P-Prism: A Computationally Efficient Approach to Scaling up Classification Rule Induction

Frederic T. Stahl, Max A. Bramer, Mo Adda

Abstract Top Down Induction of Decision Trees (TDIDT) is the most commonly used method of constructing a model from a dataset in the form of classification rules to classify previously unseen data. Alternative algorithms have been developed such as the Prism algorithm. Prism constructs modular rules which produce qualitatively better rules than rules induced by TDIDT. However, along with the increasing size of databases, many existing rule learning algorithms have proved to be computational expensive on large datasets. To tackle the problem of scalability, parallel classification rule induction algorithms have been introduced. As TDIDT is the most popular classifier, even though there are strongly competitive alternative algorithms, most parallel approaches to inducing classification rules are based on TDIDT. In this paper we describe work on a distributed classifier that induces classification rules in a parallel manner based on Prism.

1 Introduction

Scaling up data mining algorithms to massive datasets has never been more topical. That is because of the fast and continuous increase in the number and size of databases. For example in the area of Molecular Dynamics (MD), simulations are conducted which describe the unfolding and folding of proteins. These simulations generate massive amounts of data which researchers are just starting to manage to store [7]. For example one single experiment can generate datasets of

Dipl.-Ing.(FH)Frederic T. Stahl
University of Portsmouth, Lion Terrace, PO1 3HE Portsmouth e-mail: Frederic.Stahl@port.ac.uk

Prof. Max A. Bramer
University of Portsmouth, Lion Terrace, PO1 3HE Portsmouth e-mail: Max.Bramer@port.ac.uk

Dr. Mo Adda
University of Portsmouth, Lion Terrace, PO1 3HE Portsmouth e-mail: Mo.Adda@port.ac.uk

100s of gigabytes[6]. Researchers in the MD community wish to apply data mining algorithms on MD experimental data such as pattern detection, clustering or classification[4]. However, most data mining algorithms do not scale well on such massive datasets and thus researchers are forced to sample the data to which they want to apply their data mining algorithms. Catlett's work [2] shows that sampling of data results in a loss of accuracy in the data mining result. However, Catlett conducted his experiments 16 years ago and referred to data samples that were much smaller than those nowadays. Frey and Fisher [8] showed that the rate in increase of accuracy slows down with the increase of the sample size.

However, scaling up is also an issue in applications that are concerned with the discovery of knowledge from large databases rather than predictive modelling. For instance researchers are interested in discovering knowledge from gene expression datasets, for example concerning knowledge about the influence of drugs on the gene expression levels of cancer patients. A drug might be designed to suppress tumour promoting genes, so called oncogenes. In some cases the same drug might also suppress genes that are not directly related to the tumour and thus cause adverse effects which might be lethal in rare cases. If we would sample here, we might lose data that may lead to the detection of rules that might identify a risk in applying a certain drug. Furthermore not only the number of examples but also the number of attributes which describe each example contributes to the size of the dataset [5]. For example gene expression datasets often comprise thousands or even tens of thousands of genes which represent attributes in a relational data table.

We present work on a parallel data distributed classifier based on the Prism [3] algorithm. We expect to be able to induce qualitatively good rules with a high accuracy and a sufficient scale up on massive datasets, such as gene expression datasets.

2 Inducing Modular Classification Rules Using Prism

The Prism classification rule induction algorithm promises to induce qualitatively better rules compared with the traditional TDIDT algorithm. According to Cendrowska, that is because Prism induces modular rules that have fewer redundancies compared with TDIDT [3]. Rule sets such as:

IF a = 1 AND b = 1 THEN CLASS = 1
IF c = 1 AND d = 1 THEN CLASS = 2

which have no common variable cannot be induced directly by TDIDT [3]. Using TDIDT would produce unnecessarily large and confusing decision trees. Cendrowska presents the Prism algorithm [3] as an alternative to decision trees. We implemented a version of Prism that works on continuous datasets like gene expression data [11]. The basic Prism algorithm, for continuous data only, can be summarised as shown in figure 1, assuming that there are $n(>1)$possible classes. The aim is to generate rules with significantly fewer redundant terms than those derived from decision trees. Compared with decision trees Prism[1]:

- Is less vulnerable to clashes
- Has a bias towards leaving a test record unclassified rather than giving it a wrong classification
- Often produces many fewer terms than the TDIDT algorithm if there are missing values in the training set.

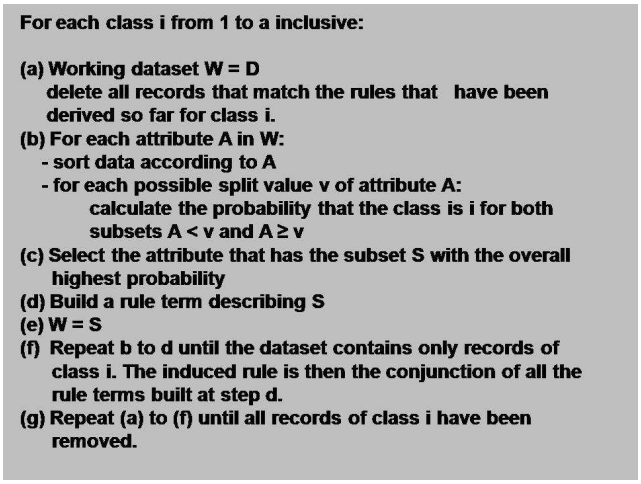

Fig. 1 The basic Prism algorithm for continuous data comprises five nested loops. The innermost loop involves sorting of the data for every continuous attribute.

However as shown in the algorithm in figure 1, the computational requirements of Prism are considerable, as the algorithm comprises five nested loops. The innermost loop involves sorting (contained in step b) the data for every continuous attribute [11]. Loosely speaking Prism produces qualitatively strong rules but suffers from its high computational complexity.

We have removed the innermost loop by pre-sorting the data once at the beginning. We did that by representing the data in the form of sorted attribute lists. Building of attribute lists is performed by decoupling the data into data structures of the form

$$\langle record\,id, attribute\,value, class\,value \rangle$$

for each attribute. Attribute lists were first introduced and successfully used in the SPRINT (Scalable PaRallelizable INduction of decision Trees) project for the parallelisation of TDIDT[10]. The use of sorted attribute lists enabled us to keep the data sorted during the whole duration of the Prism algorithm. By doing so we achieved a speedup factor of 1.8 [11].

The left-hand side of figure 2 illustrates the building of attribute lists. Note that all lists are sorted and all lists comprise a column with identifiers (id) added so that data records split over several lists can be reconstructed. As Prism removes attribute lists that are not covered by the previously induced rule term, our classifier

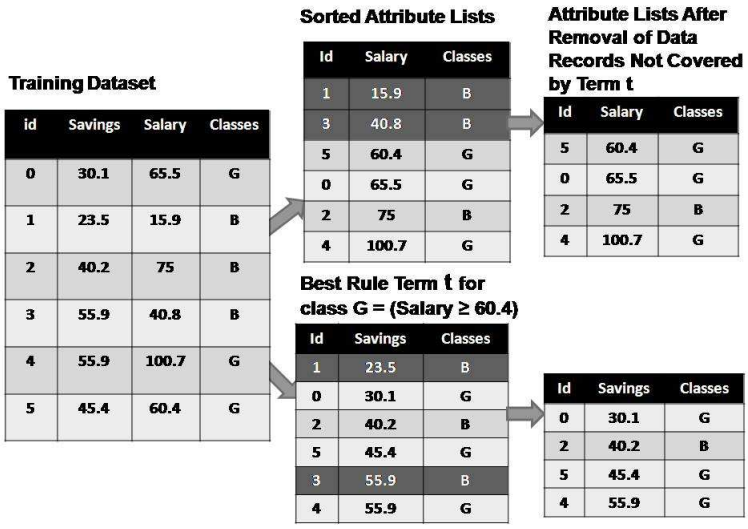

Fig. 2 The left hand side shows how sorted attribute lists are built and the right hand side shows how list records, in this case records with the ids 1 and 3, are removed in Prism.

needs to remove list records in an analogous way. For example if Prism finds a rule term $(salary \geq 60.4)$ for class G then Prism would remove the list records with the id values 1 and 3 as they are not covered by this rule. Note that the resulting list records are still sorted. This fact eliminates multiple sorting of attribute lists. The use of attribute lists in Prism enables us to parallelise the algorithm in a shared nothing environment, where each CPU has its own private memory, by data distribution [11].

3 Speeding up Prism by Parallelisation via a Distributed Blackboard System

A blackboard system is a software architecture that simulates a group of experts in front of a blackboard which have expertise in different areas. These experts communicate by reading new information from the blackboard, deriving new knowledge from it and writing this new information again on to the blackboard, thus making it accessible to the other experts. In the software architecture the blackboard is based on a server/client model. The server functions as a blackboard and the clients as experts. We are using an implementation of a distributed blackboard system developed by the Nottingham Trent University [9]. In a similar way to a shared memory version of SPRINT [13] we want to parallelise Prism by distributing 1/k chunks of each attribute list to k different expert machines. We want to synchronise the algorithm then by using the distributed blackboard system. Thus each expert machine will hold

a different part of the data and derive new knowledge from it in the form of a locally best rule term. Then each expert machine will exchange quality information about all locally best rule terms via the blackboard with the other expert machines.

4 Parallel Prism: Basic Architecture And Algorithm

As described in the previous section, the first step is to build attribute lists and distribute them to all expert machines. In the context of Parallel Prism (P-Prism), we refer to the expert machines as Worker Machines as the purpose of parallelising Prism is to split the workload, determined by the size of the training data, over k CPUs.

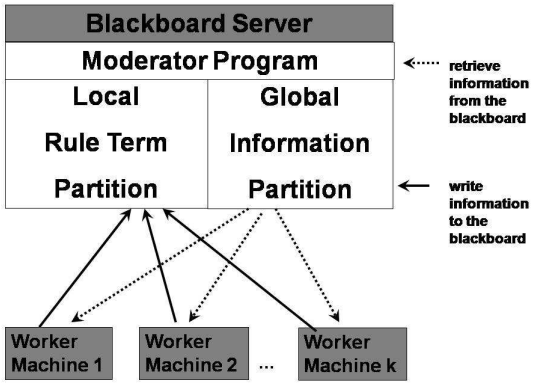

Fig. 3 Architecture of P-Prism using a blackboard server comprising two partitions, a partition for submitting rule terms to the blackboard (Local Rule Term Partition) and one to advertise global information (global information partition) to the worker machines. The moderator program on the blackboard derives the global information.

Figure 3 illustrates the basic architecture of P-Prism using a blackboard server which comprises two partitions, a partition for submitting rule terms to the blackboard, the "Local Rule Term Partition" and one to advertise Global information to the worker machines, the "Global Information Partition". Figure 4 depicts the basic P-Prism algorithm. Each worker machine M induces independently a rule term t_M for class i which is the best rule term to describe i on the local data on M. The quality of t_M is measured in the form of the probability P_M with which t_M covers class i on the local data. Each M submits t_M plus its associated P_M to the "Local Rule Term Partition" on the blackboard. The moderator program on the blackboard collects all t_Ms with their associated P_Ms and searches out the globally best rule term, which is the one with the highest P_M. The moderator program also provides global informa-

tion to the worker machines by writing it on to the ”Global Information Partition”. The global information comprises the globally best rule term or identifiers for the data covered by this rule term. Loosely speaking, global information informs the worker machines about the global state of the algorithm and thus how they shall proceed, e.g. deriving a further rule term or starting a new rule.

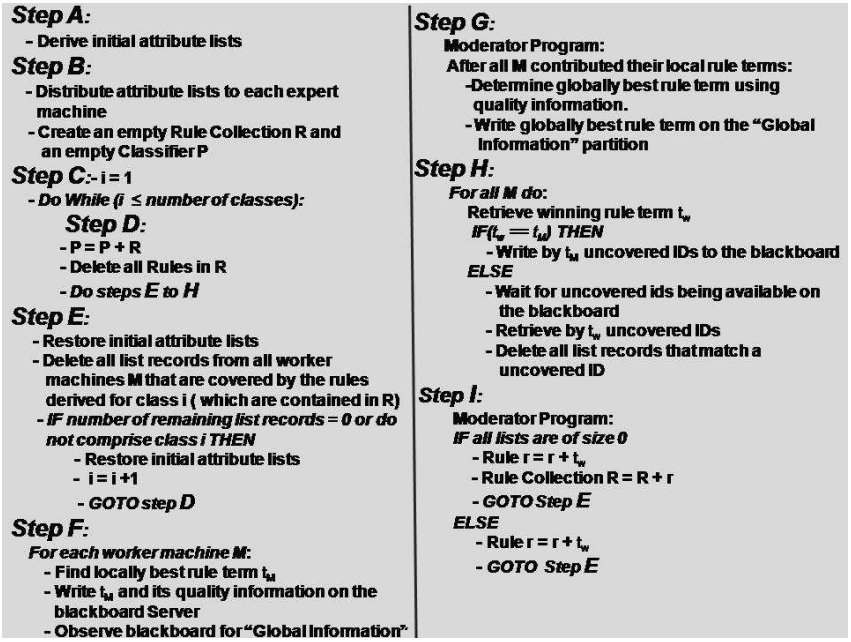

Fig. 4 Outline of the basic structure of the P-Prism algorithm. The data distribution takes place in step B and the parallelisation in step D.

Figure 4 outlines the rough structure of the proposed P-Prism algorithm. The data distribution takes place in step B by distributing the attribute lists. The parallelisation takes place in steps F to I as here every worker machine derives its local rule term and waits for the global information to become available on the blackboard.

5 Ongoing Work

So far we have set up a local area network with 4 worker machines and one blackboard server which is configured as described in section 4. The moderator program has been implemented and is fully functional. The worker machines simulate rule term induction for a complete rule in order to test the moderator program's functionality on the blackboard server. The next step is to fully implement the worker machines in order to test the computational performance of P-Prism.

5.1 Reduction of the Data Volume Using Attribute Lists

Shafer claims in his paper [10] that attribute lists could be used to buffer data to the hard disc in order to overcome memory constraints. However, buffering of attribute lists involves many I/O operations. As the data in attribute lists is even larger than the raw data, we expect a considerable slowdown of the runtime of Prism if we use buffering of attribute lists. Thus we are working on a modified version of the attribute list. As Prism mainly needs the class value distribution in a sorted attribute list, we want to reduce memory usage by building class distribution lists instead of attribute lists. Class distribution lists have the following structure:

$$\langle record\,id, class\,value \rangle$$

The class distribution list is built out of a sorted attribute list by deleting the attribute value column, thus the ids and class values in the class distribution list are sorted according to the attribute values. The total size of these lists is less than that of the raw data and even less than that of the data in the form of attribute lists. The rules induced using class distribution lists will lead to rule terms labelled with record ids instead of the actual attribute values. After all rules are induced, the record ids can easily be replaced by the actual attribute values.

The amount of memory (S) needed for Prism working on the raw data can be described by the formula $S = (8 * n + 1) * m$ bytes, where n is the number of attributes and m is the number of data records. We assume that eight bytes is the amount of memory needed to store an attribute value (assuming double precision values) and one byte to store a class value assuming a character representation. These assumptions would perfectly apply to gene expression data as a gene expression value is a double precision value. The storage needed by Prism to hold all the attribute lists in memory can be described analogously by the formula $S = (8 + 4 + 1) * n * m$ bytes. Again, the eight bytes represent an attribute value and the one byte a class value. The four byte value corresponds to an integer value for a record id in the attribute list. Representing the training data with the class distribution list structure instead of the attribute list structure eliminates the eight byte attribute values and thus only requires a memory usage of $S = (4 + 1) * n * m$ bytes [11].

However, attribute lists without the actual attribute value cannot be used exactly as stated above. We need to find a way to deal with repeated attribute values. Figure 5 illustrates the problem of repeated attribute values. The attribute list on the left hand side would find $(X \leq 2.1)$ as the rule term for the attribute X regarding class C with a covering probability of 0.67. The class distribution list on the right hand side in figure 5 represents our class distribution list with only the ids and the class values. Using only the class distribution without incorporating information about repeated attribute values for finding the best rule term would lead to a rule term of the form $(X > id0)$ for class C with a covering probability of 1. But the value of X at id 0 is 8.7, which leads to the actual rule term $(X > 8.7)$ which has a covering probability of only 0.5 as data records with ids 3 and 0 are also covered by that term. Thus we need to mark repeated attribute values in our class distribution list structure.

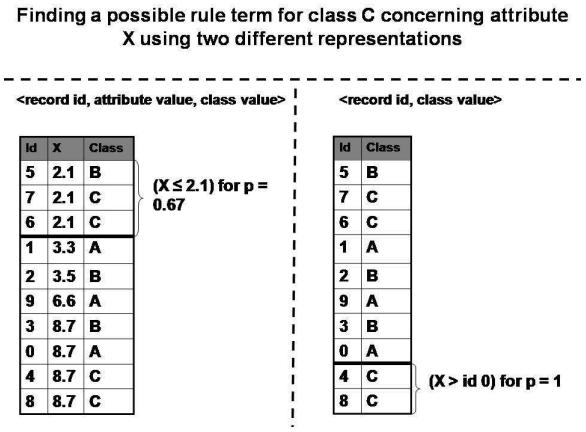

Fig. 5 Finding a possible rule term in a class distribution list without having the attribute values can lead to low wrong rule terms

One possible way to mark repeated attribute values in the class distribution list is to add another column "indicator" which is a flag that indicates repeated values, e.g. we could use an integer that is 0 if the list record corresponds to a repeated attribute value or 1 if not. Thus the class distribution list structure would have to be altered to $\langle indicator, record\,id, class\,value \rangle$. This also leads to an altered formula for the memory usage which is $S = (4+1+1) * n * m$. The additional byte here corresponds to the added indicator. The resulting S would still be smaller than those for the raw data and the traditional attribute list structure. Concerning memory usage a better way is to use signed integers for the record id in the class distribution list structure. Positive record ids can be used for non-repeated attribute values and negative ids for repeated ones. The formula for the memory usage here would remain the same, but the disadvantage of using signed integers is that we could only represent 2^{31} data records instead of 2^{32} for unsigned integers. Another way to represent repeated attribute values without using additional memory or signed integers is using upper case and lower case characters for the class values in order to indicate repeated attribute list values. For example using lower case letters for non-repeated values and upper case letters for repeated values.

Table 1 shows the actual memory usage of Prism working with raw data, attribute lists and our class distribution list calculated using signed integers or upper case and lower case class values and using $S = (4+1) * n * m$ bytes for the memory usage of the class distribution list. The datasets are gene expression datasets concerning several diseases which can be retrieved from http://sdmc.lit.org.sg/GEDatasets/ except the SAGE-tag and Affymetix dataset. They can be found at the Gene Expression Omnibus (GEO) [12]. We clearly see that using attribute lists greatly increases the memory required to store the training data. We also see that the class distribution list outperforms the representation of data using both raw data and attribute lists in relation to memory requirements.

Table 1 Examples of the memory usage of several gene expression datasets in Prism using raw data $S1 = (8 * n + 1) * m$; using the attribute list structure $S2 = (8 + 4 + 1) * n * m$ and using our proposed class distribution list structure $S3 = (4 + 1) * n * m$. The datasets are gene expression data, thus the number of attributes is determined by the number of genes. All values for memory usage are stated in megabytes.

Dataset	Genes(n)	Examples(m)	S1	S2	S3
ALL/AML Leukaemia	7129	48	2.74	4.45	1.71
Breast cancer outcome	24481	78	15.28	24.82	9.55
CNS embryonal tumour	7129	60	3.42	5.56	2.14
Colon tumour	7129	62	3.42	5.75	2.21
Lung cancer	12533	32	3.21	5.21	2.01
Prostate cancer	12600	102	10.28	16.71	6.43
Prostate cancer outcome	12600	21	2.12	3.44	1.32
Affymetix	12332	1640	161.80	262.92	101.12
SAGE-tag	153204	243	297.83	483.97	186.14

5.2 Synchronisation

Further work will be conducted on the synchronisation of the distributed classifier. In particular several worker machines will have to wait for further information on the blackboard after they write their rule term plus its quality in the form of the covering probability on the backboard. This idle time could be used, for example, to induce locally terms for a different class value.

6 Conclusions

This paper describes work on scaling up classification rule induction on massive data sets. We first discussed why classification rule induction needs to be scaled up in order to be applicable to massive data sets and focused on a particular classification rule induction algorithm. The algorithm we focused on is the Prism algorithm. It is an alternative algorithm to decision trees which induces modular rules that are qualitatively better than rules in the form of decision trees, especially if there is noisy data or there are clashes in the dataset.

Unfortunately Prism is computationally much more expensive than decision tree induction algorithms and thus is rarely used. We described the work we did to scale up the serial version of Prism by applying presorting mechanisms to it, which resulted in a speed up factor of 1.8. We further introduced the idea of scaling up Prism by distributing the workload in the form of attribute lists over several machines in a local area network and inducing rules in parallel.

We described the basic algorithm and architecture of the parallel version of Prism which we call P-Prism. We aim to parallelise Prism by using a distributed blackboard system via which Worker Machines exchange information about their locally induced rule terms.

We further outlined how we can reduce the size of the data that needs to be held in memory by each worker machine by using class distribution lists rather than attribute lists. We described the problem that repeated attribute values will cause if we use class distribution lists, proposed 3 different ways to resolve the problem and concluded that using an upper and lower case representation of the class value is the best solution.

A further problem we briefly addressed is synchronisation, in particular the idle time of worker machines caused by waiting for global information. We propose to use this idle time to induce a rule term for a different class value in the meantime.

References

1. M Bramer. Automatic Induction of Classification Rules from Examples Using N-Prism. In *Research and Development in Intelligent Systems XVI*. Springer, 2000.
2. J Catlett. *Megainduction: Machine learning on very large databases*. PhD thesis, University of Technology, Sydney, 1991.
3. J Cendrowska. PRISM: an Algorithm for Inducing Modular Rules. *International Journal of Man-Machine Studies*, 27:349–370, 1987.
4. D. Berrar, F. Stahl, C.S. Goncalves Silva, J.R. Rodrigues, R.M.M Brito, and W. Dubitzky. Towards Data Warehousing and Mining of Protein Unfolding Simulation Data. *Journal of Clinical Monitoring and Computing*, 19:307–317, 2005.
5. F. Provost and V. Kolluri. Scaling Up Inductive Algorithms: An Overview. In *Third International Conference on Knowledge Discovery and Data Mining*, pages 239–242, California, 1997.
6. F. Stahl. Systems Architecture for Distributed Data Mining on Data Warehouses of Molecular Dynamics Simulation Studies. Master's thesis, University of Applied Science Weihenstephan, 2006.
7. F. Stahl, D. Berrar, C.S. Goncalves Silva, J.R. Rodrigues, R.M.M. Brito, and W. Dubitzky. Grid Warehousing of Molecular Dynamics Protein Unfolding Data. In *Fifth IEEE/ACM Int'l Symposium on Cluster Computing and the Grid*, 2005.
8. L. J. Frey and D. H Fisher. Modelling Decision Tree Performance with the Power Law. In *eventh International Workshop on Artificial Intelligence and Statistics*, San Francisco, CA, 1999.
9. L. Nolle, K. C. P. Wong, and A. A Hopgood. DARBS: A Distributed Blackboard System. In *Twenty-first SGES International Conference on Knowledge Based Systems*, Cambridge, 2001.
10. J. C. Shafer, R. Agrawal, and M Mehta. SPRINT: A Scalable Parallel Classifier for Data Mining. In *Twenty-second International Conference on Very Large Data Bases*, 1996.
11. F. Stahl and M. Bramer. Towards a Computationally Efficient Approach to Modular Classification Rule Induction. In *Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*, Cambridge, 2007. Springer.
12. T. Barrett. NCBI GEO: mining millions of expression profiles-database and tools. *Nucleic Acids Res.*, 33:D562–D566, 2005.
13. M. J. Zaki, C.-T. Ho, and R. Agrawal. Parallel Classification for Data Mining on Shared Memory Multiprocessors. In *Fifteenth International conference on Data Mining*, 1999.