, & Muntz, 2003) which mines induced, unordered, free trees (i.e., there is no distinct root); and PathJoin (Xioa *et al.,* 2003), uFreqt (Nijssen & Kok, 2003), uNot (Asai *et al.,* 2003), and HybridTreeMiner (Chi, Yang, & Muntz, 2004) which mine induced, unordered trees. TreeFinder (Termier, Rousset, & Sebag, 2002) uses an Inductive Logic Programming approach to mine unordered, embedded subtrees, but it is not a complete method, i.e, it can miss many frequent subtrees, especially as support is lowered or when the different trees in the database have common node labels. A recent method, XSpanner (Wang *et al.,* 2004), based on a pattern-growth strategy, to mine embedded trees has also been proposed.

There has also been recent work in mining frequent graph patterns. The AGM algorithm (Inokuchi, Washio, & Motoda, 2000) discovers induced (possibly disconnected) subgraphs. The FSG algorithm (Kuramochi & Karypis, 2001) improves upon AGM, and mines only the connected subgraphs. Both methods follow an Apriori-style level-wise approach. Recent methods to mine graphs using a depth-first tree based extension have been proposed in (Yan & Han, 2002, 2003). Another method uses a candidate generation approach based on Canonical Adjacency Matrices (Huan Wang, & Prins, 2003). There are important differences in graph mining and tree mining. Our trees are rooted, and thus have a unique ordering of the nodes based on depth-first traversal. In contrast graphs do not have a root, and allow cycles. For mining graphs the methods above first apply an expensive *canonization* step to transform graphs into a uniform representation. This step is unnecessary for tree mining. Graph mining algorithms are likely to be overly general (thus not efficient) for tree mining. Our approach utilizes the tree structure for efficient enumeration.

The work by Dehaspe *et al.* (Dehaspe, Toivonen, & King, 1998) describes a level-wise Inductive Logic Programming (ILP) based technique to mine frequent substructures (subgraphs) describing the carcinogenesis of chemical compounds. Work on molecular feature mining has appeared in (Kramer, Raedt, & Helma, 2001). The SUBDUE system (Cook & Holder, 1994) also discovers graph patterns using the Minimum Description Length principle. An approach termed Graph-Based Induction (GBI) was proposed in (Yoshida & Motoda, 1995), which uses beam search for mining subgraphs. However, both SUBDUE and GBI may miss some significant patterns, since they perform a heuristic search. In contrast to these approaches, we are interested in developing efficient algorithms for tree patterns, and their use for classification.

The classification problem has been widely studied by the database, data mining, machine learning, and statistics communities (Aggarwal, 2002; Alsabti, Ranka & Singh, 1998; Ankerst, Ester, & Kriegel, 2000; Cohen 1995; Duda & Hart, 1973; Friedman, 1997; Garofalakis *et al.,* 2000; Gehrke *et al.,* 1999; James, 1985; Quinlan 1993; Murthy 1998; Rastogi & Shim 1998; Liu, Hsu, & Ma, 1998; Mehata, Agrawal, & Rissanen, 1996; Safer, Agrawal, & Meheta, 1996), However, most such methods have been developed for general multi-dimensional records. For a particular data domain such as strings or text (Aggrawal 2002; Nigam *et al.,* 2000), classification models specific to these domains turn out to be most effective. Frequent pattern based association rule classifiers such as CBA (Liu, Hau, & Ma,

1998), CAEP (Dong *et al.,* 1999) or CMAR (Li, Han, & Pei, 2001) have also recently proven successful. However, these work only on feature-vector data, whereas we are interested in structural classification. Previous work has looked at the problem of mining sequential features for use in classification (Kudenko & Hiris, 1998; Lesh, Zaki, & Ogihara, 2000). A similar approach can also be tried for trees. Recently there has been interest in learning from relational and structural data (Dzeroski & Lavrac, 2001). Previously, we proposed the Xrules method for building a structural classifier for XML data (Zaki & Aggarwal, 2003). This current paper significantly expands on the preliminary conference version. An SVM-based approach for structured output spaces has recently been proposed (Tsochantaridis *et al.,* 2004). An approach that utilizes the dependencies in a network of objects to improve predictions was presented in (Neville & Jensen, 2003). Work has also been done to build relational Bayesian classifiers (Neville, Jensen & Gallagher, 2003).

7. Summary

In this paper, we discussed an effective rule based classifier for XML data called XRULES. The technique mines frequent structures from the data in order to create the classification rules. XRULES is cost-sensitive and uses Bayesian rule based class decision making. Methods for effective rule prioritization and testing were also proposed in this paper. The technique was implemented and compared against CBA as well as an IR classifier. The better performance of XRULES over CBA might be explained through classification information being embedded in the XML structure, which XRules uses, but CBA does not. Furthermore, it outperforms the IR based method in spite of the greater amount of input used by the latter. The results show that structural mining can provide new insights into the process of XML classification.

Acknowledgments This work was supported in part by NSF CAREER Award IIS-0092978, DOE Career Award DE-FG02-02ER25538, and NSF grants EIA-0103708 and EMT-0432098.

References

Abiteboul, S., Kaplan, H., & Milo, T. (2001). Compact labeling schemes for ancestor queries. *ACM Symp. on Discrete Algorithms*.

Abiteboul, S., & Vianu, V. (1997). Regular path expressions with constraints. *ACM Int'l Conf. on Principles of Database Systems*.

Aggarwal, C. C. (2002). On effective classification of strings with wavelets. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.

Aggarwal, C., Gates, S., & Yu, P. (1999). On the merits of using supervised clustering to build categorization systems. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.

Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. *Int'l Conf. on Very Large Data Bases*.

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *IEEE Int'l Conf. on Data Engineering*.

Alsabti, K., Ranka, S., & Singh, V. (1998). CLOUDS: A decision tree classifier for large datasets. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.

Andersen, R., Birbeck, M., Kay, M., Livingstone, S., Loesgen, B., Martin, D., Mohr, S., Ozu, N., Peat, B., Pinnock, J., Stark, P., & Williams, K. (2002). *Professional XML*. Wrox Press Ltd.

Ankerst, M., Ester, M., & Kriegel, H.-P. (2000). Towards an effective cooperation of the computer and the user for classification. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.

Asai, T., Abe, K., Kawasoe, S., Arimura, H., Satamoto, H., & Arikawa, S. (2002). Efficient substructure discovery from large semi-structured data. *SIAM Int'l Conf. on Data Mining*.

Asai, T., Arimura, H., Uno, T., & Nakano, S. (2003). Discovering frequent substructures in large unordered trees. *Int'l Conf. on Discovery Science*.

Boz, O. Cost-sensitive learning bibliography. http://home.ptd.net/ olcay/cost-sensitive.html.
Chi, Y., Yang, Y., & Muntz, R. R. (2003). Indexing and mining free trees. *IEEE Int'l Conf. on Data Mining*.
Chi, Y., Yang, Y., & Muntz, R. R. (2004). Hybridtreeminer: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms. *Int'l Conf. on Scientific and Statistical Database Management*.
Cole, R., Hariharan, R., & Indyk, P. (1999). Tree pattern matching and subset matching in deterministic $o(n \log^3 n)$-time. *10th ACM Symp. on Discrete Algorithms*.
Cook, D., & Holder, L. (1994). Substructure discovery using minimal description length and background knowledge. *Journal of Artificial Intelligence Research, 1*, 231–255.
Cohen, W. W. (1995). Fast effective rule induction. *Int'l Conf. on Machine Learning*.
Doan, A., Domingos, P., & Halevy, A. (2001). Reconciling schemas of disparate data sources: A machine learning approach. *ACM SIGMOD Conf.*
Dehaspe, L., Toivonen, H., & King, R. (1998). Finding frequent substructures in chemical compounds. *ACM Int'l. Conf. on Knowledge Discovery and Data Mining*.
Domingos, P. (1999). MetaCost: A general method for making classifiers cost sensitive. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.
Dong, G., Zhang, X., Wong, L., & Li, J. (1999). CAEP: Classification by aggregating emerging patterns. *Int'l Conf. on Discovery Science*.
Duda, R., & Hart, P. (1973). *Pattern classification and scene analysis*. New York: Wiley.
Dumais, S., Platt, J., Heckerman, D., & Sahami, M. (1988). Inductive learning algorithms and representations for text categorization. *ACM Int'l Conf. on Information and Knowledge Management*.
Dzeroski, S., & Lavrac, N. (Eds.) (2001). *Relational data mining*. Springer-Verlag.
Friedman, J. H. (1977). A recursive partitioning decision rule for non-parametric classifiers. *IEEE Transactions on Computers, C-26*, 404–408.
Garofalakis, M., Hyun, D., Rastogi, R., & Shim, K. (2000). Efficient algorithms for constructing decision trees with constraints. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.
Gehrke, J., Ganti, V., Ramakrishnan, R., & Loh, W.-Y. (1999). BOAT: Optimistic decision tree construction. *ACM Int'l Conf. on Management of Data*.
Gehrke, J., Ramakrishnan, R., & Ganti, V. (1998). RainForest: A framework for fast decision tree construction of large data sets. *Int'l Conf. on Very Large Data Bases Proceedings*.
Gehrke, J., Loh, W.-Y., & Ramakrishnan, R. (1999). Data mining with decision trees. *ACM ACM Int'l Conf. on Knowledge Discovery and Data Mining Tutorial*.
Huan, J., Wang, W., & Prins, J. (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. *IEEE Int'l Conf. on Data Mining*.
Inokuchi, A., Washio, T., & Motoda, H. (2000). An apriori-based algorithm for mining frequent substructures from graph data. *European Conf. on Principles of Knowledge Discovery and Data Mining*.
James, M. (1985). *Classification algorithms*. John Wiley & Sons.
Joachims T. (2002). *Learning to classify text using support vector machines*. Kluwer Academic Publishers.
Kilpelainen, P., & Mannila, H. (1995). Ordered and unordered tree inclusion. *SIAM J. of Computing, 24*(2), 340–356.
Kramer, S., De Raedt, L., & Helma, C. (2001). Molecular feature mining in HIV data. *Int'l Conf. on Knowledge Discovery and Data Mining*.
Kudenko, D., & Hirsh, H. (1998). Feature generation for sequence categorization. *AAAI National Conf. on Artificial Intelligence*.
Kuramochi, M., & Karypis, G. (2001). Frequent subgraph discovery. *IEEE Int'l Conf. on Data Mining*.
Lesh, N., Zaki, M. J., & Ogihara, M. (2000). Scalable feature mining for sequential data. *IEEE Intelligent Systems, 15*(2), 48–56.
Li, Q., & Moon, B. (2001). Indexing and querying XML data for regular path expressions. *Int'l Conf. on Very Large Data Bases*.
Li, W., Han, J., & Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. *IEEE Int'l Conf. on Data Mining*.
Liu, B., Hsu, W., & Ma, Y. (1998). Integrating classification and association rule mining. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.
Masciari, E. Personal Communication.
Mehta, M., Agrawal, R., & Rissanen, J. (1996). SLIQ: A fast scalable classifier for data mining. *Int'l Conf. on Extending Data Base Technology*.
Murthy, S. K. (1998). Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery, (2)*4, 345–389.
Neville, J., & Jensen, D. (2003). Collective classification with relational dependency networks. *Multi-Relational Data Mining Workshop*.

Neville, J., Jensen, D., & Gallagher, B. (2003). Simple estimators for relational bayesian classifers. *Int'l Conf. on Data Mining*.

Nigam, K., McCallum, A. K., Thrum, S., & Mitchell, T. (2000). Text classification from labeled and unlabeled documents using EM. *Machine Learning, 39*(2/3), 103–134.

Nijssen, S., & Kok, J. N. (2003). Efficient discovery of frequent unordered trees. *Int'l Workshop on Mining Graphs, Trees and Sequences*.

Provost, F., & Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning, 42*, 203–231.

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

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Rastogi, R., & Shim, K. (1998). PUBLIC: A decision tree classifier that integrates building and pruning. *Int'l Conf. on Very Large Data Bases*.

Shafer, J., Agrawal, R., & Mehta, M. (1996). SPRINT: A scalable parallel classifier for data mining. *Int'l Conf. on Very Large Data Bases*.

Shamir, R., & Tsur, D. (1999). Faster subtree isomorphism. *Journal of Algorithms, 33*, 267–280.

Termier, A., Rousset, M-C., & Sebag, M. (2002). TreeFinder: A first step towards XML data mining. *IEEE Int'l Conf. on Data Mining*.

Tsochantaridis, I., Hofmann, T., Joachims, T., & Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. *Int'l Conf. on Machine Learning*.

Wang, K., & Liu, H. Q. (1998). Discovering typical structures of documents: A road map approach. *ACM Int'l Conf. on Information Retrieval*.

Xiao, Y., Yao, J.-F., Li, Z., & Dunham, M. H. (2003). Efficient data mining for maximal frequent subtrees. *Int'l Conf. on Data Mining*.

Yan, X., & Han, J. (2002). Gspan: Graph-based substructure pattern mining. *IEEE Int'l Conf. on Data Mining*.

Yan, X., & Han, J. (2003). CloseGraph: Mining closed frequent graph patterns. *ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining*.

Yoshida, K., & Motoda, H. (1995). CLIP: Concept learning from inference patterns. *Artificial Intelligence, 75*(1), 63–92.

Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. *ACM Int'l Conf. on Knowledge Discovery and Data Mining*.

Zaki, M. J., & Aggarwal, C. C. (2003). Xrules: An effective structural classifier for xml data. *ACM Int'l Conf. Knowledge Discovery and Data Mining*.

Zhang, C., Naughton, J., DeWitt, D., Luo, Q., & Lohman, G. (2001). On supporting containment queries in relational database management systems. *ACM Int'l Conf. on Management of Data*.