Learning Rules from Distributed Data

Lawrence O. Hall[1], Nitesh Chawla[1], Kevin W. Bowyer[1],
and W. Philip Kegelmeyer[2]

[1] Department of Computer Science and Engineering, ENB 118
University of South Florida, 4202 E. Fowler Ave.
Tampa, Fl 33620
{hall, chawla, kwb}@csee.usf.edu
[2] Sandia National Laboratories
Advanced Concepts Department
P.O. Box 969, MS 9214
Livermore, CA, 94551-0969
wpk@ca.sandia.gov

Abstract. In this paper a concern about the accuracy (as a function of parallelism) of a certain class of distributed learning algorithms is raised, and one proposed improvement is illustrated. We focus on learning a single model from a set of disjoint data sets, which are distributed across a set of computers. The model is a set of rules. The distributed data sets may be disjoint for any of several reasons. In our approach, the first step is to construct a rule set (model) for each of the original disjoint data sets. Then rule sets are merged until an eventual final rule set is obtained which models the aggregate data. We show that this approach compares to directly creating a rule set from the aggregate data and promises faster learning. Accuracy can drop off as the degree of parallelism increases. However, an approach has been developed to extend the degree of parallelism achieved before this problem takes over.

1 Introduction

Training data may be distributed across a set of computers for several reasons. For example, several data sets concerning telephone fraud might be owned by separate organizations who have competitive reasons for keeping the data private. However, the organizations would be interested in models of the aggregate data.

Another example is very large datasets that will not fit in a single memory which are useful in the process of learning a classifier or model of the data. It is now possible to have training data on the order of a terabyte which will not fit in a single computer's memory. A parallel approach to learning a model from the data will solve the practical problem of how to deal with learning from large data sets.

This paper describes an approach that learns a single model of a distributed training set in the form of a set of rules. A single model may be an advantage in the case that it will be applied to a large amount of data. For example, consider

M.J. Zaki, C.-T. Ho (Eds.): Large-Scale Parallel Data Mining, LNAI 1759, pp. 211–220, 2000.

the problem of visualizing "interesting" regions of a large data set. A set of rules might be learned which can do this. These rules would then be applied to similarly large data sets to guide the user to the interesting regions.

This paper examines an approach to generating rules in parallel that is related to work by [1,2]. A set of rules will be generated from disjoint subsets of the full data set used for training. Given N disjoint subsets of the full dataset there will be N sets of rules generated. Each subset of data may reside on a distinct processor. The distributed rule sets must be merged into a single rule set. Our focus is towards using a large N with very large training sets.

The final set of merged rules should be free of conflicts and have accuracy equivalent to a set of rules developed from the full dataset used for training. We discuss an approach to building a single, accurate set of rules created from N rule sets. The question of how similar to one another rule sets developed sequentially and in parallel might be is explored. Experimental results on several small, representative datasets show that accuracy tends to decline as N increases. A method to reduce this tendency is presented.

In Section 2 the generation of rules in parallel and the combination of rule sets is discussed. Section 3 contains experimental results and a discussion of the issues shown by an analysis of them. Section 4 contains a summary of the strengths and open questions associated with the presented approach to learning in parallel.

2 Generating Rules in Parallel and Combining Them

The disjoint subsets of extremely large data sets may also be very large. In principle any approach that produces rules can be used to learn from each data set. It is possible, for example, to learn decision trees [3,4] in a fast, cost effective manner. Learning a decision tree, pruning it and then generating rules from the pruned tree will be an effective competitor from a time standpoint to other rule generation approaches such as RL [5] or RIPPER [6].

In the work reported here, rules are created directly by traversing pruned decision trees (with the obvious optimization of removing redundant tests). The process of creating rules from decision trees in a more time consuming fashion has been covered in [3,7]. In learning rules it is often the case that a default class is utilized. However, it is desirable to avoid having default classes for examples because the lack of a model for some examples cannot be resolved in a straightforward way when rule sets are merged.

Each rule that is created will have associated with it a measure of its "goodness" which is based on its accuracy and the number and type of examples it covers. We are using a normalized version of Quinlan's certainty factor [8,2] to determine the accuracy of a rule R over an example set E as:

$$acc(R, E) = (TP - 0.5)/(TP + \rho FP), \tag{1}$$

where TP is the number of true positives examples covered by R when applied to E, FP is the number of false positives caused by R when applied to E, and

ρ is the ratio of positive examples to negative examples for the class of the rule contained in the training set.

A rule, R, must have acc(R,E) $\geq t$ for some threshold t in order to be considered acceptable over a set of E examples. When a rule is built on a single subset of data, its accuracy may change as it is applied to each of the other subsets of data. The rule can be discarded whenever its accuracy is less than t or only after it has been applied to all of the distributed examples and has an accuracy below the threshold.

Discarding a rule as soon as it is below the accuracy threshold will save the testing time on other processors and some communication time required to send it and its current TP/FP count to another processor. Testing time is not likely to be very high and communication time for one rule will generally be low. So, the per rule savings may be relatively low. On the other hand a rule which performs poorly on one partition and then improves to be acceptable or quite good will be ruled out under the incremental deletion approach. Our approach will be to only delete rules after testing is complete.

2.1 Merging Rule Sets Generated in Parallel

In [9] it is shown that any rule which is acceptable, by the accuracy definition in (1), on the full training set will be acceptable on at least one disjoint subset of the full data. This suggests that a rule set created by merging sets of acceptable rules learned on disjoint subsets of a full training set will contain rules that would be found on the full training set. Earlier work on building rules in parallel and then merging them [2] found that the merged set of rules contained the same rules as found by learning on the full data set and some extras. In that work, the training set was large, over 1,000,000 examples. The same paper expressed the belief that the same rule set would be found in parallel as generated sequentially.

However, in Figure 1 we show a small illustrative data set for which the rules learned by merging disjoint rule sets built on a disjoint 2 partition of the data do not include any rules learned by training on the full data set. Information gain is used to choose the attribute test for each node in the decision tree [4].

Figure 1 shows that a merged rule set, with each of the constituent rule sets developed in parallel on a disjoint training set, may in the extreme contain no rules in common with the rules created by training on the union of all the subsets (i.e. the full training set). The final merged rule set will depend upon how the examples are partitioned. The mix of examples needs to reflect the mix of available training examples.

The example data set and results from it shown in Figure 1 suggest that the accuracy of merged rules may well be different from the accuracy of the rules created from the full training set. Our experimental results will examine how different the accuracy may be and how it is affected by the number of partitions made from the training data.

As rule sets are merged, contradictions may be introduced in the sense that an example may be classified into different classes by two rules. As an individual rule is applied to more labeled examples its accuracy may change significantly.

Examples

name	attr. 1	attr. 2	class
A	1	1	C1
B	4	3	C1
C	3	2	C1
D	2	4	C2
E	5	2	C2
F	6	1	C2

Sorted on Attr. #1

name	attr. 1	class
A	1	C1
D	2	C2
C	3	C1
B	4	C1
E	5	C2
F	6	C2

Sorted on Attr. #2

name	attr. 2	class
A	1	C1
F	1	C2
C	2	C1
E	2	C2
B	3	C1
D	4	C2

Processor 1 gets examples BCD and produces the rules:

if attr 1 > 2 then C1.
if attr 1 <= 2 then C2.

or

if attr 2 > 3 then C2.
if attr 2 <= 3 then C1.

Processor 2 gets examples AEF and produces the rules:

if attr 1 <= 1 then C1.
if attr 1 > 1 then C2.

From the full training set we get the rules:

if attr 1 > 4 then C2.
if attr 1 <= 4 and attr 2 <= 3 then C1.
if attr 1 <= 4 and attr 2 > 3 then C2.

Fig. 1. An example where rules built in parallel on disjoint subsets **must** be different from rules built on the full data set. Using information gain to decide the splits.

Consider two rules R1 and R2 which classify an overlapping set of examples into two different classes. As the rules are applied to all of the subsets of the original training examples, the accuracy of one of them is expected to become less than a well-chosen threshold, t. Hence, one will be removed and the conflict resolved. However, it is possible for partially conflicting rules to survive.

For example, from the Iris data set [10] using 2 partitions we get the rules shown in Figure 2. The final accuracy after they have been applied to all training examples (but learned only from the examples in one partition) is shown in the second set of brackets associated with the rule. Both rules perform quite well when the accuracy measure in (1) is applied to them. If the conflict is not resolved, then rule ordering will affect the rules' performance. A reasonable choice might be to give higher priority to the rule with a better value of acc(R,E). Alternatively, conflict can be resolved as shown in [11,1]. Essentially conditional tests can be added to one or both rules resulting in specialization of the rules. Any examples left uncovered can then be classified by a newly created rule.

Here, we will remove the lowest performing conflicting rule with any uncovered examples being assigned to the majority class.

```
if petal width in cm > 0.4 and
   petal length in cm > 4.9
then class Iris-viginica [1/23]
                         [1/40]
if  0.5 < petal width in cm <= 1.6
then class Iris-Versicolor [0/23]
                           [4/48]
```

Fig. 2. Example of two rules from 1 fold of an Iris data 2 partition which have conflicts but survive to the final set. The numbers in brackets are the false positives and number of examples covered respectively. The second set of numbers for a rule is its accuracy after it is applied to the partition on which it was not learned.

Another type of conflict occurs when two rules for the same class created from different disjoint subsets have coverage which overlaps. For example, the rules shown in Figure 3, can be combined as the second more general rule. In general when there are overlaps among rules for the same class, the more general test is used.

```
a) if x > 7  and x < 15 then Class1
b) if x > 9 and x < 16 then Class1
c) if x > 7 and x < 16 then Class1
```

Fig. 3. Two overlapping rules, a and b, can be replaced by the third, c.

3 Experiments

The experiments reported here are from two datasets from the UC Irvine database [10] both of which consist of all continuous attributes. The IRIS data set [12] has 150 examples from 3 classes and the PIMA Indian diabetes data set has 768 examples from 2 classes. We are interested in how the accuracy is affected by partitioning these small data sets into N disjoint subsets, learning decision trees on the subsets, generating rules from the decision trees and then merging the rules into a final set of rules.

Our experiments were done using 10 fold cross validation [13]. For an individual data set and a given number of disjoint subsets, N, 10 partitions of the data were made each consisting of 90% of the train data with a unique 10% held back for testing. From each fold, N disjoint subsets are created. C4.5 is applied

to each of the N subsets and rules are created from the generated decision tree. The rules created from the j^{th} subset of data are then applied to the N-1 other subsets. The accuracy of each rule must be greater than the chosen threshold, t, in order for the rule to remain in the final set of rules. The default for the threshold t was chosen as 51, just slightly better than guessing every example belongs to the class of the rule. For the Iris data we chose t=75 rather arbitrarily. Setting t in an appropriate and systematic way must still be addressed.

For the Iris data, we have done an experiment with N=2. With the default C4.5 release 8 parameters the results on Iris for 10-fold cross validation and the results from the approach described here (with 2 different choices for certainty factors or cf's for use in pruning) are given in Table 1. The average number of rules was 6.5 for the default cf=25 and 3.1 for cf=1.

The reason for decreasing the certainty factor for pruning was to make the rules produced on the data subsets more general and less likely to overfit on the small number of examples. On this dataset there was a small positive impact.

Table 1. Results on the Iris data set using 10-fold cross-validation for a 2 processor partition. sd - standard deviation.

C4.5 % Correct ± sd	Pruned (cf=25) % Correct ± sd	Pruned (cf =1)% Correct ± sd
95.3 ± 6.01	94 ± 6.96	94.7 ± 5.81

The results for the simulated parallel approach are insignificantly worse than for C4.5 with default parameters but comparable to C4.5 with the cf=1 (94.7% and std= 5.81%).

A more interesting experiment is to look at a significant number of partitions. With the larger Pima data set, experiments were run with N=2, N=4, . . ., N=10, and N=12. The results of a 10-fold cross-validation experiment with C4.5 using its default pruning (cf=25) were an average accuracy of 73.90% with sd=4.26% and an average of 23.8 rules. Figure 4 shows plots of accuracy, standard deviation and the number of rules for 10-fold cross validation experiments with each of the above N disjoint partitions. The performance of rules created from the unpruned tree, the pruned tree with the certainty factor of 25 and a certainty factor of 1 are shown. It can be seen that the accuracy of the rule set generally decreases as N increases. The standard deviation tends to get large, suggesting that performance on some folds is quite poor. The number of rules that remain in the final set remains fairly constant as N is increased. There are significantly less rules, after conflict resolution, than when training is done on the full data set.

3.1 Discussion

The results obtained here come from small datasets. However, we believe the issue of rule accuracy falling off can also occur with larger datasets. Our results are consistent with those found in [14] where experiments were conducted on

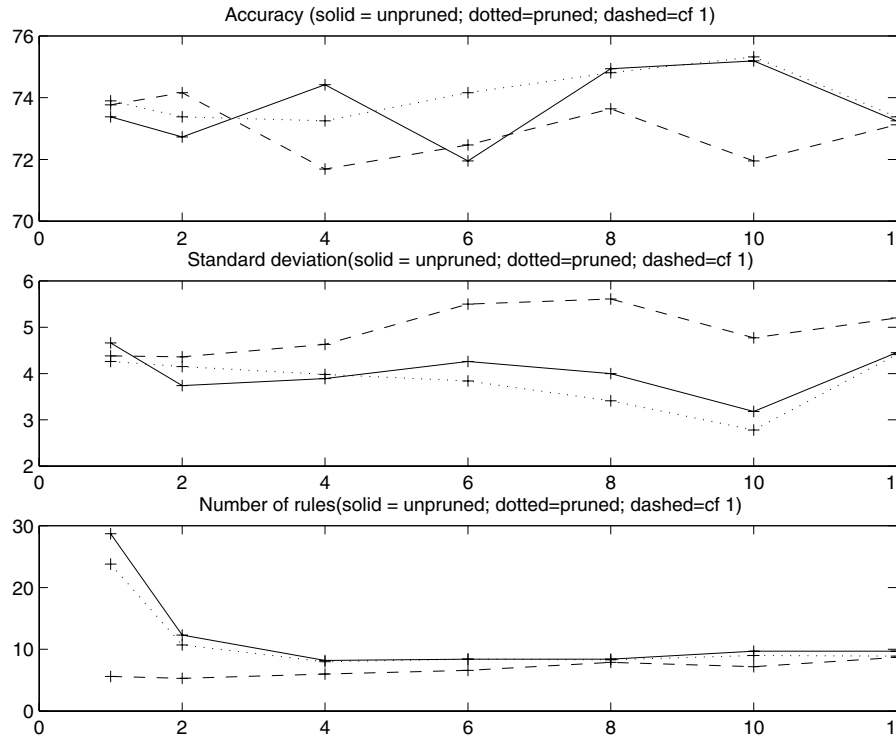

Fig. 4. Results from PIMA dataset experiments.

data sets related to the human genome project. It was found that more partitions resulted in lower performance on unseen data. Their approach to combining the classifiers was different, relying on voting among the learned set of classifiers.

Some success in mitigating the effect of learning in parallel may be gained by using a combiner or arbitrator approach to integrating multiple classifiers [15,16]. However, such approaches entail retaining all N classifiers learned in parallel and may be problematic for large N. There is not a single model of the data either.

In [2] an approach similar to ours was used on a very large dataset (over 1,000,000 examples) and there was no drop off in accuracy for partitions up to N=4. Our results suggest that accuracy would fall off as N increased.

If there are enough representative examples of each class in each of N disjoint partitions, the combined ruleset will have high accuracy. Clearly, the limit case is that each of the N subsets has an example which exactly or in the case of continuous data, almost exactly, matches each of the examples in the other subsets. So, the data really consists of only $|S_i|$ distinct examples, where $|S_i|$ is the set of examples at the i^{th} compute node.

The very worst case is that $|S_i| = C$, the number of classes in the data. In this case each subset consists of just one example of each class. Clearly, this is an unreasonable choice for N and no one would make it in practice.

Under the approach to parallel rule generation covered here there is the usual question of how large N can be before communication costs begin to slow the rule generation process significantly. However, there is the more important question of determining an N for which the accuracy of the resultant rule set is acceptable. In datasets that are too large to learn from on a single processor, it will not be possible to know what the maximum accuracy is.

Clearly with this approach a tradeoff between accuracy and speed exists. The use of more processors promises that each can complete its task faster on a smaller training set at the usual cost of coordinating the work of all the processors and waiting for the combination of rules to be completed. However, there is a second *accuracy* cost that will be paid at some point as N becomes large. What the point of significant accuracy falloff is and how to recognize it is an open question.

Improving Highly Parallel Performance. On very small datasets, the rules learned will tend to be too general. A good rule on one dataset may prove to wrongly classify many examples on another processor which belong to a different class. Specializing such rules by adding conditional tests to them can help rule out some or all of the examples that are incorrectly classified on a given processor by a rule created on a different processor.

In a couple of small experiments, we have adopted the following strategy to improve rule performance. Any rule that is within 5% of the acceptability threshold and was created on a processor other than the current processor is a candidate for specialization. The rule is specialized by taking the examples that are classified by the rule and growing a decision tree on them. Then one takes the test(s) along the best branch and adds this to the rule to create a new specialized rule. The specialized rule as well as the original rule will be tested against the next subset of examples. The accuracy of the specialized rule is only reported on the examples available on the current processor. Both the original rule and the specialized rule can be further specialized as new data is encountered as long as their performance remains within 5% of the threshold, t.

A good feature of this approach is that there will be no decrease in performance as long as the original rule remains in the final rule set, but has a lower priority than its specializations. Any examples left uncovered by the specialized rules will still be classified by the more general rule. Of course, the general rule will only exist at the end if its accuracy value is above the threshold.

As a simple example of the potential for improvement consider the example of Figure 1. Assume you got the second set of rules using attribute 2 (Att2) from processor 1 (Proc1) and applied them to the examples held by processor 2. The rule

```
if Att2 <= 3 then C1
```

now gets only 2/5 examples correct. If it is specialized to be

```
if Att2 <= 3 and Att1 < 5  then C1
```

the rule will cover 3/3 examples correctly. Further it essentially matches the second rule obtained from the whole training set in Figure 1. This example shows how specialization would work.

On the Pima data, specialization was applied to the two partition case raising the accuracy slightly from 73.38% to 73.77%.

Rule specialization could be decided upon in other ways than in our experiment. For example, after learning, a pessimistic estimate of the rules performance [3] could be generated. For a test subset on which the classification performance of the rule was more that x% below the estimate, specialization could be carried out. To get a better estimate of the performance of a specialized rule, it might be tested against all the data on which it was not created (e.g. broadcast to all processors). This would make the conflict resolution process more accurate.

4 Summary

This paper discusses an approach to creating rules in parallel by creating disjoint subsets of a large training set, allowing rules to be created on each subset and then merging the rules. It is shown that this approach can provide good performance. It is also pointed out that the rules discovered in parallel may be different from those discovered sequentially. While it is true that rules which perform well on the full data set will perform well on at least one subset of the data, it is not necessarily the case that these rules will be discovered.

In an empirical study, it is shown that the accuracy of the merged rule sets can degrade as the number of processors, N, is increased. This raises the question of how to choose N to maximize speed and keep accuracy high. The approach discussed here uses a threshold of goodness for rules. Rules that perform below the threshold are deleted from the final rule set. The question of how to most effectively set the threshold is an open one.

It is shown that the performance of rules can be improved by further specializing those that are under performing. Conditions can be added as the rules are applied tested on data stored on other processors. Both the specialized rules and original rules remain as long as their accuracy is above the threshold.

We have pointed out issues and potential fixes to an approach that promises to provide scalable, accurate rules generated from a parallel computing system. It will enable learning from large distributed data sets.

Acknowledgments

This research was partially supported by the United States Department of Energy through the Sandia National Laboratories LDRD program, contract number DE-AC04-76DO00789. It was also partially supported by a sabbatical at the University of California, Berkeley thanks to the hospitality of Prof. Zadeh.

References

1. G. Williams, *Inducing and Combining Multiple Decision Trees*. PhD thesis, Australian National University, Canberra, Australia, 1990.
2. F. Provost and D. Hennessy, "Scaling up: Distributed machine learning with cooperation," in *Proceedings of AAAI'96*, pp. 74–79, 1996.
3. J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992. San Mateo, CA.
4. J. Quinlan, "Improved use of continuous attributes in C4.5," *Journal of Artificial Intelligence Research*, vol. 4, pp. 77–90, 1996.
5. S. Clearwater, T. Cheng, H. Hirsh, and B. Buchanan, "Incremental batch learning," in *Proceedings of the Sixth Int. Workshop on Machine Learning*, pp. 366–370, 1989.
6. W. Cohen, "Fast effective rule induction," in *Proceedings of the 12th Conference on Machine Learning*, 1995.
7. R. Kufrin, "Generating C4.5 production rules in parallel," in *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pp. 565–570, July 1997.
8. J. Quinlan, "Generating production rules from decision trees," in *Proceedings of IJCAI-87*, pp. 304–307, 1987.
9. F. Provost and D. Hennessy, "Distributed machine learning: Scaling up with coarse-grained parallelism," in *Proceedings of the Second International Conference on Intelligent Systems for Molecular Biology*, 1994.
10. C. Merz and P. Murphy, *UCI Repository of Machine Learning Databases*. Univ. of CA., Dept. of CIS, Irvine, CA. `http://www.ics.uci.edu/~mlearn/MLRepository.html`.
11. L. Hall, N. Chawla, and K. Bowyer, "Decision tree learning on very large data sets," in *International Conference on Systems, Man and Cybernetics*, pp. 2579–2584, Oct 1998.
12. R. Fisher, "The use of multiple measurements in taxonomic problems," *Ann. Eugenics*, vol. 7, 1936.
13. S. Weiss, R. Galen, and P. Tadepalli, "Maximizing the predictive value of production rules," *Artificial Intelligence*, vol. 45, pp. 47–71, 1990.
14. P. Chan and S. Stolfo, "Scaling learning by meta-learning over disjoint and partially replicated data," in *Proceedings of the Florida Artificial Intelligence Society*, 1996.
15. S. Stolfo, A. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P. Chan, "JAM: Java agents for meta-learning over distributed databases," in *Proc. KDD-97*, 1997.
16. P. K. Chan and S. J. Stolfo, "Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection," in *Proc. KDD-98*, 1998.