Knowledge-Based Systems 29 (2012) 12–19

Contents lists available at SciVerse ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Jmax-pruning: A facility for the information theoretic pruning of modular classification rules

Frederic Stahl *, Max Bramer

University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE Portsmouth, UK

ARTICLE INFO

Article history:
Available online 10 August 2011

Keywords:
J-pruning
Jmax-pruning
Modular classification rule induction
Pre-pruning
Classification

ABSTRACT

The Prism family of algorithms induces modular classification rules in contrast to the Top Down Induction of Decision Trees (TDIDT) approach which induces classification rules in the intermediate form of a tree structure. Both approaches achieve a comparable classification accuracy. However in some cases Prism outperforms TDIDT. For both approaches pre-pruning facilities have been developed in order to prevent the induced classifiers from overfitting on noisy datasets, by cutting rule terms or whole rules or by truncating decision trees according to certain metrics. There have been many pre-pruning mechanisms developed for the TDIDT approach, but for the Prism family the only existing pre-pruning facility is *J-pruning*. *J-pruning* not only works on Prism algorithms but also on TDIDT. Although it has been shown that *J-pruning* produces good results, this work points out that *J-pruning* does not use its full potential. The original *J-pruning* facility is examined and the use of a new pre-pruning facility, called *Jmax-pruning*, is proposed and evaluated empirically. A possible pre-pruning facility for TDIDT based on *Jmax-pruning* is also discussed.

1. Introduction

The growing interest in the area of data mining has led to various developments for the induction of classification rules from large data samples in order to classify previously unseen data instances. Classification rule induction algorithms can be categorised in two different approaches: the Top Down Induction of Decision Trees (TDIDT) [1] also known as the 'divide and conquer' approach and the 'separate and conquer' approach. The 'divide and conquer' approach induces classification rules in the intermediate form of a decision tree, whereas the 'separate and conquer' approach directly induces '*IF… THEN…*' rules. The 'divide and conquer' approach can be traced back to the 1960s [2] and resulted in wide selection of classification systems such as C4.5 and C5.0. The 'separate and conquer' approach [3] can also be traced back to the 1960s. Its most notable member is the Prism family of algorithms as a direct competitor to the induction of decision trees.

The original Prism algorithm described in [4] identified the tree structure induced by 'separate and conquer' as the major handicap of decision trees which makes them vulnerable to overfitting, especially on noisy datasets. Prism has been shown to produce a similar classification accuracy compared with decision trees and in some cases, even outperforms decision trees. This is particularly the case, if the training data is noisy. There is also some recent interest in parallel versions of Prism algorithms, in order to make them scale better on large datasets. The framework described in [5], the Parallel Modular Classification Rule Inducer (PMCRI) allows to parallelise any member of the Prism family of algorithms.

Nevertheless also Prism, like any classification rule induction algorithm, is prone to overfitting on the training data. For decision trees there is a large variety of pruning algorithms that modify the classifier during or after the induction of the tree in order to make the tree more general and reduce unwanted overfitting [6]. For the Prism family of algorithms there is only one method described in the literature, and that is *J-pruning* [7,17]. *J-pruning* is based on the *J-measure* [8], an information theoretic measure for quantifying the information content of a rule. Also there is an extension of the PMCRI framework mentioned above, the J-PMCRI framework [9] that incorporates *J-pruning* into PMCRI, not only for quality reasons but also because *J-pruning* reduces the number of rules and rule terms induced and thus the runtime of Prism and PMCRI.

This paper examines the *J-pruning* method described in [7,17] and proposes a new pruning method *Jmax-pruning*, that aims to exploit the *J-measure* further and thus further to improve Prism's classification accuracy. The basic Prism approach is described in Section 2 and compared with decision trees, also some particular members of the Prism family are introduced briefly. *J-pruning* is described and discussed in Section 2.3 and *Jmax-pruning* as a new variation of *J-pruning* is introduced and discussed in Section 3.2 and evaluated in Section 4. The ongoing work is described in

* Corresponding author.
E-mail addresses: Frederic.Stahl@port.ac.uk (F. Stahl), Max.Bramer@port.ac.uk (M. Bramer).

0950-7051/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2011.06.016

Section 5 and discusses J-PrismTCS a version of Prism solely based on the *J-measure* and more importantly proposes the development of a version of *Jmax-pruning* for Decision Tree induction algorithms. Section 6 concludes the paper with a brief summary and discussion of the findings presented.

2. The Prism Family of Algorithms

As mentioned in Section 1, the representation of classification rules differs between the ‘divide and conquer’ and ‘separate and conquer’ approaches. The rule sets generated by the ‘divide and conquer’ approach are in the form of decision trees whereas rules generated by the ‘separate and conquer’ approach are modular. Modular rules do not necessarily fit into a decision tree and normally do not. The rule representation of decision trees is the main drawback of the ‘divide and conquer’ approach, for example rules such as:

IF A = 1 AND B = 1 THEN class = x
IF C = 1 AND D = 1 THEN class = x

cannot be represented in a tree structure as they have no attribute in common. Forcing these rules into a tree will require the introduction of additional rule terms that are logically redundant, and thus result in unnecessarily large and confusing trees [4]. This is also known as the replicated subtree problem [10]. Cendrowska illustrates the replicated subtree using the two example rules above in [4]. Cendrowska assumes that the attributes in the two rules above comprise three possible values and both rules predict class *x*, all remaining classes are labelled *y*. The simplest tree that can express the two rules is shown in Fig. 1 and the total rule set for predicting class *x* encoded in the tree is:

IF A = 1 *AND B* = 1 *THEN Class* = *x*
IF A = 1 *AND B* = 2 *AND C = 1 AND D = 1 THEN Class* = *x*
IF A = 1 *AND B* = 3 *AND C* = 1 *AND D* = 1 *THEN Class* = *x*
IF A = 2 *AND C* = 1 *AND D* = 1 *THEN Class* = *x*
IF A = 3 *AND C* = 1 *AND D = 1 THEN Class* = *x*

The fact that modular rules cause trees to grow needlessly complex makes them unsuitable for expert systems as they may require unnecessary expensive tests by the user [4].

‘Separate and conquer’ algorithms aim to avoid the replicated subtree problem by inducing directly sets of ’modular’ rules, avoiding unnecessarily redundant rule terms that are induced just for the representation in a tree structure. The basic ‘separate and conquer’ approach can be described as follows, where the statement

```
Rule_set rules = new Rule_set();
```

creates a new rule set:

```
Rule_Set rules = new Rule_set ( );
While Stopping Criterion not satisfied{
  Rule rule = Learn_Rule;
  Remove all data instances covered from Rule;
  rules.add (rule);
}
```

The *Learn_Rule* procedure generates the best rule for the current subset of the training data where best is defined by a particular heuristic that may vary from algorithm to algorithm. The stopping criterion is also dependent on the algorithm used. After inducing a rule, the rule is added to the rule set and all instances that are covered by the rule are deleted and a new rule is induced on the remaining training instances.

In Prism each rule is generated for a particular Target Class (TC). The heuristic Prism uses in order to specialise a rule is the probability with which the rule covers the TC in the current subset of the training data. The stopping criterion is fulfilled as soon as there are no training instances left that are associated with the TC.

Cendrowska’s original Prism algorithm selects one class as the TC at the beginning and induces all rules for that class. It then selects the next class as TC and resets the whole training data to its original size and induces all rules for the next TC. This is repeated until all classes have been selected as TC. Variations exist such as PrismTC [11] and PrismTCS (Target Class Smallest first) [7]. Both select the TC anew after each rule induced. PrismTC always uses the majority class and PrismTCS uses the minority class. Both variations introduce an order in which the rules are induced, where there is none in the basic Prism approach. However the predictive accuracy of PrismTC cannot compete with that of Prism and PrismTCS (personal communication). PrismTCS does not reset the dataset to its original size and thus is faster than Prism, which produces a high classification accuracy and also sets an order in which the rules should be applied to the test set.

The basic PrismTCS algorithm is outlined below where A_x is a possible attribute value pair and *D* is the training dataset. The statement

```
Rule_set rules = new Rule_set();
```

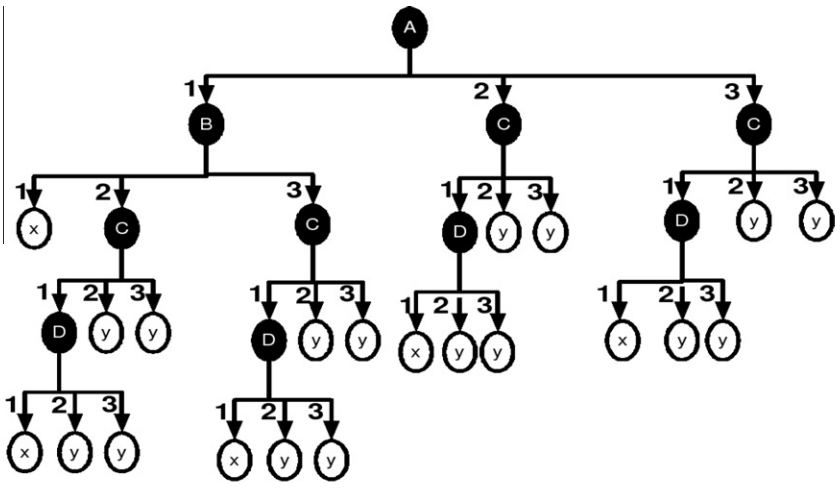

Fig. 1. Cendrowska’s replicated subtree example.