Towards a Computationally Efficient Approach to Modular Classification Rule Induction

Frederic Stahl, Max Bramer
School of Computing, University of Portsmouth, PO1 3HE, UK
{Frederic.Stahl; Max.Bramer}@port.ac.uk

Abstract

Induction of classification rules is one of the most important technologies in data mining. Most of the work in this field has concentrated on the Top Down Induction of Decision Trees (TDIDT) approach. However, alternative approaches have been developed such as the Prism algorithm for inducing modular rules. Prism often produces qualitatively better rules than TDIDT but suffers from higher computational requirements. We investigate approaches that have been developed to minimize the computational requirements of TDIDT, in order to find analogous approaches that could reduce the computational requirements of Prism.

1. Introduction

In research areas such as bioinformatics and cosmology researchers are confronted with huge amounts of data to which they wish to apply data mining algorithms such as association rule mining or artificial intelligence techniques such as pattern matching. Constructing a model from a dataset in the form of classification rules is often the method of choice to enable the classification of previously unseen data. However most of these algorithms are faced with the problem of scaling up to large datasets. The most common method of inducing classification rules is the Top Down Induction of Decision Trees (TDIDT) algorithm [1], which is the basis for well-known classification algorithms such as C5.0.

There have been several attempts to improve the scalability of TDIDT, notably the Scalable Parallelisable Induction of Decision Trees (SPRINT[1]) algorithm, which promises to scale up well without loss of any accuracy. The basic techniques used by SPRINT are pre-sorting of the data and parallelisation [2]. However, using the intermediate representation of a decision tree is not necessarily the best way to induce decision rules, especially if the data contains a lot of noise and clashes. The Prism algorithm presented by Cendrowska [3] is an alternative approach to developing classification rules which can often produce rules of a higher quality when there are clashes, noise or missing values in the dataset [4]. Unfortunately Prism has much higher computational requirements than TDIDT and thus in practice is seldom used, especially for large datasets. This work transfers the

[1] SPRINT stands for **S**calable **Pa**a**R**allelisable **IN**duction of decision **T**rees

approach used by SPRINT [2] in a modified form to the Prism algorithm. We will consider only the case where all attributes are continuous.

2. Inducing Decision Rules: The Prism Algorithm

Cendrowska [3] identified as a serious disadvantage of TDIDT that it generally constructs rules with substantial redundancies. Rulesets such as:

IF a = 1 AND b = 1 THEN CLASS = 1
IF c = 1 AND d = 1 THEN CLASS = 1

which have no common variable cannot be induced directly by TDIDT. Using TDIDT will frequently result in unnecessarily large and confusing decision trees, and that in turn causes unnecessary problems, for example if the values of redundant attributes are not known or are expensive to find out for an unseen instance, resulting in the decision rules needing to be pruned to remove redundant rule terms. Cendrowska presents Prism as an alternative to decision tree algorithms with the aim of generating rules with significantly fewer redundant terms compared with those derived from decision trees, from the beginning.

The basic Prism algorithm for a dataset D can be summarised as follows, assuming that there are n (>1) possible classes and that all attributes are continuous.

```
For each class i from 1 to n inclusive:

  (a) working dataset W = D
      delete all records that match the rules that have
      been derived so far for class i.
  (b) For each attribute A in W
      - sort data according to A
      - for each possible split value v of attribute A
        calculate the probability that the class is i
        for both subsets A < v and A ≥ v
  (c) Select the attribute that has the subset S with
      the overall highest probability
  (d) build a rule term describing S
  (e) W = S
  (f) Repeat b to d until the dataset contains only
      records of class i. The induced rule is then
      the conjunction of all the rule terms built at
      step d.
  (g) Repeat a to f until all records of class i have
      been removed.
```

The computational requirements of Prism are considerable. The basic algorithm comprises five nested loops and thus is not suitable for massive training sets. Nevertheless it can offer a higher quality of rules than TDIDT, for example it is less vulnerable to clashes. A clash occurs when a rule induction algorithm encounters a subset of the training set which has more than one classification but which cannot be processed further [4]. Prism has a bias towards leaving a test record unclassified rather then giving it a wrong classification. Furthermore Prism can produce rules with many fewer terms than the TDIDT algorithm, if there are

missing values in the training set [4]. Loosely speaking Prism produces qualitatively strong rules, but suffers from its high computational complexity.

3. Data Decoupling and Pre-Sorting to Improve Prism

In the SPRINT [2] version of TDIDT the data is initially decoupled into attribute lists. This enables each attribute to be processed independently of the others and thus overcomes memory constraints. The decoupling is performed by building data structures of the form <*record id, attribute value, class value*> for each attribute. A detailed description of the SPRINT algorithm is given in [2], on which the figures given in this section are based.

Figure 1 illustrates the creation of attribute lists from a sample data table, which is the part of SPRINT which we are using in an analogous way in our new version of Prism. Note that each list is sorted and each comprises a column with identifiers (ids) added so that data records split over several lists can be reconstructed.

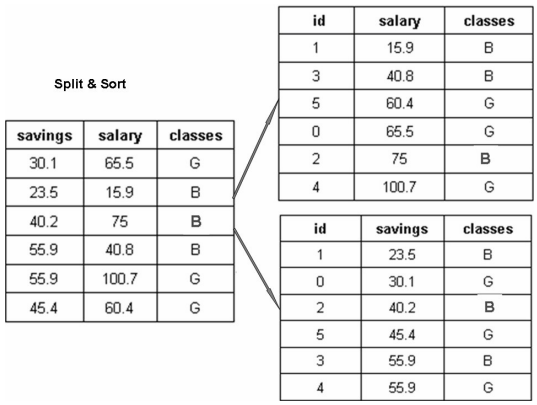

Figure 1 The building of sorted attribute lists

In contrast to SPRINT, Prism removes records that are not covered by a found rule term. When our Prism, which works with attribute lists, finds a rule term, it removes all records in the corresponding attribute list that are not covered by that term. By using the ids of the deleted list records, Prism further deletes all records in its other attribute lists that match these ids.

Figure 2 shows how data records are removed from the attribute lists. Assume we are generating rules for the class value G, Prism would find the highest probability of the class being G for the term *salary* ≥ *60.4*. In the salary list the records with ids 1 and 3 are the only ones not covered by this term, thus the records with ids 1 and 3 are removed from all the attribute lists. The shaded records in both lists are those that are covered by this term.

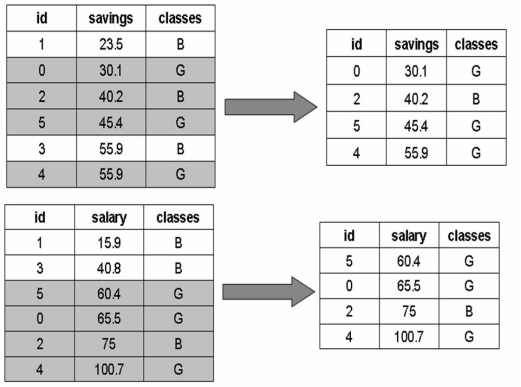

Figure 2 Removing records from the attribute lists

What is important here is that the resulting attribute lists shown on the right-hand side of Figure 2 are still sorted. Thus we have eliminated the multiple sorting which formed part of step (b) in the original algorithm description in Section 2. This removal of one of the five nested loops in Prism makes its runtimes considerably faster. A further effect of the usage of attribute lists is that memory constraints are removed. Our version of Prism now only needs to hold one attribute list in memory at any stage of the algorithm instead of the complete dataset. This allows us to process datasets that are too large to fit wholly into memory, which is clearly desirable but imposes the requirement for many I/O operations in order to buffer unneeded attribute lists onto the hard disc.

4. Evaluation of Improved Prism

We used a test dataset containing 100 attributes and 1000 records. The data consists of continuous double precision values with three classes. We implemented three algorithms:

(1) Prism without pre-sorting.
(2) Prism using attribute lists and pre-sorting of attributes.
(3) Prism using attribute lists, pre-sorting of attributes and buffering of unneeded attribute lists to the hard disc.

We want to show by comparing algorithms (1) and (2) that pre-sorting in Prism has a positive impact on efficiency and thus makes it scale better on bigger datasets. Although all the attribute lists were held in memory simultaneously for experiments (1) and (2), we conducted a further experiment (3) to investigate the overhead imposed by I/O operations in cases where we need to deal with memory constraints that make it necessary to hold only one attribute in memory at any time. We ran each algorithm on the training set 100 times and recorded the runtimes. We also checked that all the results were reproducible.

Table 1 shows the average runtimes of each implemented algorithm on the dataset described above. The average runtime values are based on 100 runs. In each case the first run was not taken into account as it is influenced by the operating system which caches the algorithm. Comparing Prism with and without pre-sorting we can see that the pre-sorting approach speeds up Prism by about a factor of 1.8. We can

also see the substantial overhead imposed by the buffering of unneeded attribute lists onto the hard disc.

Table 1 Average runtimes of the test algorithms on the test dataset

Algorithm	Average Runtime in ms
Prism without pre-sorting	230995
Prism with pre-sorting	128189
Prism with I/O and pre-sorting	1757897

5. Ongoing Work

5.1 Ongoing Work on Serial Prism

Pre-sorting has a high potential to improve the complexity of Prism, but (as for SPRINT) it involves many time consuming I/O operations if the attribute lists do not fit into the memory. Thus one part of our present work lies in the reduction of the size of the attribute lists. Here we describe how we can reduce the size of attribute lists in the case of continuous attributes. The size of the data that needs to be held in memory by the conventional Prism is (8*n+1)*m bytes, where n is the number of attributes and m is the number of records. Eight bytes is the amount of storage needed for an attribute value (assuming double precision values) and one byte corresponds to the size of a class value assuming a character. The storage needed to hold all the attribute lists in memory is (8+4+1)*n*m bytes. The eight bytes are the size of an attribute value; the four bytes correspond to a record id and the one byte to a class value. However we could reduce these memory requirements simply by working without the attribute value in the attribute lists. We only need the complete attribute list structure <*record id, attribute value, class value*> to sort the attribute list according to its values. After sorting, the Prism algorithm could work with only the distribution of the class values in the attribute list. <*record id, class value*> for each attribute would need to be held in memory. Thus the memory requirement after sorting could be reduced to (4+1)*n*m bytes, which would be considerably less than the memory required by the conventional Prism. (The memory requirement of categorical attributes can be reduced in a similar way.) A version of Prism that works with this reduced list approach is currently in development.

5.2 Ongoing Work on Parallel Prism

We are currently focussing on data parallelisation on a *shared nothing* machine. In a shared nothing system, in contrast to a shared memory system, each processor has its own memory assigned to it. From the hardware point of view the shared nothing system could be realised by a cluster of PCs, thus it would be the cheapest and most flexible hardware configuration. It is especially flexible as it is easy to upgrade by simply adding more PCs into the cluster. Data parallelism in Prism could be achieved by having n processors that work on portions of the training set and consequently generate a global set of classification rules. The data parallelism that SPRINT uses is called *attribute data parallelism*. The basic approach is to divide

the attribute lists equally among different processors. Thus each processor is responsible for 1/n attributes [5]. This approach could be used analogously by Prism but we expect it to cause work balancing problems after the first iteration as Prism removes parts of the attribute lists during its iterations. Thus it could easily happen that part attribute lists are completely removed on some processors and not on others. We are currently developing a further distributed workload balancing mechanism by assigning not only one but two or more chunks of each attribute list which are not in ascending order to a processor. Thus we would ensure that the records removed by Prism are less concentrated on a certain processor.

6. Conclusions

This paper presents ongoing work and first results in the attempt to speed up a classification rule induction algorithm that is an alternative to decision trees. It points out some weaknesses of the traditional TDIDT algorithm and the computational inefficiency of the alternative Prism algorithm. We propose (a) pre-sorting of the data and (b) parallelisation as methods to improve Prism's computational efficiency. With regard to (a) we mapped the attribute list structure used in the SPRINT algorithm onto Prism. In comparison with the runtimes of the conventional Prism algorithm the runtimes of Prism with pre-sorting are about a factor of 1.8 faster. However, the data in the form of attribute lists is much bigger in size then the raw data and for large datasets this can result in massive I/O operations between the hard disc and the memory. Our ongoing work is on a new attribute list structure that is smaller in size and thus promises to reduce the data volume of the I/O operations. With regard to (b) we introduced a more advanced attribute list parallelisation strategy than those of SPRINT in order to overcome work balancing problems caused by the nature of Prism. Future work will comprise with regard to (a) the implementation of Prism with a smaller attribute list structure and with regard to (b) the implementation of data parallel Prism with a novel work balancing structure.

References

1. Quinlan, J.R., Induction of decision trees. Machine learning, 1986. 1(1): p. 81-106.
2. Shafer, J.C., R. Agrawal, and M. Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. in 22th International Conference on Very Large Data Bases. 1996.
3. Cendrowska, J., PRISM: an Algorithm for Inducing Modular Rules. International Journal of Man-Machine Studies, 1987. 27: p. 349-370.
4. Bramer, M. Automatic Induction of Classification Rules from Examples Using N-Prism. in Research and Development in Intelligent Systems XVI. 2000: Springer Verlag.
5. Zaki, M.J., C.-T. Ho, and R. Agrawl. Parallel Classification for Data Mining on Shared Memory Multiprocessors. in 15th International Conference on Data Engineering. 1999.