Strategies for Parallelizing Data Mining

D.B. Skillicorn
Department of Computing and Information Science
Queen's University, Kingston, Canada
skill@qucis.queensu.ca

Abstract

We classify parallelization strategies for data mining algorithms, concentrating on those techniques in which training data is partitioned, and extracted properties shared between processors at the end of each phase. This approach has been extensively investigated for association rules and decision trees. We sketch some similar work for supervised and unsupervised neural networks, and for the kDNF *technique. Possibilities for more sophisticated, hybrid, algorithms are also indicated.*

Parallel data mining techniques differ from other parallel algorithms chiefly in the volume of data that they must handle. For such algorithms, it is critical to reduce data to compact forms as early as possible, and to move data as little as possible, both between disks and processors, and between processors. Complexity measures that emphasise the cost of communication (and often memory use) are more relevant than those based on execution time of the computation phases.

In general, data mining algorithms are given a set of examples, and learn concepts, that is simpler representations that characterize the examples. Very often, concepts are, or can easily be expressed as, predicates. In some cases, the goal is to embed these concepts in a classifier (mapping each new example to an appropriate class). A good example is a decision tree. In other cases, the goal is to determine the concepts themselves, since they provide insight into the structure of the data.

Sequential data mining algorithms have extremely large runtimes, even for modestly-sized data, and the volume of data available for analysis is growing rapidly. It is natural to look for parallel algorithms to do mining.

Parallelization strategies for data mining algorithms can be categorized as shown in Figure 1. The Figure shows single phases; these are typically repeated many times during a single execution of the algorithm.

Techniques of Type 1 represent naive parallelizations. Although the time spent in computation may be reduced, using p processors, by a factor of p, the need for the each processor to examine all of the data prevents significant performance improvements. Typical examples in this class are naive parallelizations of association rules and decision trees, and parallel genetic algorithms. The common requirement is a need to examine all of the data to compute global frequencies or scores. The data distribution technique for association rules [1] performs relatively poorly for essentially the same reason.

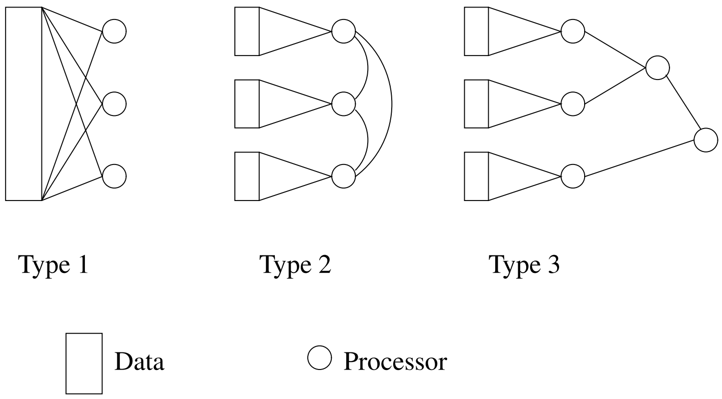

Figure 1. Classification of data mining parallelization strategies.

Type 2 represents the majority of approaches being investigated at present. On each pass, a processor is responsible for *1/p*th of the data, and exchanges information gleaned from the data at the end of each pass. This information may represent, in order of increasing sophistication, data based on frequencies or counts, partial conceptual data, or complete concepts. The critical property is that the volume of information exchanged is much smaller than that of the data itself.

For example, the count distribution association rule technique [1] exchanges the counts of local support for each itemset [1]. Similarly, the SPRINT [8] and ScalParC [5] decision tree techniques exchange split points in attribute tables.

While it is important that the volume of data to be exchanged is minimized, this is less critical than it seems for the following reason. Traditional (i.e. numeric and scientific) parallel algorithms go to great lengths to avoid the use of data-intensive communication patterns, such as total exchange. As a result, little attention has been paid to their implementations, which are often extremely poor. Recent work on implementations of BSP [3, 4] has shown that performance improvements of two orders of magnitude over both manufacturers' implementations and software implementations such as MPI are possible. This casts some doubt on the conclusions of previous performance studies (for example, it appears that the performance improvement produced by the Intelligent Data Distribution association rule technique [2] over the Data Distribution technique [1] would largely have been generated automatically by the BSP runtime support).

Less well-known data mining algorithms that can be effectively parallelized in this way are supervised and unsupervised neural networks, and inductive logic techniques such as *kDNF*. In [7] we show that there is essentially only one scalable way of parallelizing neural network training, exemplar parallelism. In this approach, the training set is partitioned into p subsets; each processor uses a deterministic learning technique on its subset, computing an error vector; these error vectors are exchanged among processors; and each forms the vector sum of the error vectors and uses it to adjust the set of weights. This process is then repeated for an appropriate number of epochs. Once again this fits the Type 2 pattern, but here the error vectors being exchanged represents intermediate ground between simple data (counts) and concepts. Note also that, while association rules generate intermediate data (candidates) that grows polynomially with the 'width' of the input (the number of items), neural network intermediate data (weights) grows at a rate more closely connected to the complexity of the concepts being learned (the number of internal nodes).

The *kDNF* technique is (the complement of) an inductive technique suggested originally by Valiant [9]. Training data consists of a set of positive and a set of negative examples, which we assume to be binary strings for simplicity. Each binary string denotes the presence of absence of each member of a list of attributes in an example. The algorithm's goal is to find a disjunction of conjuncts of length no greater than k in the attributes which describes all the positive examples, but none of the negative ones. (In some

[1] Interestingly, the count distribution rule outperforms the other two algorithms investigated in [1], even though it significantly replicates computation. This bears out the claim made above that reduction of communication is much more important in practice than reduction of computation. There is further discussion of this point, backed by experimental data in [5].

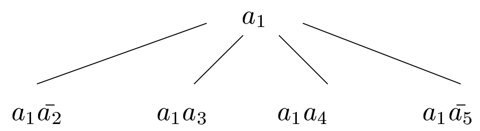

Figure 2. Sample k-conjunct tree.

data mining applications it may be enough to find all of the 'interesting' conjuncts that describe no negative examples.) This is a particular attractive form for results since humans find DNF intuitive (e.g. a bird is a creature that flies or a penguin or an ostrich).

The *kDNF* algorithm has received little attention in the literature, but Page and Srinivasan have implemented it (David Page, personal communication) and so have we. The approach is to find k-conjuncts using trees based on the positive examples. For example, if $a_1\bar{a_2}a_3a_4\bar{a_5}$ (=10110) is a positive example and $k = 2$ then the tree shown in Figure 2 is grown. Each node is given a score, the number of positive examples it covers (a_1 is equivalent to a_1-don'tcare-don'tcare-don'tcare-don'tcare), provided that it covers no negative examples. If it covers any negative examples, it is left unscored.

The scores of nodes decrease down the branches of the tree, since instantiating a new attribute cannot cover more positive cases than before. However, instantiating a new attribute may create a node that no longer covers some negative exemplars that were previously covered, so that unscored nodes may have descendants with a positive score. Thus the highest scoring nodes are found along a frontier of the tree representing those nodes closest to the root that cover no negative examples.

The formation of the disjunction may be done in a number of ways. In the standard formulation of the problem, where the goal is to produce a single disjunction covering all positive examples, it suffices to set a bit associated with each positive example when a conjunct that covers it is found. When the bits are set for all of the positive examples, a greedy algorithm is used to construct the disjunction. The highest scoring conjunct is chosen, and the scores of the remaining conjuncts altered to ignore the examples covered by the first conjunct. This process is repeated until all positive examples are covered. This approach to disjunction is not particularly useful for data mining, since it tends to overemphasise conjuncts with high scores, which tend to capture trivial facts about the example set. A much better approach would be to score conjuncts that cover a moderate number of examples more highly in building the disjunct, but I am not aware of any principled way to do so in this context.

Observations of a sequential implementation of *kDNF* indicate that the runtime increase linearly with the number of examples, polynomially with the number of attributes, and exponentially (with large constants) with k. We have managed to find 6-conjuncts on some data, and this apparently represents that largest value of k yet achieved. Runtimes in the tens of hours on a SUN Ultra are required. Any reasonably amount of parallelism is unlikely to push accessible values of k much higher, but it may be useful in practice to bring runtimes of many hours down to a few minutes.

The algorithm is trivially parallelized by constructing the trees based on each positive exemplar independently. This is a Type 1 solution, since the full set of positive and negative exemplars must be available to each processor (historically, these sets have been small). More interesting Type 2 extensions can, however, be made by noting the following. First, branch and bound techniques can be used to prune the search in some trees based on knowledge of answers already obtained by others (if only the highest scoring conjuncts are required). Second, a node that covers negative examples becomes a candidate conjunct by the instantiation of a single new attribute. This means that the complement of the newly-instantiated attribute must describe some of the negative examples. Sharing this information with other processors allows them to search paths that use the same instantiation, and hence rule out the same negative examples, earlier. Third, some processors may be used to compute the inverse problem, in which the role of positive and negative examples is reversed. Conjuncts discovered by this inverse search are properties characteristic of some of the negative examples. This information can be used by the positive search to prune its search trees (and concepts discovered by the positive search can, of course, be used to further speedup the negative search). In these last two extensions, the data shared between processors is conceptual. Further progress towards a Type 2 solution seems possible by partitioning conjuncts and example sets based on the values of some of their attributes, in the style of the Candidate Distribution association rule technique [1].

Type 3 represents a possibility about which little is known. For this type, an initial phase, perhaps repeated, reduces the data to a compact form. However, further processing of this compact form takes place to extract further information. Many inductive logic approaches (e.g. Progol [6]) have the potential for this style of parallelism.

This approach also fits well with some hybrid parallel data mining techniques. For example, unsupervised neural networks are opaque, that is the result is a system which is capable of classifying examples, but which provides no conceptual information to the user about *why* examples are classified as they are. Understandably, such higher-order information is felt to be useful. However, once an unsupervised network has been trained, it is possible to partition the training set by outcome, so that each partition contains those examples which produce the same output classification. A technique such as *kDNF* can then be used to produce a conceptual description of the commonalities of each partition. This blend of conceptual discovery techniques with those that generate opaque solutions seems to have exciting potential.

References

[1] R. Agrawal and J. Shafer. Parallel mining of association rules: Design, implementation and experience. Technical Report RJ10004, IBM Research Report, February 1996.

[2] E.-H. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *ACM-SIGMOD Internation Conference on Management of Data*, May 1997.

[3] J.M.D. Hill and D.B. Skillicorn. Lessons learned from implementing BSP. In *High-Performance Computing and Networks*, Springer Lecture Notes in Computer Science Vol. 1225, pages 762–771, April 1997. Also appears as Oxford University Computing Laboratory Technical Report TR-96-21.

[4] J.M.D. Hill and D.B. Skillicorn. Practical barrier synchronisation. In *6th Euromicro Workshop on Parallel and Distributed Processing (PDP'98)*, pages 438–444, Barcelona, Spain, January 1998. IEEE Computer Society Press.

[5] M.V. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. In *Proceedings of IPPS/SPDP'98*, 1998. to appear.

[6] S. Muggleton. Inverse entailment and Progol. *New Generation Computing Systems*, 13:245–286, 1995.

[7] R.O. Rogers and D.B. Skillicorn. Using the BSP cost model to optimize parallel neural network training. In *Workshop of Biologically Inspired Solutions to Parallel Processing Problems (BioSP3), in conjunction with IPS/SPDP'98*, March 1998. to appear.

[8] J. Schafer, R. Agrawal, and M. Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of VLDB22*, Mumbai, India, 1996.

[9] L.G. Valiant. A theory of the learnable. *Communications of the ACM*, 27(11):1134–1142, November 1984.