Induction of Modular Classification Rules: Using Jmax-pruning

Frederic Stahl and Max Bramer

Abstract The Prism family of algorithms induces modular classification rules which, in contrast to decision tree induction algorithms, do not necessarily fit together into a decision tree structure. Classifiers induced by Prism algorithms achieve a comparable accuracy compared with decision trees and in some cases even outperform decision trees. Both kinds of algorithms tend to overfit on large and noisy datasets and this has led to the development of pruning methods. Pruning methods use various metrics to truncate decision trees or to eliminate whole rules or single rule terms from a Prism rule set. For decision trees many pre-pruning and post-pruning methods exist, however for Prism algorithms only one pre-pruning method has been developed, *J-pruning*. Recent work with Prism algorithms examined *J-pruning* in the context of very large datasets and found that the current method does not use its full potential. This paper revisits the *J-pruning* method for the Prism family of algorithms and develops a new pruning method *Jmax-pruning*, discusses it in theoretical terms and evaluates it empirically.

1 Introduction

Classification rule induction from large training samples has a growing commercial importance and can be traced back to the 1960s [7]. Two general approaches to classification rule induction exist the 'separate and conquer' and the 'divide and conquer' approaches [14]. 'Divide and conquer' is better known as Top Down Induction of Decision Trees (TDIDT) [10] as it induces classification rules in the intermediate representation of a decision tree. The 'separate and conquer' approach can be traced back to the AQ learning system in the late 1960s [9]. Compared with TDIDT AQ generates a set of *IF..THEN* rules rather than decision trees, which is

---

Frederic Stahl, Max Bramer
University of Portsmouth, School of Computing, Buckingham Building, Lion Terrace, PO1 3HE Portsmouth, UK e-mail: {Frederic.Stahl; Max.Bramer}@port.ac.uk

useful for expert systems applications that are based on production rules. However most research concentrates on the TDIDT approach.

An important development of the ‘separate and conquer’ approach is the Prism family of algorithms [5, 2, 3]. Prism induces rules that are modular and that do not necessarily fit into a decision tree. Prism achieves a comparable classification accuracy compared with TDIDT and in some cases even outperforms TDIDT [2], especially if the training data is noisy. Recent research on the Prism family of algorithms comprises a framework that allows the parallelisation of any algorithm of the Prism family in order to make Prism algorithms scale better to large training data. The framework is called Parallel Modular Classification Rule Inducer (PMCRI) [13].

Like any classification rule induction algorithm Prism suffers from *overfitting* rules to the training data. Overfitting can result in a low predictive accuracy on previously unseen data instances (the test set) and a high number of induced rules and rule terms. There exist a variety of pruning methods for decision trees [6] that aim to reduce the unwanted *overfitting*, however there is only one published method of pruning rules of the Prism family, *J-pruning* [3]. *J-pruning* uses the *J-measure*, an information theoretic means to quantify the information content of a rule. *J-pruning pre-prunes* the rules during their induction. *J-pruning* has been integrated in PMCRI and not only improves the predictive accuracy but also lowers the number of rules and rule terms induced and thus also improves the computational efficiency of Prism algorithms [12].

This paper revisits the J-measure and *J-pruning*, develops *Jmax-pruning*, a variation of *J-pruning* and evaluates them empirically. Section 2 outlines the Prism Family of algorithms and compares them to TDIDT. Section 3 outlines *Jmax-pruning* followed by an empirical evaluation in Section 4. Ongoing work is discussed in Section 5 which comprises a new variation of the Prism approach and *Jmax-pruning* for TDIDT. Some concluding remarks can be found in Section 6.

2 The Prism Family of Algorithms

As mentioned in Section 1, rule representation 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 = 1

IF c = 1 AND d = 1 THEN class = 0

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 [5]. This is also known as the replicated subtree problem [14].

'Separate and conquer' algorithms induce directly sets of 'modular' rules like those above 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:

```
Rule_Set = [];
While Stopping Criterion not satisfied{
    Rule = Learn_Rule;
    Remove all data instances covered from 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 [4] and PrismTCS (Target Class Smallest first) [3]. 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:

```
Step 1: Find class i that has the fewest instances in the training
        set
Step 2: Calculate  for each Ax  p(class = i| Ax)
Step 3: Select the Ax with the maximum  p(class = i| Ax)
        and create a subset D' of D that comprises all instances
        that match the selected Ax.
Step 4: Repeat 2 to 3 for D' until D' only contains instances
        of classification i. The induced rule is then a
        conjunction of all selected Ax and i.
Step 5: Create a new D' that comprises all instances of D except
        those that are covered by all rules induced so far.
Step 6: IF D' is not empty repeat steps 1 to 5 until D' does not
        contain any instances of classification i.
```

We will concentrate here on the more popular PrismTCS approach but all techniques and methods outlined here can be applied to any member of the Prism family.

2.1 Dealing with Clashes

A *clash set* is a set of instances in a subset of the training set that are assigned to different classes but cannot be separated further. For example this is inevitable if two or more instances are identical except for their classification. Cendrowska's original Prism algorithm does not take into account that there may be clashes in the training data. However the Inducer software implementations of the Prism algorithms do take clashes into account [2, 4]. What happens in the case of a clash in Inducer is that all instances are treated as if they belong to the TC. [2] mentions that the best approach is to check if the TC is also the majority class. If it is then the rule currently being induced is taken otherwise the rule is discarded. If a clash is encountered and the majority class is not the TC then the rule is discarded and all instances in the clash set that match the TC are deleted. The reason for manipulating the clash set this way is that if the rule were discarded and the clash set kept then the same rule would be induced all over again and the same clash set would be encountered again.

2.2 Dealing with Continuous Attributes

Continuous attributes are not handled by Cendrowska's original Prism algorithm. One way to deal with continuous attributes is discretisation of the attribute values prior to the algorithm's application, for example applying ChiMerge [8] before the application of a Prism algorithm. Bramer's Inducer software [4] provides implementations of Prism algorithms that deal with continuous attributes, these are also used in all Prism implementations used in this work. Dealing with continuous attributes can be integrated in step two in the pseudo code above before the calculation of $p(class = i|A_x)$. If A_x is continuous then the training data is sorted for A_x. For example let A_x comprise the following values after sorting, -3, -4.2, 3.5, 5.5 and 10, then the data is scanned for these attribute values in either ascending or descending order. For each attribute value, for example 5.5, two tests $p(class = i \mid A_x < 5.5)$ and $p(class = i \mid A_x \geq 5.5)$ are conducted. The one with the largest conditional probability for all the values of the attribute is kept and compared with those conditional probabilities from the remaining attributes.

2.3 J-pruning

As mentioned in the introduction, classifiers are generally pruned to prevent them from overfitting. Pruning methods can be divided into two categories, *pre-pruning* and *post-pruning*. Post-pruning is applied to the classifier after it has been induced whereas pre-pruning is applied during the rule induction process. For Prism algorithms only one pruning method has been developed so far, *J-pruning* [3], a pre-pruning method based on the J-measure [11], a measure for the information content of a rule. *J-pruning* can also be applied to decision tree induction algorithms and has shown good results on both kinds of algorithms [3]. As also mentioned in the introduction, *J-pruning* has found recent popularity in [12], as it reduces the number of rules and rule terms induced considerably and thus increases the computational efficiency.

According to [11] the theoretical average information content of a rule of the form *IF Y = y THEN X = x* can be measured in bits and is denoted by *J(X,Y=y)*.

$$J(X;Y=y) = p(y) \cdot j(X;Y=y) \qquad (1)$$

As shown in equation (1) $J(X;Y=y)$ is essentially a product of $p(y)$, the probability with which the left hand side of the rule will occur, and $j(X;Y=y)$ which is called the j-measure (with a lower case j) and measures the goodness-of-fit of a rule. The j-measure, also called the *cross-entropy*, defined in equation (2):

$$j(X;Y=y) = p(x \mid y) \cdot log_2(\frac{p(x \mid y)}{p(x)}) + (1 - p(x \mid y)) \cdot log_2(\frac{(1 - p(x \mid y))}{(1 - p(x))}) \qquad (2)$$

For a more detailed description of the J-measure Smyth's paper [11] is recommended. Bramer's essential interpretation of the J-measure is that if a rule has a high J-value then it also is likely to have a high predictive accuracy [3]. Hence the J-value is used as an indicator of whether appending further rule terms is likely to improve a rule's predictive accuracy or lower it due to overfitting. The J-value of the rule may go up or down when appending rule terms, also it may go down and up again. However it is possible to calculate the maximum J-value that the rule with its current terms might maximally achieve if additional terms were added. This upper bound cannot of course be exceeded but its value is not necessarily achievable.

Bramer's basic *J-pruning* is applied to Prism by calculating the J-value of the rule before the induction of a new rule term and the J-value that the rule would have after a newly induced rule term is appended. If the J-value goes up then the rule term is appended. In the case where the J-value goes down, the rule term is not appended and a test is applied to determine whether the majority class of the instances that are covered by the rule is also the TC. If the majority class is the TC then the rule is truncated and kept and all instances in the current subset of the training set are treated as if all instances belong the TC. If the majority class is not the TC, then the rule is discarded and the clash resolution described in Section 2.1 is invoked.

3 Variation of J-pruning

In general there is very little work on pruning methods for the Prism family of algorithms. Bramer's *J-pruning* in the Inducer software seems to be the only pruning facility developed for Prism algorithms. This section critiques the initial *J-pruning* facility and outlines *Jmax-pruning*, a variation that makes further use of the J-measure.

3.1 Critique of J-pruning

Even though *J-pruning* described in Section 2.3 achieves good results regarding the overfitting of Prism, it does not seem to exploit the J-measure to its full potential. The reason is that even if the new rule term decreases the J-value, it is possible that the J-value increases again when adding further rule terms [12]. If the rule is truncated as soon as the J-value is decreased it may result in the opposite of overfitting, an over generalised rule with a lower predictive accuracy. The relatively good results for *J-pruning* achieved in [3] could be explained by the assumption that it does not happen very often that the J-value decreases and then increases again. However, how often this happens will be examined empirically in Section 4.

3.2 Jmax-pruning

According to [11], an upper bound for the J-measure for a rule can be calculated using equation (3):

$$J_{max} = p(y)\cdot\, max\{\ p(x\mid y)\cdot log_2(\frac{1}{p(x)}),\, (1-p(x\mid y))\cdot log_2(1\mid 1-p(x))\} \qquad (3)$$

If the actual J-value of the rule currently being generated term by term matches the maximum possible J-value (J_{max}) it is an absolute signal to stop the induction of further rule terms.

A concrete example is used to show how the J-values of a rule can develop. The example used a dataset extracted from the UCI repository, the soybean dataset [1]. Here we induce rules using our own implementation of PrismTCS without any *J-pruning*. The original dataset has been converted to a training set and a test set where the training set comprises 80% of the data instances. The 39th rule induced is:

IF (temp = norm) AND (same-lst-sev-yrs = whole-field) AND (crop-hist = same-lst-two-yrs) THEN CLASS = frog-eye-leaf-spot

This is a perfectly reasonable rule with a J-value of 0.00578. However looking at the development of the J-values after each rule term appended draws a different picture:

First Term

IF (temp = norm) THEN CLASS = frog-eye-leaf-spot

(J-value = 0.00113, J_{max} = 0.02315)

Here the rule has J-value of 0.00113 after the first rule term has been appended. The J-value for the complete rule (0.00578) is larger than the current J-value, which is to be expected as the rule is not fully specialised yet on the TC.

Second Term

IF (temp = norm) AND (same-lst-sev-yrs = whole-field) THEN CLASS = frog-eye-leaf-spot

(J-value = 0.00032, J_{max} = 0.01157)

Now the J-value is decreased to 0.00032 and J_{max} to 0.01157. Here *J-pruning* as described in Section 2.3 would stop inducing further rule terms, the finished rule would be

IF (temp = norm) THEN CLASS = frog-eye-leaf-spot

with a J-value of 0.00113. However looking at the value of J_{max}, after the second rule term has been appended, it can be seen that it is still higher than the previous J-value for appending the first rule term. Thus it is still possible that the J-value may increase again above the so far highest J-value of 0.00113. Inducing the next rule term leads to:

Third Term

IF (temp = norm) AND (same-lst-sev-yrs = whole-field) AND (crop-hist = same-lst-two-yrs) THEN CLASS = frog-eye-leaf-spot

(J-value = 0.00578, J_{max} = 0.00578)

In this case the rule was finished after the appending of the third rule term as it only covered examples of the TC. However, the interesting part is that the J-value increased again by appending the third rule term and is in fact the highest J-value obtained. Using Bramer's original method would have truncated the rule too early leading to an overall average information content of 0.00113 instead of 0.00578. The J-value and the *Jmax* value are rounded to five digits after the decimal point and appear identical but are actually slightly different. Looking at more digits the values are in fact for the J-value 0.005787394940853119 and for the J_{max} 0.005787395266794598. In this case no further rule terms can be added to the left-hand side of the rule as the current subset of the training set only contains instances of the TC, but if this were not the case it would still not be worthwhile to add additional terms as the J-value is so close to *Jmax*.

Overall this observation strongly suggests that pruning the rule as soon as the J-value decreases does not fully exploit the J-measure's potential. This work suggests that *J-pruning* could be improved by inducing the maximum number of possible rule terms until the current subset cannot be broken down further or the actual J-value is equal to or within a few percent of *Jmax*. As a rule is generated all the terms are labelled with the actual J-value of the rule after appending this particular rule term. The partial rule for which the largest rule J-value was calculated would then be identified and all rule terms appended afterwards truncated, with clash handling as described in Section 2.1 invoked for the truncated rule [12]. We call this new pre-pruning method *Jmax-pruning*.

4 Evaluation of Jmax-pruning

The datasets used have been retrieved from the UCI repository [1]. Each dataset is divided into a test set holding 20% of the instances and a training set holding the remaining 80% of the instances.

Table 1 shows the number of rules induced per training set and the achieved accuracy on the test set using PrismTCS with *J-pruning* as described in Section 2.3 and *Jmax-pruning* as proposed in Section 3.2.

What is also listed in Table 1 as 'J-value recovers', is the number of times the J-value decreased and eventually increased again when first fully expanding the rule and then pruning it using *Jmax-pruning*. Using the original *J-pruning* as described in Section 2.3 would not detect these J-value recoveries and lead to a rule with a lower J-value and thus lower information content than it could possibly achieve.

What can be seen is that in all cases *Jmax-pruning* performs either better than or produces the same accuracy as *J-pruning*. In fact seven times *Jmax-pruning* produced a better result than *J-pruning* and nine times it produced the same accuracy as *J-pruning*. Taking a closer look in the rule sets that have been produced in the nine cases for which the accuracies for both pruning methods are the same revealed that identical rule sets were produced in seven out of these nine cases. The two exceptions are the 'Car Evolution' and 'ecoli' datasets, however in these two exceptions the classification accuracy was the same using *Jmax-pruning* or *J-pruning*. In the cases where there are identical classifiers there were no J-value recoveries present. In Section 3.1 we stated that the good performance of *J-pruning* [3], despite its tendency to generalisation, can be explained by the fact that there are not many J-value recoveries in the datasets and thus the tendency to over generalisation is low. Looking into the last column of table 1 we can see the number of J-value recoveries. In seven cases there are none, thus there is no potential for over generalisation by using

Table 1 Comparison of *J-pruning* and *Jmax-pruning* on PrismTCS.

Dataset	Number of Rules	Accuracy (%)	Number of Rules	Accuracy (%)	J-value recovers
	J-Pruning		J-max Pruning		
monk1	4	79	12	86	4
monk3	3	98	3	98	0
vote	3	94	3	94	0
genetics	8	70	8	70	0
contact lenses	4	95	4	95	0
breast cancer	24	96	24	96	0
soybean	39	88	43	89	4
australian credit	20	89	20	89	0
diabetes	29	75	31	76	1
crx	18	83	18	83	0
segmentation	83	79	86	82	2
ecoli	23	78	26	78	3
Balance Scale	10	72	36	74	21
Car Evaluation	4	76	4	76	1
Contraceptive Method Choice	19	44	28	45	8
Optical Recognition of handwritten Digits	456	57	467	58	6

J-pruning and for the remaining datasets there is only a very small number of J-value recoveries with the exception of the ‘Balanced Scale’ dataset for which a 2% higher accuracy has been retrieved by using *Jmax-pruning* compared with *J-pruning*.

Loosely speaking, if there are no J-value recoveries present, then Prism algorithms with *Jmax-pruning* will produce identical classifiers to Prism algorithms with *J-pruning*. However, if there are J-value recoveries, it is likely that Prism algorithms with *Jmax-pruning* will produce classifiers that achieve a better accuracy than Prism algorithms with *J-pruning*.

What can also be read from Table 1 is the number of rules induced. In all cases in which both pruning methods produced the same accuracy, the classifiers and thus the number of rules were identical. However in the cases where the J-value recovered, then the number of rules induced with *Jmax-pruning* was larger than the number of rules induced with *J-pruning*. This can be explained by the fact that in the case of a J-value recovery the rule gets specialised further than with normal *J-pruning* by adding more rule terms while still avoiding overfitting. Adding more rule terms results in the rule covering fewer training instances from the current subset. This in turn results in that before the next iteration for the next rule less instances are deleted from the training set, which potentially generates more rules, assuming that the larger the number of training instances the more rules are generated.

5 Ongoing Work

5.1 *J-PrismTCS*

Annother possible variation of PrismTCS that is currently being implemented is a version that is solely based on the J-measure. Rule terms would be induced by generating all possible categorical and continuous rule terms and selecting the one that results in the highest J-value for the current rule instead of selecting the one with the largest conditional probability. Again the same stopping criterion as for standard PrismTCS could be used, which is that all instances of the current subset of the training set belong to the same class. We call this variation of PrismTCS, *J-PrismTCS*.

5.2 *Jmax-Pruning for TDIDT*

J-pruning has also been integrated into the TDIDT approach as a pre-pruning facility and achieved a higher classification accuracy than TDIDT without *J-pruning* [3]. Encouraged by the good results outlined in Section 4 which were achieved with *Jmax-pruning* in PrismTCS, we are currently developing a version of *Jmax-pruning* for TDIDT algorithms. The following pseudo code describes the basic TDIDT algorithm.

```
IF   All instances in the training set belong to the
     same class
THEN return value of this class
ELSE (a) Select attribute A to split on
     (b) Divide instances in the training set
         into subsets, one for each value of A.
     (c) Return a tree with a branch for each non
         empty subset, each branch having a decendent
         subtree or a class value produced by applying
         the algorithm recursively
```

The basic approach of *J-pruning* in TDIDT is to prune a branch in the tree as soon as a node is generated at which the J-value is less than at its parent node [3]. However performing the *J-pruning* is more complicated than for Prism algorithms as illustrated in the example below.

Figure 1 illustrates a possible tree which is used to explain *J-pruning* and *Jmax-pruning* for TDIDT. The nodes labelled with '?' are placeholders for possible subtrees. Now assuming that a depth first approach is used and the current node being expanded is node 'D'. In this case *J-pruning* would take the incomplete rule

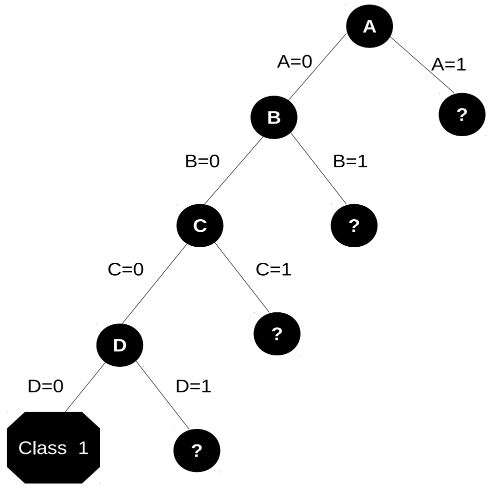

Fig. 1 Example of a decision tree for *J-pruning*.

(1) IF (A=0) AND (B=0) AND (C=0)...

the complete rule

(2) IF (A=0) AND (B=0) AND (C=0) AND (D=0) THEN class = 1

and the possible incomplete rule

(3) IF (A=0) AND (B=0) AND (C=0) AND (D=1)...

into account. Rule (2) is completed as all instances correspond to the same classification which is (class = 1). However instances covered by incomplete rule (3) correspond in this case to more than one classification.

J-pruning now compares the J-values of the incomplete rules (1) and (3). If the J-value of rule (3) is less than the J-value of rule (1) then rule (3) is completed by assigning it to the majority class of the corresponding instances. The complication is that the calculation of the J-value of a rule requires us to know its right-hand side. In the case of a complete (non-truncated) rule, such as rule (2), this is straightforward, but how can the J-value be calculated for an incomplete rule?

The method described by Bramer [3] is to imagine all possible alternative ways of completing the incomplete rule with right-hand sides class=1, class=2 etc., calculate the J-value of each such (completed) rule and take the largest of the values as the estimate of the J-value of the incomplete rule.

In a similar way to J-pruning for Prism algorithms, J-pruning for TDIDT in its current form does not necessarily exploit the full potential of the J-measure as again it is possible that if rule (3) were not truncated at the node labelled ‘?’ but expanded to complete the decision tree in the usual way the J-value for some or possibly all of the resulting complete branches might be at least as high as the J-value at node D.

Applying the idea of Jmax-pruning rather than J-pruning to TDIDT may increase the classification accuracy. This could be done by developing the complete decision tree and labelling each internal node with a J-value estimated as described above. Each branch (corresponding to a completed rule) can then be truncated at the node that gives the highest of the estimated J-values, in a similar way to the method described in Section 3.2, with each truncated rule assigned to the majority class for the corresponding set of instances.

This method appears attractive but there is a possible problem. Using the example from figure 1 and assuming that the estimated J-value of rule (1) is greater than the estimated J-value of rule (3) and that the majority class of the instances at node D is '1', then rule (1) would be truncated at node D and rule (3) would cease to exist, giving two completed rules in this subtree:

(1) IF (A=0) AND (B=0) AND (C=0) THEN class = 1

and

(2) IF (A=0) AND (B=0) AND (C=0) AND (D=0) THEN class = 1

Both rules are illustrated in figure 2. Rule (2) is now redundant. It is just a special case of rule (1), with the same classification, and can be discarded.

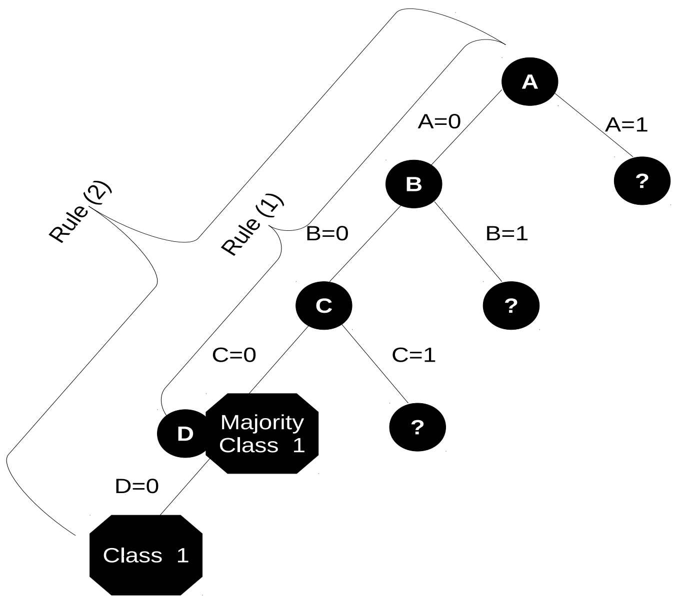

Fig. 2 Example of a decision tree with a redundant rule

Now suppose instead that in the above the majority class of the instances at node (1) were ‘2’ (rather than ‘1’). In this case a different picture would emerge, with rules (1) and (2) having different classifications. How likely this situation is to occur in practice and how it should best be handled if it does both remain to be determined.

6 Conclusions

Section 2 discussed the Prism family of algorithms as an alternative approach to TDIDT to the induction of classification rules. The Prism family of algorithms was highlighted and *J-pruning*, a pre-pruning facility for Prism algorithms based on the J-measure, which describes the information content of a rule, was introduced. Section 3 criticised *J-pruning* as it does not fully exploit the potential of the J-measure. The J-value of a rule may go up or down when rule terms are appended to the rule. *J-pruning* truncates a rule as soon as the J-value decreases even if it may recover (increase again). The proposed *Jmax-pruning* exploits the possibility of a J-value recovery and achieves in some cases, examined in Section 4, better results compared with *J-pruning*, but in every case examined *Jmax-pruning* achieved at least the same or a higher classification accuracy compared with *J-pruning*.

The ongoing work comprises the development of J-PrismTCS, a version of PrismTCS that is solely based on the J-measure, by using it also as a rule term selection metric as discussed in Section 5.1. Furthermore the ongoing work comprises the development of a TDIDT algorithm that incorporates *Jmax-pruning* as discussed in Section 5.2.

References

1. C L Blake and C J Merz. UCI repository of machine learning databases. Technical report, University of California, Irvine, Department of Information and Computer Sciences, 1998.
2. M A Bramer. Automatic induction of classification rules from examples using N-Prism. In *Research and Development in Intelligent Systems XVI*, pages 99–121, Cambridge, 2000. Springer-Verlag.
3. M A Bramer. An information-theoretic approach to the pre-pruning of classification rules. In B Neumann M Musen and R Studer, editors, *Intelligent Information Processing*, pages 201–212. Kluwer, 2002.
4. M A Bramer. Inducer: a public domain workbench for data mining. *International Journal of Systems Science*, 36(14):909–919, 2005.
5. J. Cendrowska. PRISM: an algorithm for inducing modular rules. *International Journal of Man-Machine Studies*, 27(4):349–370, 1987.
6. F Esposito, D Malerba, and G Semeraro. A comparative analysis of methods for pruning decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):476–491, 1997.
7. E B Hunt, P J Stone, and J Marin. *Experiments in induction*. Academic Press, New York, 1966.
8. R Kerber. Chimerge: Discretization of numeric attributes. In *AAAI*, pages 123–128, 1992.

9. R S Michalski. On the Quasi-Minimal solution of the general covering problem. In *Proceedings of the Fifth International Symposium on Information Processing*, pages 125–128, Bled, Yugoslavia, 1969.
10. R J Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann, 1993.
11. P. Smyth and R M Goodman. An information theoretic approach to rule induction from databases. *Transactions on Knowledge and Data Engineering*, 4(4):301–316, 1992.
12. F T Stahl. *Parallel Rule Induction*. PhD thesis, University of Portsmouth, 2009.
13. F T Stahl, M A Bramer, and M Adda. PMCRI: A parallel modular classification rule induction framework. In *MLDM*, pages 148–162. Springer, 2009.
14. I H Witten and F Eibe. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.