XRules: An Effective Structural Classifier for XML Data

Mohammed J. Zaki *
Rensselaer Polytechnic Institute
zaki@cs.rpi.edu

Charu C. Aggarwal
IBM T.J. Watson Research Center
charu@us.ibm.com

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.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Data Mining

Keywords

XML/Semi-structured data, Classification, 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 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. The classification problem has been widely studied by the database, data mining and machine learning communities [1, 4, 7, 10, 11, 12, 14, 15, 16]. However, most such methods have been developed for general multi-dimensional records. For a particular data domain such as strings or text [1, 17], classification models specific to these domains turn out to be most effective.

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 [10]. This method works quite well for most text applications containing a small number of class labels. However, the use of the text format for classification ignores a significant amount of structural information in the XML documents. In many cases, the classification behavior of the XML document is 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. A second, but more promising methodology for XML mining is to directly use association based classifiers such as CBA [16], CAEP [9] or CMAR [13], 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. This also results in loss of structural information, but the overall accuracy is still somewhat better than the "bag of words" approach for a text classifier.

Recent work has focused on the use of rule based classifiers [16] as an effective tool for data classification. Rule based classifiers have also been extended to the string classification problem [1]. Rule based classifiers are an interesting method which integrate the problem of associations and classification. These techniques provide an effective and scalable alternative for classification, and often turn out to be highly interpretable by their nature.

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

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

SIGKDD '03, August 24-27, 2003, Washington, DC, USA

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

We model XML documents as ordered, labeled, rooted trees, i.e., child order matters, and each node has a label. We do not distinguish between attributes and elements of an XML document; both are mapped to the label set.

2.1 Trees and Embedded Subtrees

We denote a tree (an XML document) as $T = (V, B)$, where V is the set of labeled nodes, and B the set of branches. The label of each node is taken from a set of labels (also called items) $L = \{1, 2, 3, ..., m\}$; different nodes can have the same label. Each branch, $b = (x, y)$, is an ordered pair of nodes, where x is the parent of y. The *size* of T is the number of nodes in T.

We say that a tree $S = (V_s, B_s)$ is an *embedded subtree* of $T = (V, B)$, denoted as $S \preceq T$, provided i) $V_s \subseteq V$, and ii) $b = (x, y) \in B_s$, if and only if x is an ancestor of y in T. Note that in the traditional definition of an *induced* subtree, for each branch $b = (x, y) \in B_s$, x must be a parent of y in T. 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 will not be captured by the traditional definition. If $S \preceq T$, we also say that T *contains* S. A (sub)tree of size l is also called a l-(sub)tree.

2.2 Cost-based Classification

The classification model discussed in this paper can be used for the general case of cost-sensitive classification [8]. 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 $\mathcal{C} = \{c_1 \dots c_k\}$ 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.[1] 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: $\alpha(\mathcal{R}, \mathcal{D}) = \frac{\eta(\mathcal{D})}{|\mathcal{D}|}$.

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

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

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

Let D be any collections of trees with class labels drawn from $\mathcal{C}$. 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 T, i.e.,

$$\pi^A(T, D) = |\{S \in D | T \preceq S\} \quad (2)$$

The *(relative) support* of T in D, denoted $\pi(T, D)$, as the fraction of trees in D that contain T, i.e.,

$$\pi(T, D) = \frac{\pi^A(T, D)}{|D|} \quad (3)$$

[1] Tree Structured XML documents are the most widely occurring 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 [5].

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.
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 were refer to as support and strength. 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

$$\pi(T \Rightarrow c_i) = P(T \wedge c_i) = \frac{\pi^A(T, \mathcal{D}_i)}{|\mathcal{D}|} = \pi(T, \mathcal{D}_i) \times \frac{|\mathcal{D}_i|}{|\mathcal{D}|} \quad (4)$$

The last step follows from Equation 3. 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.
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

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

Let's assume that we have k classes ($k \geq 2$), and let $\overline{\mathcal{C}}_i = \mathcal{C} - \{c_i\}$ be the set of all classes other than c_i. We define $\overline{\mathcal{D}}_i = \mathcal{D} - \mathcal{D}_i$ to be the set of trees in $\mathcal{D}$ with their classes taken from $\overline{\mathcal{C}}_i$. 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 $\overline{\mathcal{C}}_i$. That is, we compare $\rho(T \Rightarrow c_i)$ with $\rho(T \Rightarrow \overline{\mathcal{C}}_i)$. Using the observation that $\mathcal{D} = \mathcal{D}_i + \overline{\mathcal{D}}_i$, we can rewrite Equation 5 as:

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

It is clear that $\rho(T \Rightarrow c_i) = 1 - \rho(T \Rightarrow \overline{\mathcal{C}}_i)$.
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 $\overline{\mathcal{C}}_i$. Formally, it is defined as follows:

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

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 $\pi^A(T, \overline{\mathcal{D}}_i)$). Plugging into Equation 7, $\gamma(T \Rightarrow c_i) = \frac{\rho(T \Rightarrow c_i) \times \pi^A(T, \mathcal{D})}{\rho(T \Rightarrow \overline{\mathcal{C}}_i) \times \pi^A(T, \mathcal{D})} \frac{|\overline{\mathcal{D}}_i|}{|\mathcal{D}_i|} = \frac{\rho(T \Rightarrow c_i)}{\rho(T \Rightarrow \overline{\mathcal{C}}_i)} \frac{|\overline{\mathcal{D}}_i|}{|\mathcal{D}_i|}$. □
Weighted Confidence: We define another measure called the *weighted confidence*, which combines the above two measures, given as follows:

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

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 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:*

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

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:
$\rho^w(T \Rightarrow c_i) = \frac{\gamma(T \Rightarrow c_i) \times \pi(T, \overline{\mathcal{D}}_i)}{\gamma(T \Rightarrow c_i) \times \pi(T, \overline{\mathcal{D}}_i) + \pi(T, \overline{\mathcal{D}}_i)} = \frac{\gamma(T \Rightarrow c_i)}{\gamma(T \Rightarrow c_i) + 1}$ □
Like ρ the value of ρ^w lies between $[0, 1]$, while γ can take values between $[0, \infty]$. In our experiments, we will study the effects of using one measure over the other. Let δ denote the measure of strength; for confidence $\delta \equiv \rho$ [2], for weighted confidence $\delta \equiv \rho^w$, and for likelihood $\delta \equiv \gamma$.

We use the notation $T \overset{\pi, \delta}{\Rightarrow} c_i$ to denote a rule with support π and strength δ. Our goal is to learn a structural ruleset $\mathcal{R} = \{R^1, R^2, \cdots, R^m\}$, where each rule is of the form $R^i : T^i \overset{\pi, \delta}{\Rightarrow} c^i$, with $\pi \geq \pi_j^{\min}$ and with $\delta \geq \delta^{\min}$. That is, rules which satisfy a user-defined level of minimum support $\pi^{\min}$, 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 $\gamma^{\min} = 1.0$;

2.4.1 Bayesian Interpretation of Strength

Given k classes $\mathcal{C} = \{c_1, \ldots, c_k\}$, with $\overline{\mathcal{C}}_i = \mathcal{C} - c_i$. 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 $\overline{\mathcal{C}}_i$. 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 classes, i.e., assign T to class c_i if $P(c_i|T) > P(c_j|T)$. Since we compare a class c_i against the negative class $\overline{\mathcal{C}}_i$, we assign T to class c_i if

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

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

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

The three strength measures differ in which equation they use for class prediction. For instance, confidence measure

[2]The notation $\equiv$ denotes that the two entities are equivalent.

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 $\frac{P(T|c_i)}{P(T|\overline{\mathcal{C}}_i)} > \frac{P(\overline{\mathcal{C}}_i)}{P(c_i)}$. Plugging $P(T|c_i) = \frac{\pi^A(T,\mathcal{D}_i)}{|\mathcal{D}_i|} = \pi(T,\mathcal{D}_i)$ (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 $\gamma(T \Rightarrow c_i) = \frac{\pi(T,\mathcal{D}_i)}{\pi(T,\overline{\mathcal{D}}_i)}$. 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 $\frac{P(\overline{\mathcal{C}}_i)}{P(c_i)} = 1$. In general (for proportional or equal cost model) it makes logical 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 11 (consider its LHS):

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

Once again, ignoring class priors ratio (i.e, setting $\frac{P(\overline{\mathcal{C}}_i)}{P(c_i)} = 1$), we obtain the definition of weighted confidence in Equation 9. Thus LHS of Equation 11 corresponds to $\rho^w(T \Rightarrow c_i)$, and by Bayes rules we assign T to class c_i if $\rho^w(T \Rightarrow c_i) > \rho^w(T \Rightarrow \overline{\mathcal{C}}_i)$.
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 $\overline{\mathcal{C}}_i$ (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 $\mathcal{R} = \{R^1, R^2, \cdots, R^m\}$, 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 a 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., $\pi_j^{\min}, \forall j = 1, \cdots k$. XMiner outputs a set of frequent rules for each class, $\mathcal{R}_j = \{R^1, \cdots, R^{m_j}\}$, 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

Since we want only predictive rules, we need to remove any rule that lacks predictive power. Consider 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 $\overline{\mathcal{C}}_i$, 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 [16].

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 the following conditions are met:

1. The strength of R^i is greater than that of R^j, i.e., $\delta_i > \delta_j$.
2. $\delta_i = \delta_j$, but the support of R^i is greater than that of R^j, i.e., $\pi_i > \pi_j$.

3. $\delta_i = \delta_j$ and $\pi_i = \pi_j$, but R^i contains a smaller number of nodes than R^j, i.e., $|T^i| < |T^j|$.

4. If none of the above is true, then T^i occurs lexicographically before T^j. We note that the lexicographic ordering of tree structures is based on a pre-order traversal of the nodes in the tree.

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 $\Delta = \{S \in \mathcal{D} \mid \nexists (R^i : T^i \Rightarrow c_j^i) \in \mathcal{R} \wedge (T_i \preceq S)\}$, be the set of examples from the training set $\mathcal{D}$ 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 1 the overall base cost-sensitive accuracy is given as

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

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 $w_i = \frac{|\mathcal{D}_i|}{|\mathcal{D}|}$ in $\frac{w_i|\Delta_i|}{|\mathcal{D}_i|}$, the term to be maximized, we get $\frac{|\Delta_i|}{|\mathcal{D}|}$. 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 $\overline{c}_i$. 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 $\overline{c}_i$, 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. Such ambiguous vases 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 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. The goal of 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 an example 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 $\mathcal{R}(S) = \emptyset$, when there are no matching rules. In this case, the class is predicted to be the default class, i.e., $S.c =$ *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 all classes (see Section 4), so we can compute the strength of T^k for the negative class $\overline{\mathcal{C}}_i$ ($T^k \stackrel{\delta^n}{\Rightarrow} \overline{\mathcal{C}}_i$). 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 Equations 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 $\delta_i^\mu = \frac{\sum_{j=1}^{r} \delta^j}{r}$. 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 confidence method for combining evidence, since it gave us the best results. We note that for average strength-based methods, 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. Several recent methods for tree mining have been proposed, such as FREQT [6], TreeMiner [21], and TreeFinder [19]. FREQT is based on an apriori-style, level-wise, candidate generation and pattern matching based counting approach. A similar approach is described in [20]. TreeFinder uses an Inductive Logic Programming approach, and 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. TreeMiner uses a novel vertical representation for fast subtree support counting. It is a complete method, and outperforms a level-wise method similar to FREQT. We thus chose TreeMiner as a basis for XMINER. Given a dataset $\mathcal{D}$ 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. XMINER extends TreeMiner to find 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 X be a k-subtree of a tree T. Let x_k refer to the last node of X. Each node in T has a well-defined *number*, i, according to its position in a depth-first (or pre-order) traversal of the tree. We use the notation n_i to refer to the ith node according to this numbering scheme ($i = 0 \ldots |T| - 1$). Let $T(n_l)$ refer to the subtree rooted at node n_l, and let n_r be the right-most leaf node in $T(n_l)$. The *scope* of node n_l is given as the interval $[l, r]$, i.e., the lower bound is the position (l) of node n_l, and the upper bound is the position (r) of node n_r. Figure 1 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).

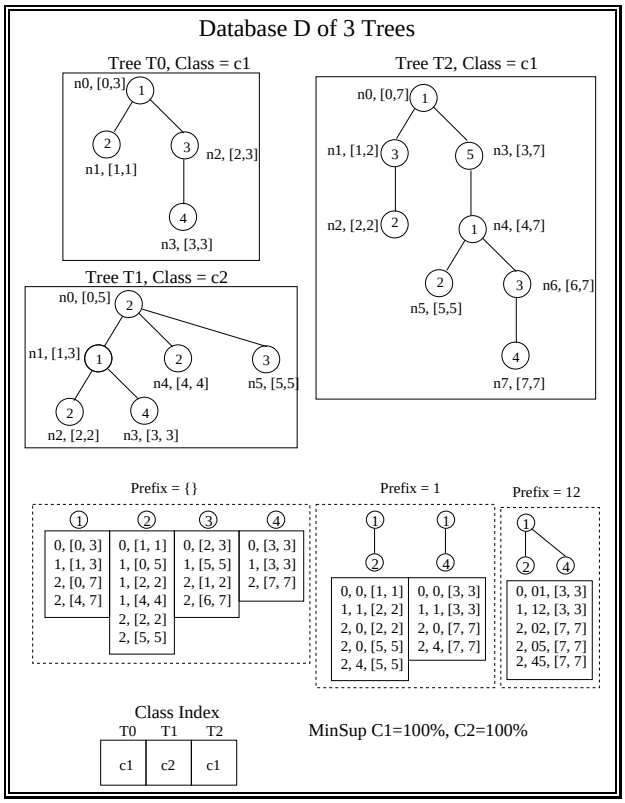

Figure 1: Tree Mining Example

Let $\mathcal{D}$ denote a database of trees (i.e., a forest), and let subtree $S \preceq T$ for some $T \in \mathcal{D}$. Each occurrence of S can be identified by its *match label*, which is given as the set of

matching positions (in T) for nodes in S. 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 S has a match label $\{t_{i_1}, t_{i_2}, \ldots t_{i_m}\}$,if and only if: 1) $L(s_k) = L(t_{i_k})$ for all $k = 1, \ldots m$ (where $L(n)$ is the label for node n, and 2) branch $b(s_j, s_k) \in S$ iff t_{i_j} is an ancestor of t_{i_k} in T. Condition 1) indicates that all node labels in S have a match in T, while 2) indicates that the tree topology of the matching nodes in T is the same as S. A match label is unique for each occurrence of S in T.

4.2 Prefix Group and Scope Lists

We say that two k-subtrees X, Y are in a *prefix equivalence group* iff they share a common prefix up to the $(k-1)$th node. Let P be prefix subtree of size $k-1$. We use the notation $[P]_{k-1}$ to refer to its group, which contain all the last items (k-th node) of trees that share P as their prefix. We use the notation $\mathcal{L}(X)$ to refer to the *scope-list* of X. Each element of the scope-list is a triple (t, s, m), where t is a tree id (tid) in which X occurs, s is the scope of x_k, and m is a match label for X. 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 1 shows the scope lists for the frequent single items (the minimum support is 100% for both classes). 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 2 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 1 shows the class index for the example database.

```
XMiner (D, π_i^min ∀i = 1 ··· k):
    [P]_0 = { frequent 1-subtrees for any class};
Enumerate-Xrules([P]_0);

Enumerate-Xrules([P]):
    for all elements x ∈ [P] do
        [P_x] = ∅;
        for all elements y ∈ [P] do
            R = x⊗y;
            L(R) = L(x) ∩_⊗ L(y);
            if for any R ∈ R, R is frequent for any class
                then [P_x] = [P_x] ∪ {R};
        Enumerate-Xrules([P_x]);
```

Figure 2: 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 $l_x <= l_y$ and $u_x >= u_y$. When we join 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 $(t_y, s_y, m_y) \in \mathcal{L}(y)$ and $(t_x, s_x, m_x) \in \mathcal{L}(x)$, 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 $(t_y, s_y, \{m_y \cup l_x\})$ 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 $(t_y, s_y, m_y) \in \mathcal{L}(y)$ and $(t_x, s_x, m_x) \in \mathcal{L}(x)$, 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 $(t_y, s_y,$-$\{m_y \cup l_x\})$ to the scope-list of y in $[P_x]$. We refer to this case as an *out-scope* test.

Figure 1 shows the process of scope-list joins for both in-scope and out-scope tests. To check if a new candidate is frequent, one can derive a per class count using the class index. For example, consider the tree in prefix group [1], with the branch $(1, 2)$. It appears in tids 0,1, and 2 (we count only once per tid). Using the class index we find that is occurs in classes c_2, c_1, c_2 respectively. Its support for class c_1 is 1 and for class c_2 is 2. It is thus 100% frequent locally in both classes.

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 were constructed using a clustering process [2]. Then, a nearest neighbor classifier was implemented on these sets of clusters. The CBA implementation was provided to us by its authors [16].

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) [18], 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.

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, while `other` class corresponds to all users visiting the CS department from any other domain. 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. We first construct 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. For each node in master tree $\mathcal{W}$, we assign probabilities of following its children nodes, including the option of backtracking to its parent, such that sum of all the probabilities is 1. 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.

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. Second, we generate a larger collection of trees, denoted T_D. Subset of trees from T_p are selected as training and testing pools, and T_D is also split into training and testing sets. If a tree $T \in T_D$ contains a tree from the signature pool then T has class c_1, otherwise it has class c_2. To control the effects of structure in the classification process, a fraction f_c, called *confusion ratio*, of trees that belong to one class (c_1) are added to other class (c_2), after flattening out. 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.

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 with two-way confusion. Finally DS4 is same as DS2, but with two-way addition only half the time ($f_c = 0.5$). The small data set DS5 was produced by a different synthetic XML document generator [3].

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
DS5.train	129	66	63	51.2	48.8
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
DS5.test	72	42	30	58.3	41.7

5.2 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.

Table 2 shows the weighted accuracy results for the three classifiers on different data sets. The table shows the accuracy for all three cost models. The best accuracy is highlighted in bold. 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 82.99% and 85.30% for the proportional model compared to IRC's accuracy from 73.76% to 77.64% and CBA's accuracy between 75.7% to 77.23%. Thus, the accuracy of XRules is about 8-10% higher (in absolute accuracy) than that of IRC and 5-10% higher than that of CBA 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 example consider the confusion matrix for CSLOG1-2 shown in Table 3, which shows the number of test cases, by class, that were correctly and incorrectly classified by the three classifiers (with proportional cost-model). CBA essentially labels each test case as `other`, thus it is ineffective for any

[3] provided by Elio Masciari (personal communication)

Table 2: Accuracy Results

DB	Classifier	Accuracy (%)		
		Proportional	Equal	Inverse
CSLOG1-2	XRules	**82.99**	**74.83**	**74.75**
	IRC	74.81	68.00	61.19
	CBA	77.23	50.00	22.77
CSLOG2-3	XRules	**84.61**	**75.70**	**76.19**
	IRC	77.64	70.85	64.06
	CBA	76.43	50.00	23.57
CSLOG12-3	XRules	**85.30**	**75.70**	**76.08**
	IRC	76.22	69.87	63.52
	CBA	76.43	50.00	23.57
CSLOG3-1	XRules	**83.81**	**74.09**	**76.08**
	IRC	73.76	67.73	61.70
	CBA	75.70	50.00	24.30
DS1	XRules	**71.87**	**74.02**	**76.31**
	CBA	51.35	54.24	57.12
DS2	XRules	**80.71**	**76.01**	**71.31**
	CBA	68.96	50.00	31.04
DS3	XRules	**61.63**	**61.63**	**61.63**
	CBA	50.00	50.00	50.00
DS4	XRules	**68.31**	**67.78**	**67.24**
	CBA	61.65	61.44	61.23
DS5	XRules	**88.63**	**88.81**	**88.99**
	CBA	50.00	48.10	46.19

Table 3: Confusion Matrix (CSLOG1-2)

	Predicted Class					
	XRules		IRC		CBA	
	`edu`	`other`	`edu`	`other`	`edu`	`other`
`edu`	870	817	1177	510	0	1687
`other`	396	5722	1356	4366	0	5722

cost-model other than the proportional one.

For the equal and inverse cost models, we find that XRules has higher accuracy than CBA and IRC since it explicitly incorporates cost. In the case of the CSLOG data sets, the accuracy of XRules is about 6% higher than that of IRC and 25% higher than that of CBA for the equal cost model. The situation is more pronounced for inverse model, where the accuracy of XRules is 14% higher than that of IRC and 50% higher than CBA!

On synthetic data sets, which do not have content (only structure), the IR classifier does not work. So we compared only XRules and CBA. The results are shown in Table 2. Once again, we found that CBA degenerated into a default classifier most of the time, labeling each test case with the majority class, though it did have a small number of rules (less than 17) relating to the two classes. As we can see for proportional cost model on DS1, DS3, and DS5, CBA fails to classify the test cases correctly, delivering an accuracy of only 50%, whereas the accuracy of XRules is 11-38% higher. On DS2 and DS4 CBA has some discrimination power, but the accuracy of XRules is still 12-17% higher. For the equal and inverse model, XRules outperforms CBA by up to 40%!

In summary, XRules gives consistently better 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 an IR based classifier which explicitly learns over the content, but only implicitly over the structural information in the XML documents. Therefore, the improved results of our structural classification process are especially significant.

5.3 Efficiency Results

Table 4 shows the number of frequent patterns (rules) mined by XMiner, and time for training and testing. 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 is needed to determine the default class in training, and to find the accuracy in testing. The running time can be improved by storing the rules in an appropriate index structure; currently XRules performs a linear search for matching rules.

Table 4: Number of Rules and Time

DB	Sup	Rules	Train Time (s)		Testing Time (s)
			XM	Total	
CSLOG1-2	0.3%	28911	5.5	469.7	425.8
CSLOG2-2	0.3%	19098	3.6	273.9	277.5
CSLOG12-3	0.35%	29028	7.4	858.8	447.9
CSLOG3-1	0.2%	31661	4.8	470.8	487.4
DS1	0.3%	883	7.2	147.6	152.4
DS2	0.3%	1589	5.9	210.2	231.1
DS3	0.3%	739	7.6	137.7	129.5
DS4	0.3%	900	5.8	140.1	125.5
DS5	15%	347	0.2	0.3	0.1

5.4 Choice of Rule Strength

We next study how the choice of strength measure affects the accuracy of XRules, as shown in Table 5. 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 2.4.1. 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 2.4.1. 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). On DS5, all measures give the same 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 many patterns with similar global supports but different local supports of rare classes, the likelihood/weighted confidence measure will usually provide better results.

5.5 Effect of Minimum Strength

Table 6 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 (82.87% for $\gamma^{\min} = 5$) and then starts to drop. The same effect

Table 5: Effect of Strength Measure

DB	Strength	Proportional	Equal	Inverse
CSLOG1	γ	81.09	74.73	**74.75**
	ρ	**82.99**	72.33	68.28
	ρ^w	81.43	**74.83**	74.61
CSLOG2	γ	82.34	**75.70**	**76.19**
	ρ	**84.61**	73.95	70.56
	ρ^w	82.62	75.69	75.89
CSLOG12	γ	81.22	74.09	**76.08**
	ρ	**85.30**	73.69	68.96
	ρ^w	82.54	**75.70**	75.76
CSLOG3	γ	81.22	**74.09**	**76.08**
	ρ	**83.81**	73.02	71.38
	ρ^w	81.35	74.07	75.90
DS1	γ	71.30	73.68	76.07
	ρ	**71.87**	71.69	71.51
	ρ^w	71.73	**74.02**	**76.31**
DS2	γ	55.06	61.65	68.23
	ρ	**80.71**	**76.01**	**71.31**
	ρ^w	60.45	65.06	69.68
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	58.29	58.88	59.47
	ρ	**68.31**	**67.78**	**67.24**
	ρ^w	61.23	61.62	62.00
DS5	γ	88.63	88.81	88.99
	ρ	88.63	88.81	88.99
	ρ^w	88.63	88.81	88.99

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 82.87% accuracy compared to 82.99% accuracy using confidence, in Table 5), and can improve the accuracy for the inverse model.

Table 6: Effect of Likelihood Ratio (CSLOG1-2)

$\gamma^{\min}$	Proportional	Equal	Inverse	#Rules	Time
1	81.09	**74.73**	74.75	28911	432.2
2	81.57	74.67	74.85	28553	428.9
3	82.38	73.83	75.46	25698	378.7
4	82.68	72.58	75.32	24190	355.9
5	**82.87**	72.14	75.72	17732	270.3
6	82.56	69.93	76.47	12006	180.5
7	82.42	69.37	76.17	10008	149.3
8	82.04	67.81	76.35	8936	134.6
9	81.53	66.27	76.50	8424	126.1
10	81.48	65.79	**76.54**	8199	123.5
15	80.73	62.62	76.25	7848	120.9
20	79.90	60.28	76.20	7636	121.9

6. CONCLUSIONS AND 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. Since the technique performs better than the CBA classifier, this indicates that the system relies on the classification information hidden in the structures for an effective rule generation process. 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.

7. REFERENCES

[1] C. C. Aggarwal. On Effective Classification of Strings with Wavelets. *SIGKDD*, 2002.

[2] C. Aggarwal, S. Gates, P. Yu. On the merits of using supervised clustering to build categorization systems. *SIGKDD*, 1999.

[3] R. Agrawal, R. Srikant. Fast Algorithms for Mining Association Rules. *VLDB Conference*, 1994.

[4] K. Alsabti, S. Ranka, V. Singh. CLOUDS: A Decision Tree Classifier for Large Datasets. *SIGKDD*, 1998.

[5] R. Andersen et al. Professional XML. *Wrox Press Ltd*, 2002.

[6] T. Asai, et al. Efficient substructure discovery from large semi-structured data. *2nd SIAM Int'l Conference on Data Mining*, 2002.

[7] W. W. Cohen. Fast Effective Rule Induction. *Int'l Conf. Machine Learning*, 1995.

[8] P. Domingos. MetaCost: A general method for making classifiers cost sensitive. *SIGKDD*, 1999.

[9] G. Dong, X. Zhang, L. Wong, J. Li. CAEP: Classification by Aggregating Emerging Patterns. *Int'l Conference on Discovery Science*, 1999.

[10] R. Duda, P. Hart. *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[11] J. Gehrke, V. Ganti, R. Ramakrishnan, W.-Y. Loh. BOAT: Optimistic Decision Tree Construction. *SIGMOD*, 1999.

[12] M. James. Classification Algorithms, *Wiley*, 1985.

[13] W. Li, J. Han, J. Pei. CMAR: Accurate and Efficient Classification Based on Multiple Class-Association Rules. *IEEE Int'l Conf. on Data Mining*, 2001.

[14] J. R. Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993.

[15] R. Rastogi, K. Shim. PUBLIC: A Decision Tree Classifier that Integrates Building and Pruning. *VLDB*, 1998.

[16] B. Liu, W. Hsu, Y. Ma. Integrating Classification and Association Rule Mining. *SIGKDD*, 1998.

[17] K. Nigam, A. K. McCallum, S. Thrum, T. Mitchell. Text Classification from labeled and unlabeled documents using EM. *Machine Learning*, 39(2/3):103-134, 2000.

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

[19] A. Termier, M-C. Rousset, M. Sebag. TreeFinder: a First Step towards XML Data Mining. *IEEE Int'l Conf. on Data Mining*, 2002.

[20] K. Wang, H.Q. Liu. Discovering Typical Structures of Documents: A Road Map Approach. *SIGIR*, 1998.

[21] M. J. Zaki. Efficiently Mining Frequent Trees in a Forest. *SIGKDD*, 2002.