

Graduate School
University of South Florida
Tampa, Florida

CERTIFICATE OF APPROVAL

Master's Thesis

This is to certify that the Master's Thesis of

NITESH CHAWLA

with a major in Computer Science has been approved by
the Examining Committee on January 9, 2000
as satisfactory for the thesis requirement
for the Master of Science in Computer Science degree

Examining Committee:

Major Professor: Lawrence O. Hall, Ph.D.

Member: Kevin Bowyer, Ph.D.

Member: Dmitry Goldgof, Ph.D.

RIDE: RULE-LEARNING IN A DISTRIBUTED ENVIRONMENT.

by

NITESH CHAWLA

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science
College of Engineering
University of South Florida

May 2000

Major Professor: Lawrence O. Hall, Ph.D.

©Copyright by Nitesh Chawla 2000
All rights reserved

DEDICATION

I would like to dedicate this thesis to my parents for instilling in me patience and hard work that a career of research demands. I would also dedicate it to my brother Hamesh for being my pillar of support, my sister Sugana for her confidence in me and my Anita.

ACKNOWLEDGMENTS

My sincere thanks go to my advisor, Professor Lawrence O. Hall, who over the period of my thesis has been very supportive and encouraging. His consistent advice, encouragement, insight, and confidence were very useful to me. At every step, his guidance provided a spur to my confidence and helped me achieve what I did.

Many thanks to my committee for their assistance. Professor Kevin W. Bowyer helped throughout my Master's research work with his broad knowledge of the literature, which much aided in the conception and implementation of ideas. Thanks to Professor Dmitry Goldgof for assisting me in this thesis. Sincere thanks to Dr. W. Philip Kegelmeyer of Sandia National Labs. Discussions with Philip were very insightful and helped me carve a path for my research. I would like to thank him and Mark Christon of Sandia National Labs for the vorticity examples in Chapter 1 of the thesis. The vorticity examples were generated by Mark Christon, using MUSTAFA to visualize data generated by GILA, a massively parallel machine.

I would like to thank fellow researchers Catherine D. Smith, Thomas E. Moore and Jimmy Chao for their inputs.

A special thanks to my good friend Lynn Fletcher Heath, discussions with whom were very useful. She has been a great mentor and friend.

I would also like to extend a big thanks to my very close friends Bijoy, Saurabh and Giju for being so supportive and all my other friends and fellow researchers.

This research was partially supported by the United States Department of Energy through the Sandia National Laboratories LDRD program, contract number DE-AC04-76DO00789.

TABLE OF CONTENTS

LIST OF TABLES	iii
LIST OF FIGURES	iv
ABSTRACT	vii
CHAPTER 1 INTRODUCTION	1
1.1 Overview of the thesis	2
CHAPTER 2 DECISION TREE LEARNING	4
2.1 Decision trees	4
2.1.1 Pruning of decision trees	6
2.1.2 C4.5	7
2.1.3 Pruning in C4.5	9
2.1.4 CART	10
2.2 Rules	10
CHAPTER 3 RELATED WORK	13
CHAPTER 4 RULE COMBINATION I	19
4.1 What are conflicts?	19
4.2 Conflict resolution	22
4.3 Experiments	26
4.4 Summary	28
CHAPTER 5 RULE COMBINATION II	30
5.1 Introduction	30
5.2 Algorithm details	31
5.3 Experiments and discussions	36
5.3.1 Conflict resolution approaches	36
5.3.2 What is the right metric?	49
5.4 Weighted voting	55
5.5 Learn again?	60
CHAPTER 6 AVATAR DISCUSSION: A REAL-WORLD EXAMPLE	64
6.1 Introduction	64
6.2 Sample data set	65
6.3 How to train?	67

6.3.1	Training experiment	69
6.4	Testing	72
CHAPTER 7	SUMMARY AND FUTURE WORK	76
7.1	Summary	76
7.2	The journey ahead	76
REFERENCES		78

LIST OF TABLES

Table 1.	Types of conflicts.	21
Table 2.	Results on the Iris data set using 10-fold cross-validation for a 2 processor partition.	27
Table 3.	Results on the Iris data set using 10-fold cross-validation for a 3 processor partition.	27
Table 4.	Accuracy on the Pima Indian Diabetes data set using 10-fold cross-validation for a 2-processor partition.	28
Table 5.	Results on the Iris data set using 10-fold cross-validation for a 2 processor partition, using Pairwise and Global conflict resolution.	37
Table 6.	Results on the Iris data set using 10-fold cross-validation for a 4 processor partition, using Pairwise and Global conflict resolution and pruning with certainty factor = 1.	37
Table 7.	C4.5 10-fold cross-validation Accuracy results.	37
Table 8.	Results on the Pima Diabetes data set using 10-fold cross-validation for a 4,6,8,10,12 processor partition, using Pairwise and Global conflict resolution.	41
Table 9.	Standard Deviation results on the Pima Diabetes data set using 10-fold cross-validation for a 4,6,8,10,12 processor partition, using Pairwise and Global conflict resolution.	41
Table 10.	Results on the Pima Diabetes data set comparing 10-fold cross-validation on the 12-partition with c4.5.	52
Table 11.	Results on the Pima Indian Diabetes data set comparing 10-fold cross-validation on the 12-partition with C4.5.	53
Table 12.	Results from voting with Ripper on the Pima Indian Diabetes data set.	58

LIST OF FIGURES

Figure 1.	a) x-vorticity. and b) z-vorticity.	1
Figure 2.	An example of building a decision tree from a training set.	5
Figure 3.	Rules generated from a decision tree.	11
Figure 4.	Example of two rules in a conflict. These two rules have all but one condition as same.	20
Figure 5.	The attribute Plasma glucose concentration has an overlapping range of values. This is a potential conflict though all the other conditions are different.	20
Figure 6.	Rules having no conditions in common.	21
Figure 7.	A conflicting case. Rules 4 and 6 conflict.	26
Figure 8.	Summary of the procedure.	29
Figure 9.	An example where rules built in parallel on disjoint subsets <i>must</i> be different from rules built on the full data set. Using information gain to decide the splits.	33
Figure 10.	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.	36
Figure 11.	Accuracy results for Pima Indians Diabetes data set using global conflict resolution.	38
Figure 12.	Standard Deviation results using global conflict resolution.	39
Figure 13.	Number of rules graph for the Pima Indians Diabetes data using global conflict resolution.	40
Figure 14.	Percentage accuracy spread for 2-processor partition of the Pima data set.	42

Figure 15. Percentage accuracy spread for 4-processor partition of the Pima data set.	43
Figure 16. Percentage accuracy spread for 6-processor partition of the Pima data set.	44
Figure 17. Percentage accuracy spread for 8-processor partition of the Pima data set.	45
Figure 18. Percentage accuracy spread for 10-processor partition of the Pima data set.	46
Figure 19. Percentage accuracy spread for 12-processor partition of the Pima data set.	47
Figure 20. Accuracy results for Mammography data set using local and global performance estimates.	48
Figure 21. Accuracy results for Mammography data set for local and global performance estimates assigned by Ripper metric.	51
Figure 22. Accuracy results for the Mammography data set using local and global performance estimates assigned by LaPlace estimate.	54
Figure 23. Accuracy comparison between Certainty factor metric, Ripper metric and LaPlace metric.	56
Figure 24. Comparison of Voting strategies against conflict resolution for the Pima Indians Diabetes data set.	58
Figure 25. Accuracy comparison between conflict resolution and voting using Ripper.	59
Figure 26. Accuracy results for decision trees grown with $m=5$ and default pruning.	61
Figure 27. Summary.	63
Figure 28. Summary.	65
Figure 29. AVATAR front panel.	66
Figure 30. a) First frame. and b) Second frame. c) Third frame.	68
Figure 31. A view of AVATAR features panel.	68

Figure 32.	The above figure shows that the input to the training session is the Exodus data set. At the end of the train session the output produced is as follows: Flat data file : A file comprising of all the nodes data and the corresponding saliencies. Flat names file : A file comprising of all the attributes listing and classes(saliency). Exodus file : The training Exodus file recreated with saliency added as another nodal variable.	70
Figure 33.	A training frame with marked regions.	71
Figure 34.	A second training frame with marked regions.	71
Figure 35.	The above figure shows that the input to the training session is the Exodus data set. At the end of the train session the output produced is as follows: Flat results file : A file containing grouping of nodes by time-step and their corresponding saliency. HTML file : An HTML file providing a web interface for web query. Exodus file : A similar Exodus file is created with saliency added as a feature to each node.	72
Figure 36.	Node correctly classified as very interesting.	73
Figure 37.	Node incorrectly classified as very interesting.	74
Figure 38.	Node correctly classified as interesting.	75

RIDE: RULE-LEARNING IN A DISTRIBUTED ENVIRONMENT.

by

NITESH CHAWLA

An Abstract

Of a thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science
College of Engineering
University of South Florida

May 2000

Major Professor: Lawrence O. Hall, Ph.D.

In this thesis a procedure for learning in a distributed manner is proposed (LRDE). Decision trees may be grown on disjoint subsets of training data on different processors and converted to rules. The rules from these disjoint subsets of data can then be combined together. Three approaches to rule combination are presented.

1. Conflict resolution : Conflicts are resolved between two rule sets and then the rule sets are combined. This approach is an extension to the work by Williams [Wil90]. The results of this approach are promising accuracy wise for the data sets tested, but communication costs seem prohibitive. This approach requires data communication which will become a bottleneck as the size of the data sets increases.
2. Rule goodness : Each rule is passed to each processor and tested against that processors data set. A threshold for the goodness of the rule is established based on the class distribution. Any rule whose performance after evaluation against every other processors' data set is not above that threshold is ignored. Since the rules are learned on disjoint subsets of data, this approach allows conflicts in the final merged rule sets. If two rules conflict with each other, the rule having worse performance is discarded. Two approaches towards discarding conflicting rules were tried, *pairwise conflict resolution* and *global conflict resolution*. In the pairwise conflict resolution approach, the rules are ordered by their performance estimate and compared as pairs and the worse performing rule is removed. In the global conflict resolution approach, a table of conflicting rules across the entire rule set is created and the worst performing rules are eliminated. The global conflict resolution approach is more promising than the pairwise conflict resolution approach.
3. A comparison of the conflict resolution approaches with the voting approach is also presented. In the voting approach, data sets are distributed across proces-

sors, resulting in disjoint partitions. Rules are learned on each of the partition and then used to classify unseen examples. The rules could be given a uniform weight of one or weighted by a performance metric. This approach would resolve conflicts at the time of classification, assigning a class to the unseen example based on the majority (weighted) vote.

Results on data sets from the UCI Irvine repository are analyzed. A real world application of the pattern recognition algorithm, AVATAR (Adaptive Visualization Aid for Touring and Recovery), is also discussed.

Abstract Approved: _____
Major Professor: Lawrence O. Hall, Ph.D.
Professor, Department of Computer Science and Engineering
Date Approved: _____

CHAPTER 1

INTRODUCTION

This thesis explores learning decision rules from distributed data sets in a parallel environment. Modern scientific data sets are becoming so large, that it has become very difficult to learn on a single processor from all available data. We are entering into a world of terabytes of data, and to capture the exact underlying description from a terabyte of labeled data is a challenge. For example, visualization tools need to be able to guide the user through the data set and do an exploratory analysis of data and help highlight the relevant details.

To illustrate, consider an example¹. Figure 1 shows the instantaneous x- and z-vorticity fields for large-eddy simulation with a Reynolds number of 10,000. This flow exhibits Taylor-type instabilities and Taylor-Görtler vortices. The latter can be seen most clearly in the vortex-pairs in the x-vorticity image, at the right end of the box. These features are of great interest. But though they are easy to recognize once located, they are difficult to describe or articulate a priori.

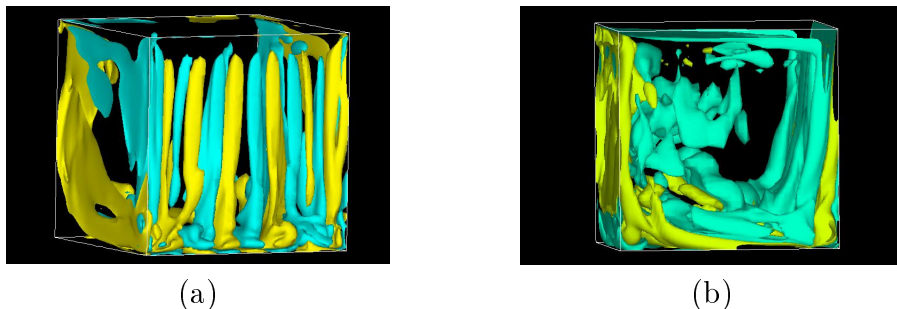


Figure 1. a) x-vorticity. and b) z-vorticity.

¹The vorticity examples were generated by Mark Christon, using MUSTAFA to visualize data generated by GILA, a massively parallel machine and were provided by W. Philip Kegelmeyer.

Another example could be a diabetes data set comprised of medical records collected from a population. It is desirable to learn rules based on these medical records and identify patients in the population having diabetes. This process of learning rules discovers information embedded in the data and presents empirical rules for classification of unseen data.

The problem calls for pattern recognition techniques, perhaps with visualization tools providing an interface for learning. Very large data sets may not fit in memory, thus the need for a distributed and/or parallel learning environment.

The visualization tool will provide the mechanism, for a supervised labeling session. The data set will be labeled by experts to identify salient regions in the data set. This will result in a very large number of training instances because a visualization data set typically consists of a series of patterns over time. The labeled data generated in this way can be used to train a classifier.

In this thesis, a method is proposed to deal with learning from very large data sets. The data will be partitioned into disjoint subsets, decision trees learned on each and converted into decision rules. The decision trees are learned using C4.5 [Qui92] release 8. The rules will be combined based on a set of heuristics. Ideally, the final learned rule model should be free of conflicts and have an accuracy equivalent to that of the rule set learned from the entire data set.

1.1 Overview of the thesis

This thesis delves into an approach for distributed learning which results in a single classifier model. Chapter 2 presents the related work in this area. Chapter 3 presents background information on decision trees and rules and some of the existing approaches to learn trees and rules. Chapter 4 proposes a variation in the conflict resolution algorithm proposed in [Wil90] and how that fits in the distributed environment. Chapter 5 proposes another distributed rule learning mechanism based on

some work in [FJP96]. Chapter 5 also deals with alternate strategies of rule combination. Chapters 4 and 5 also contain experiments to illustrate the theory. Chapter 6 shows an application of the pattern recognition techniques proposed in the thesis and discusses the approach to develop an Adaptive Visualization Aid for Touring and Recovery (AVATAR) for large data sets. This is an integration of the learning algorithm with the MUSTAFA visualization tool. Chapter 7 provides the summary and directions for future work.

CHAPTER 2

DECISION TREE LEARNING

2.1 Decision trees

Inductive learning identifies relationships between the attribute values of training examples and the class of the examples, thus establishing a learned function. Decision trees [BFOS84, Qui87, Qui92] are a popular classifier for inductive inference. Decision trees are trained on examples comprised of a finite number of predicting attributes together with class labels and a learned model is established based on tests on these attributes. This learning mechanism approximates discrete valued functions as the target attribute. The types of training examples applicable to decision trees are diverse, they could range from consumer credit records to medical records. Decision trees help in extracting the underlying meaning in data.

A decision tree is a directed tree structure comprised of nodes. Each node of the decision tree specifies a test on an attribute and the branch emanating from that node is labeled by the value of the attribute tested at the node. The structure of a decision tree is comprised of a unique root node of an in-degree zero and an out-degree greater than or equal to zero, one or more leaves, having an in-degree of equal to zero and an out-degree of zero and a series of intermediate nodes having an in and out-degree of greater than or equal to one. Each attribute test is strictly propositional, any boolean function can be written as a decision tree, though this is extensible to functions with a larger range of outputs. A decision tree is essentially in a disjunctive-conjunctive form, wherein each path is a conjunction of the attribute value pairs and the tree by itself is a disjunction of all these conjunctions. An instance arriving at the root

Examples			
	Temp	wind-speed	Decision
1.	89,	low,	Bike-day
2.	40,	low,	Couch-day
3.	60,	medium,	Couch-day
4.	80,	high,	Bike-day
5.	61,	low,	Bike-day
6.	65,	medium,	Couch-day
7.	83,	high,	Bike-day
8.	72,	low,	Bike-day

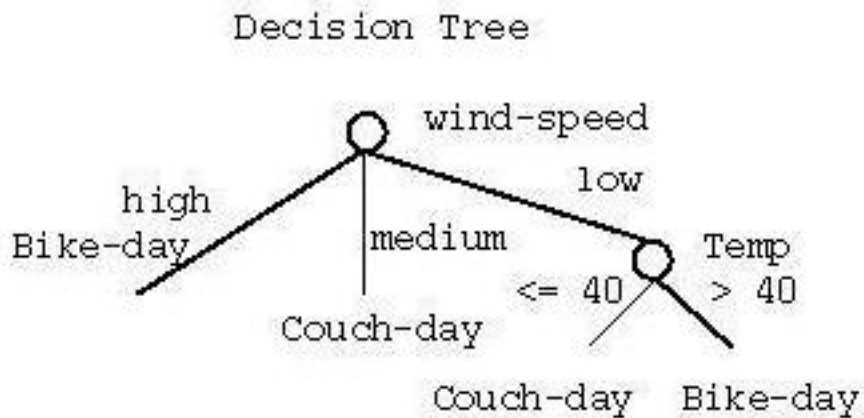


Figure 2. An example of building a decision tree from a training set.

node, takes the branch it matches based on the attribute-value test and moves down the tree following that branch. This continues until a path is established to a leaf node, providing the classification of the instance. If the target attribute is true for the instance, it is called a 'positive example', otherwise it is called a 'negative example'. Figure 2 shows a set of examples and a learned decision tree.

Decision trees have been found to be a very useful tool in the field of Data Mining for an exploratory analysis of data. Hunt et al, were the pioneers in the field of decision trees [HMS66].

Steps to Tree Construction.

1. If all the instances in the training set belong to one class, then return that class as default class and terminate the tree building.
2. Else, evaluate each attribute at a node using some heuristic for calculating the “goodness” of the attribute.
3. Take the best evaluating attribute, and create branches from the node, with each branch labeled with a value of the attribute. For continuous attributes branches are labeled with $\leq x$ and $> x$.
4. Each branch ends at a node. Partition the training set to each node based on the test at the level above.
5. When all or most of the instances at a node belong to one class, declare that node as a leaf.
6. Continue the procedure of growing nodes until all the instances fall into some leaf.

There are different measures that can be applied to determine the goodness of a particular split for an attribute [BFOS84, Min89b, Qui92].

2.1.1 Pruning of decision trees

Aiming to perfectly classify the given set of training examples, a decision tree may overfit the training set. If the training set has noise or outlying erroneous instances, the decision tree would be grown deep to perfectly classify all these examples. This would give rise to a decision tree that performs very well on the training set, but not so well on unseen examples.

Mitchell [Mit97] defines overfitting as

“Given a hypothesis space H , a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances”. [Page 67, [Mit97]]

To avoid overfitting, the approach used is *pruning of decision trees*. This allows the tree to completely grow and overfit the data, if needed, and then it is pruned. Pruning involves evaluating every node of the decision tree and the subtree of which it is the root, as a potential candidate for removal. A node is converted into a leaf node by assigning the most common classification associated at that node. There are two classes of methodologies for pruning a decision tree, Cost-complexity pruning [BFOS84] and Reduced-error pruning [Qui87]. There are a number of methods to prune a decision tree [Min89a, OJ98].

2.1.2 C4.5

C4.5 constructs trees using a divide-and-conquer approach [Qui92]. The selection of a split or test at a node is based on a statistical property, information gain.

Continuous attribute splits are associated with binary tests of \leq and $>$. If A is a continuous valued attribute, then the splits will be $A \leq X$ and $A > X$. To choose the threshold for a split, sort all the training instances associated with the attribute in consideration, say X . If there are N such attribute values, then there will be $N - 1$ possible split thresholds on X . Each such threshold can be given as

$$T = (v_i + v_{i+1})/2$$

, where v_i is the value of attribute X and $v_i \leq v_j \mid i \leq j$ so the values are in sorted order. This has the effect of dividing the training instances into two unique subsets at the node at which the attribute is evaluated. The ability to choose a threshold

t to maximize the splitting criterion favors continuous attributes with many distinct values [Qui96].

The gain ratio establishes the goodness of the attribute in separating the training examples according to the target concept. Given a set of K of training instances at a node, C the number of classes and $p(K, j)$ the proportion of cases in K that belong to the j th class [Qui96], the information is:

$$Info(K) = - \sum_{j=1}^C p(K, j) * \log_2(p(K, j))$$

The information gained by a test T with m outcomes is

$$Gain(K, T) = Info(K) - \sum_{i=1}^m \frac{|K_i|}{|K|} * Info(K_i)$$

The information gained by a test is strongly affected by the number of outcomes, being strongly biased towards cases with many outcomes. Information gain will be maximal when each subset has only one case. Hence, Quinlan uses the gain ratio criterion [Qui92, Qui96] to select among attributes. The bias towards continuous attributes with many distinct values is overcome by adding a penalty term to the Gain which is the number of distinct values at node K to the number of examples at K ; N is the number of distinct values of a continuous attribute. The threshold ranking value (TRV) at node K is then

$$TRV = GAIN(K, T) - \log_2(N - 2)/|K|$$

The attribute with highest TRV and its associated split will be used in the decision tree. In [Qui96], it has been shown that this produces compact and accurate trees when compared with the gain ratio criterion.

2.1.3 Pruning in C4.5

C4.5 uses an approach called the pessimistic pruning [Qui92]. Pessimistic pruning doesn't require a separate test set for the pruning process. It has been shown to be quite fast, producing trees perform adequately [Min89a, Qui92].

Error complexity and cost complexity pruning have been shown to produce small and adequately performing trees [Min89a, OJ97]. These approaches are not very feasible for small data sets as they require a separate test set, but work well for large data sets. The downside is the computational complexity arising from large decision trees, as this entails creating and evaluating every possible subtree from the initial decision tree.

A less time consuming method which appears to result in accurate trees of reasonable size [Min89a] is reduced error pruning [Qui87]. Even though this approach also requires a separate test set, it is less time consuming than error complexity pruning since it considers only reductions of the tree which reduce error on the pruning test set. However, this results in larger trees than error complexity pruning, which can be an issue for large data sets [HCB98].

Oates and Jensen [OJ98], have pointed out that for large data sets for which training sets initially consist of a subsample, with the addition of more training examples, the tree size tends to increase without the accuracy being affected. They used C4.5 release 5 in their experiments, and they have shown that only the case of error-complexity pruning (in some cases) kept trees from growing when no increase in the accuracy of the tree was observed. Hall, Chawla and Bowyer [HCB98] found that the trees were much smaller using C4.5 release 8 on the Australian data set [OJ98, BM98]. They used pessimistic pruning, and the accuracy seemed to grow slightly with the tree size.

2.1.4 CART

CART, an acronym for Classification and Regression Trees, was developed by a group of statisticians. [BFOS84]. CART also grows decision trees using a divide and conquer approach, but allows for only binary partitions or splits at each node. There are various attribute selection criteria in CART [BFOS84]. A popular one is the *Gini* selection criteria.

This attribute selection criteria measures the separability between classes. The partition that maximally reduces the diversity between classes is chosen.

Assuming c possible classes at a node and the node split into m possible partitions, the gini index for the i^{th} partition is

$$gini_i = 1 - \sum_{j=1}^c \left(\frac{p_{ij}}{p_i}\right)^2$$

where p_{ij} is the number of examples i with class label j , c is the number of classes and p_i is the total number of examples at partition i .

The gini index for the total split is given as

$$gini_{split} = \sum_{i=1}^m \left(\frac{p_i}{p}\right) gini_i$$

The splitting decision with the minimum $gini_{split}$ is chosen.

2.2 Rules

Decision trees after pruning are simpler and more accurate, but they do not necessarily provide much insight. The decision at each node is established based on the outcomes at the antecedent nodes, making them difficult to trace. In a decision tree the leaf indicates a classification. If we trace the path from the root to that leaf, a set of

RULES

1. If wind-speed = high Then Bike-day
2. If wind-speed = medium Then Couch-day
3. If wind-speed = low and Temp \leq 40 then Couch-day
4. If wind-speed = low and Temp $>$ 40 Then Bike-day

Figure 3. Rules generated from a decision tree.

decisions at each node can be encompassed in a rule for that particular leaf. Figure 3 shows the decision tree from Figure 2 converted to set of rules.

This example takes the form of a production rule [Win92]. C4.5rules [Qui92], rewrites the tree as a collection of rules, creating a rule for each leaf of the tree and performs a computationally intensive post-pruning process on the rules.

Ripper[Coh95], is a propositional rule learning algorithm, extensible naturally to a first-order representation. Ripper is an offshoot of IREP [FW94]. It is computationally faster than C4.5rules and is comparable in accuracy to C4.5rules [Coh95]. Also, Ripper has been shown to be much more efficient than C4.5rules on noisy data sets [Coh95]. Ripper learns rules by ordering the classes by their frequency in the data set and growing rules for classes in the order of lower frequency to higher, removing the instances covered, both positive and negative for the rule created. This process is repeated, with the new rules being learned on the leftover examples, until a single class remains; this class is then used as a default class. The modifications of Ripper to IREP are in three areas, the metric for rule-performance, the stopping condition for adding rules to the rule set and optimization of rules.

Another concept of learning rules in an expert system domain is described in [FB85]. They have described a method for learning hierarchical knowledge from the knowledge base of an expert system. Each training instance is described by low level features and high level concepts. The intermediate knowledge basically represents a relationship between the low level features and high level concepts.

The reasoning hierarchy can be represented as [FB85]

They represent the learning formulation as

“Given:

A set of training instances

(or a set of $LN \rightarrow HN$).

Find: Rules of the form $LN \Rightarrow IN$ and $IN \Rightarrow HN$

consistent with the training instances”.

[FB85].

where IN : Intermediate level node. HN : High level node. LN : Low level node.

CHAPTER 3

RELATED WORK

Learning from large data sets is challenging, as the data will not fit in memory and the amount of time required to sift the data serially will be exorbitant. This thesis describes parallel approaches to learning from such data sets. In particular, the thesis investigates learning rules from disjoint sets of data and combining the rules learned into a single model. The data can be distributed to different processors and learning done in parallel.

There are other possible approaches to parallel learning from large data sets. Shafer et al [SAM96] built a decision tree in parallel. SPRINT is implemented on an IBM SP2. SPRINT associates an attribute list for each attribute in the data. Entries in the attribute list are called attribute records and are comprised of an attribute value, a class label, and the index of the training record. As a tree is grown and nodes are split, the attribute lists associated with that node are partitioned and passed on to the children. The splitting rule used by SPRINT is the Gini index. For continuous attributes histograms are maintained, these histograms are labeled as C_{above} and C_{below} . For categorical attributes a count matrix is maintained. One of the major concerns associated with parallel decision tree building is finding good split-points and partitioning the data accordingly. SPRINT assumes a nothing-shared environment and distributes the attribute lists uniformly over available processors of a machine. The partitioning is achieved by first distributing the training set examples equally among all the processors; each processor then generates attributes lists in parallel. This leads to an even partitioning of the continuous attributes. The continuous

attributes are sorted using a parallel sorting algorithm [DNS91] . With this sorting operation each processor gets fairly equal-sized sorted sections of attribute lists. SPRINT achieves uniform workload balancing. Shafer et al have found SPRINT to have excellent scale-up and speed-up characteristics.

One disadvantage of SPRINT is the attribute lists, whose size is proportional to the size of the training set. Another downside is the expensive disk I/O when the attribute list does not fit in main memory.

Joshi et al [JKK98] have proposed a parallel construction of a decision tree, capable of handling large data sets. They have shown that SPRINT is unscalable in runtime and memory requirements. ScalParC, the parallel decision tree algorithm proposed in [JKK98], is claimed to overcome these shortcomings of SPRINT and is claimed to be "truly scalable in both runtime and memory requirements". It uses a distributed hash table to implement the splitting. A scalable parallel hashing paradigm is used to achieve scalability in the splitting phase. Some experiments were run using ScalParC [MHB] on the DOE's ASCI Red machine and the decision trees generated had impossible splits. There is a flaw in the design of the algorithm.

A parallel implementation of C4.5 has been done by Darlington et al [DGST97]. Two approaches are discussed in the paper. In one approach, the entire training set is duplicated onto all processors. A master processor grows a decision tree until there are as many leaves in the intermediate tree as the number of processors. Each processor is then allocated one of the leaves and a pointer to the start and end of data for the corresponding leaf by the master processor. Each processor then completes the tree construction for its leaf. This approach is unscalable memory wise, as an increase in the training set size would increase memory costs. In the other approach, data is partitioned across processors. As in the previous approach, a master processor grows a decision tree until there are as many leaves in the intermediate tree as the number of processors and then each processor is allocated a leaf. The data for each leaf must

be communicated to each corresponding processor. This entails high communication costs and is a bottleneck. The experiments were run using three different data sets from the UCI Repository of Machine Learning Databases [BM98]. The performance of the algorithms on the data sets scaled up to 16 processors.

Rules can be created from decision trees and then post-processed. This is computationally expensive. A parallel approach to pruning the rules has been described in [Kuf97]. Kufirin has described three approaches to parallel pruning; *rule based* divides the rules over the processors, *instance-based* divides the training instances across available processors and *class-based*, this approach divides the class rule sets corresponding to one of the possible classes on the processors. Though, the paper presents the computational gain over the sequential C4.5rules, scalability with large data sets has not been shown.

Chan and Stolfo [CS93a, CS93b] have proposed an arbiter and combiner strategy. In their strategy, base classifiers are learned on the disjoint subsets of data. In the arbiter strategy, the arbiter learned is used to arbitrate among the decisions generated by different classifiers. For learning the arbiter, there is a separate validation set. For all the examples in the validation set on which there is no majority vote by the base classifiers, an arbiter is learned by the same learning algorithm that was used to generate the base classifiers. In the combiner strategy, the predictions generated by the base classifiers form the training set for the combiner. The combiner computes a prediction from the base classifier prediction. This approach combines the predictions from the base classifiers by learning a relationship between these predictions and the correct prediction. Chan and Stolfo [CS95] present a comparison of arbiter and combiner strategies by applying a learning algorithm to disjoint subsets of data. The experiments show that the arbiter strategy sustains the accuracy compared to the classifier learned on the entire data set. The combiner strategy experienced a drop in accuracy with the increase in the number of subsets, which can be attributed to

the lack of enough information content from the small subsets. In a few cases an improvement in the accuracy was observed. In the approach discussed in [CS96] they propose partitioning the training set but allowing the replication of the training set over the partitions. The results with the replication of the training set do not show any measurable improvement in accuracy. This leads to the conclusion that with data replication not much is gained in accuracy. The authors conclude that meta-learning can be successfully applied to disjoint partitions of data, without much reduction of accuracy while maximizing computational performance.

Provost and Hennesey introduce an approach in [FJP96], in which rules are learned on disjoint subsets of data over different processors and then combined into a single model. Rules are learned over the distributed disjoint subsets. When a rule is "satisfactory", that is it meets the evaluation criterion for the subset of the data, it is communicated to the other processors to get a global evaluation. If the rule adheres to the evaluation criterion globally, it becomes a "satisfactory" rule.

Models can be learned on distributed sets of data and then the models can be combined using a voting scheme. Bagging [Bre96] uses the approach of resampling, but resampling is random and all models are given equal weight. The performance of bagged classifiers has usually exceeded that of a single classifier. Since, bagging replicates data it may be possible that some data may not be a part of any training set. For a training set of diverse nature, some important data points may be missed in the training phase. In boosting [Fre95] resampling occurs based on how well the training samples are classified by the previous model. The models are weighted depending on the estimated error rate. In boosting, since the training set for one model depends on the previous model, it requires sequential runs and thus is not readily adapted to parallel environment. Merz in [Mer99] looks at combining multiple learned models with the goal of forming an improved classifier. The paper looks at combining multiple models from predictions based on a voting scheme. The author uses stacking [Wol92]

and correspondence analysis [Gre84] to combine multiple models into one. Stacking is the methodology in which the models learned are tested against a data partition that was held out during training. The predictions on the examples in the held out data partition are paired with the correct class and used for further training. Correspondence analysis explores the relationship between the training examples and the classifications of them by the learned model. There will be one row for each training example and for m classifiers and c classes it will have $(m+1)*c$ columns per row, thus correspondence analysis will be very costly for a large data set. Bauer and Kohavi have presented an empirical comparison of Bagging, Boosting and Variants [BK99]. They found that the best performance came from AdaBoost [FS95]. They found Bagging to increase performance without any degradation. Boosting was found to be affected by noise.

The paper [SB99] has proposed a "Distance Metric" methodology, wherein the trees are grown over the subsets of data and a maximum likelihood estimate of a central tree is estimated. For instance, if there are 10 trees grown, each tree is a potential candidate for a central tree. For each such tree, 200 neighboring trees are grown by adding a split at a node or removing a split at a node. The tree with maximum log-likelihood is chosen. After all the trees have been tried, the tree structure with maximum likelihood estimate is reported. They do an experiment where $n = 13$ and each n is a clinical trial at a different hospital. Only continuous attributes are in these trees. The performance is better than with individual trees or a tree grown on full data. This approach will not work well unless the distribution of tree shapes is unimodal.

To deal with large data sets, Provost [FPO99] proposes sampling the data. The point is do we really need all the data? Provost strongly believes that all the data is not needed for learning. He says the learning curve needs to be observed [FPO99]. After some time the learning reaches a saturation point in terms of accuracy, and

adding more examples for the learning phase is not helpful. Bailey et al in [BCG⁺99] talk about the issue of the importance of all attributes in an example. They propose the idea of vertical partitioning, and use only the attributes that carry enough information content as those attributes would be used by the classifier. The rest of the attributes may not be needed.

Most machine learning algorithms have an inherent assumption that all the errors have the same cost. In most real world problems, resultant errors have different types of cost. For example in a breast-cancer detection algorithm, the cost of not identifying the person having micro-calcifications is a very costly error. Domingos [Dom99] has proposed a method *MetaCost* for making error-based classifiers cost-sensitive. If the cost of misclassifying the examples of a class becomes more expensive relative to the other classes, the decision boundaries of that class will expand into the regions of the other classes. The MetaCost algorithm basically creates bootstrap samples of the data space and learns separate classifiers for each space. Then, the probability of each class for each example is estimated. The examples are then relabeled with the optimal class by the cost matrix. The paper shows experiments comparing Metacost to C4.5R (a combination of C4.5 and C4.5rules) and stratification.

“Globally, the average cost reduction obtained by MetaCost compared to C4.5R is approximately twice as large as that obtained by undersampling, and five times that of oversampling.”[Page 155, [Dom99]]

Many metrics/algorithms have been proposed to associate a performance/goodness measure with a rule(s). Bayardo and Agrawal [BA99] propose that the best rule adhering to any of the metrics should lie along a support/confidence border. It is difficult to come up with a metric that quantifies the goodness of a rule. The concept proposed by the paper is based on a partial ordering of rules defined both in terms of rule confidence and support.

CHAPTER 4

RULE COMBINATION I

Chapter 3 discussed various techniques for combining classifiers or creating a single classifier in parallel. This chapter delves into a strategy of combination of decision trees to create a single classifier model. The goal is to improve the accuracy and coverage of the resulting learned model. Disjoint subsets of very large data sets are created. Decision trees are learned on each of the disjoint subsets, decision trees can be learned in a fast, cost-effective manner. Taking the learning time into account, decision trees learn quickly [Qui92, BFOS84] compared to the other learning algorithms such as RL [FB85] and Ripper [Coh95]. Decision trees can be translated into an equivalent set of rules, which are a more expressive form. Rules can be combined by taking a merge of N rule sets into one rule set. In doing so the coverage may not suffer but the accuracy could be affected; rules from different rule sets may conclude different decisions for a training sample. That is, the rules may be in conflict [Wil90].

4.1 What are conflicts?

Williams [Wil90] has defined two rules to be conflicting, if they have all but one condition as common, but lead to different classification decisions. For instance, the rules in Figure 4 conflict. The two rules have all but one condition the same. The different conditions are $z \leq 20$ and $w > 10$ in Rule 1 and Rule 2, respectively. There is an overlap in the decision space because of the common conditions.

A pair of conflicting rules may fire for the same example, with one of them belonging to the right class. The rule will be classified depending upon the order of firing

```

Rule 1:
  x < 4
  y > 8
  z <= 20
      -> class A
Rule 2:
  x < 4
  y > 8
  w > 10
      ->class B

```

Figure 4. Example of two rules in a conflict. These two rules have all but one condition as same.

```

Rule 1:
  Plasma glucose concentration > 161.000   -> class 1 [82.9]
Rule 2:
  Plasma glucose concentration > 127.000
  Two Hour serum insulin <= 156.000
  Body mass index > 26.800
  Body mass index <= 29.800   -> class 0 [100.0]

```

Figure 5. The attribute Plasma glucose concentration has an overlapping range of values. This is a potential conflict though all the other conditions are different.

of the two rules, if only one rule firing is allowed. This conflict should be resolved, with the example falling under the “correct” rule.

The above condition defining a conflict is a very restricted form. There can be other rules which may be differing in more than one condition but at the same time be in a conflict. The rules which are not mutually exclusive are possible conflicting rules.

Figure 5 is another instance of two rules being in a conflict. In this particular case, there is only one common condition, Plasma glucose concentration, between the

Rule 1:
 sepal length in cm > 5.2 -> class Iris-viginica [100.0%]

Rule 2:
 petal width in cm <= 0.3 -> class Iris-setosa [100.0%]

Figure 6. Rules having no conditions in common.

Table 1. Types of conflicts.

Conflict Type	Example
The first is as in Williams dissertation [Wil90]; two rules having all but one condition same and deriving different classes are defined to be in conflict.	Figure 4
The second is that n conditions involve different attributes and there are m other conditions that overlap to some degree (i.e. are not mutually exclusive). In this case, $0 \leq m$ and $0 \leq n$.	Figure 5 Figure 6

two rules. This pair of rules is a potential conflict. Figure 6 is another instance of a conflict. Rule 1 and Rule 2 each have one completely different attribute and could possibly fire for the same examples as they are not mutually exclusive.

Thus, mutual exclusivity is the necessary and sufficient condition for rules to not be in a conflict. These conflicting rules must be identified and resolved to remove the conflicts. The approach of conflict resolution pursued is an extension of Williams' work [Wil90], where multiple decision trees, each with a different bias, were induced from the same data set.

The various type of conflicts can be summarized in Table 1.

4.2 Conflict resolution

1. *Learn rules*

With training data distributed to N processors, each processor grows a tree using the data it has and converts the tree to rules. Then each processor exchanges rules with every other processor.

2. *Eliminate redundant conditions*

For each rule check if there is any condition that would subsume other condition(s) in the rule. For instance in,

```
x > 8
x > 10
y <= 12
-> Class A
```

Condition $x > 10$ subsumes the condition $x > 8$ and thus would eliminate $x > 8$ giving the following rule.

```
x > 10
y > 12
-> Class A
```

Similarly, another case could be,

```
x <= 8
x <= 6
y > 10
-> Class A
```

Condition $x \leq 6$ subsumes the condition $x \leq 8$, giving the final rule

```
x <= 6
y > 10
-> Class A
```

3. *Conflict identification*

The theory of conflict identification has been discussed in the Section 4.1 and enumerated in Table 1.

4. *Eliminate the conflicts*

Conflict resolution should get rid of all the conflicts, while keeping the coverage and improving or maximizing the accuracy. The attempt is towards maximizing the individual coverage of each of the conflicting rules while defining a new rule to cover the conflicting set of examples. Different strategies were developed to resolve the different types of conflicts that may be encountered. The first conflict strategy is similar to the one proposed by Williams in [Wil90]

- (a) Strengthen each of the rules in conflict, by negating the differing condition in the rules (negating the conditions means the $>$ sign becomes \leq and vice-versa).

r1: cond and cond1 \rightarrow class1

r2: cond and cond2 \rightarrow class2

r1 and r2 are in conflict, so we add two new strengthened rules nr1 and nr2.

nr1 : cond and cond1 and \neg cond2 \rightarrow class1

nr2 : cond and cond2 and \neg cond1 \rightarrow class2

For example,

r1: p4 $>$ 0.5 and p3 \leq 0.7 \rightarrow C1

r2: p4 $>$ 0.5 and p2 $>$ 1.0 \rightarrow C2

are in conflict since only one condition differs in both the rules and they both result in different classes.

Strengthening r1 and r2 we get,

nr1 : $p4 > 0.5$ and $p3 \leq 0.7$ and $p2 \leq 1.0 \rightarrow C1$

nr2 : $p4 > 0.5$ and $p3 > 0.7$ and $p2 > 1.0 \rightarrow C2$

These two rules are now mutually exclusive and will only fire for the examples exclusive to each. In the process of strengthening the rules, the coverage for the examples satisfying the conditions $p4 > 0.5$ and $p3 \leq 0.7$ and $p2 > 1.0$ is lost. The theory is when rules conflict, the set of examples falling in the conflicting set are the examples satisfying the common condition(s) and as well as the differing condition(s). When the rules are strengthened this conflicting set of examples has no rule defined for them as the original rules have the differing conditions negated, therefore the scope is reduced.

For the two rules, r1 and r2 in a conflict, the conflicting set of examples are the examples satisfying the conditions cond and cond1 and cond2. Thus, the next step is to restore the lost example coverage. It may just happen that the conflict examples are of one class (class1), then introduce the new rule as

ncr : cond and cond1 and cond2 \rightarrow class1

For the example being considered, it would be

ncr : $p4 > 0.5$ and $p3 < 0.7$ and $p2 > 1.0 \rightarrow C1$

If the examples that satisfy the conditions are of mixed classes, then find a new condition (for continuous attributes, an attribute and a split point) that partitions the examples into pure subsets. If one cannot be found, take the best test and add it to the rules. This may be continued until two pure subsets of examples can be created or all the attributes and their

split points have been exhausted. This will lead to the creation of two or more new rules for the conflicting set of examples, one or more for each of the classes.

$\text{ncr1} : \text{cond and cond1 and cond2 and cond3} \rightarrow \text{class1}$

$\text{ncr2} : \text{cond and cond1 and cond2 and } \neg \text{cond3} \rightarrow \text{class2}$

where cond3 is the new attribute added to create two rules with opposing splits ($>$ and \leq).

Thus, from a pair of conflicting rules a set of rules is created, keeping the same coverage as the two but improving the accuracy as the conflicting set of examples would be resolved.

- (b) For the case in which n conditions involve different attributes and there are m other conditions which overlap to some degree (i.e. are not mutually exclusive). In this case $0 \leq n$ and $0 \leq m$. In the case that $m > 0$ (like in the above where $m = 1$), adjust one or more of the partially overlapping conditions based on the conflict set to resolve the conflict. Efficiency here is coverage, which should not suffer as a result of conflict resolution, if the conflicts cannot be resolved efficiently, then strengthen the last condition in the two rules and create another rule(s) as explained in step 4(a).

A more complex problem is when one rule overlaps with an entire interval of another rule. Figure 7 gives an example where Rule 4 and Rule 6 conflict. For this case strengthen Rule 4 but this will exclude some examples (for instance add the condition $\text{petal width in cm} \leq 1.500$). Now a new rule is needed to cover lost examples. The new rule will be created by taking a combination of the common conditions between the two rules and adding one or more condition(s) as explained in step 4(a), unless another rule for the class Iris-viginica covers the excluded examples. Here we have Rule 5

```

Rule 4:
    petal length in cm > 4.900
    petal width in cm > 0.400
    -> class Iris-viginica [100.0]
Rule 5:
    petal width in cm > 1.500
    -> class Iris-viginica [100.0]
Rule 6:
    petal length in cm > 4.900
    petal width in cm > 1.500
    petal width in cm <= 1.700
    -> class Iris-versicolor [66.7]

```

Figure 7. A conflicting case. Rules 4 and 6 conflict.

which does cover them. It is in conflict with Rule 6 though, and this will be resolved.

In the case when there are no conditions in common between the two rules, strengthen the last condition and create a rule(s) for the lost coverage as explained in (1).

4.3 Experiments

The experiments were run on the Iris data set[BM98] and Pima Indians Diabetes data set[BM98]. The Iris data set has four continuous-valued attributes, three classes and 150 examples. The Pima Indians Diabetes data set has eight continuous-valued attributes, two classes and 768 examples. The experiments were performed simulating a 2-processor and 3-processor partition for the Iris data set and a 2-processor partition for the Pima data set. The experimental results were obtained using a 10-fold cross-validation. For the Iris data set the train set was 135 examples and the test set was 15 examples for each of 10 folds. For a 2-processor partition the train set was split in the middle to give two subsets of 67 and 68 examples each. For each fold, decision trees were generated using C4.5, converted into rules and then combined into

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

Results	C4.5 %	UnPruned %	Pruned %
Accuracy	95.3	96	96
Standard Deviation	± 6.01	± 5.33	± 5.33

Table 3. Results on the Iris data set using 10-fold cross-validation for a 3 processor partition.

Results	C4.5 %	Unpruned %	Pruned %
Accuracy	95.3	96.67	96.67
Standard Deviation	± 6.01	± 5.37	± 5.37

a single model using the conflict resolution approach discussed in the previous section. The model generated was used to test against the unused test set. The 3-processor partition led to 3 partitions of 45 examples each. Again trees were generated over the three partitions, converted into rules and combined together using conflict resolution. Tables 2 and 3 depict the accuracy and standard deviation results.

The accuracy is slightly better than the C4.5 decision trees (but not statistically significant) for both the pruned as well as unpruned trees. Since no pruning was done with the default pruning parameter, pruning was done with certainty factor = 1. Still, pruning was done in only 4 of the decision trees generated. The 3-processor partition resulted in 1 less error.

The results from 10-fold cross-validation on the Pima Indians Diabetes data set for a 2-processor partition are shown in Table 4. The Pima data set has 768 examples, giving 691 examples for training and 77 examples for testing. The train set was divided into subsets of two of 345 and 346 examples each.

The experiments with the Pima data set also indicate a slight increase in performance. Neither performance increase is of statistical significance.

Table 4. Accuracy on the Pima Indian Diabetes data set using 10-fold cross-validation for a 2-processor partition.

Results	C4.5 %	UnPruned %	Pruned %
Accuracy	73.38	74.94	73.64
Standard deviation	± 4.66	± 2.93	± 4.12

4.4 Summary

In the approach discussed in this section, training data sets are partitioned into disjoint subsets and decision trees learned on each. Decision trees are then converted into rules, rules are combined into a single model and used for classification. Figure 8 depicts the procedure graphically.

When conflicts are identified, the conflicting set of examples need to be exchanged between processors for re-learning. The data communication will be a cumbersome process. This procedure entails communication of data as well as rules among processors, the data or example in conflict must be communicated to a single processor for conflict resolution. This will affect the scalability of the algorithm and run time, once we go to giga and tera bytes of data. The increase in the data communication costs would become prohibitive. Hence we reluctantly abandon this promising approach. Taking these points into perspective, an alternate algorithm was tried for conflict resolution, which does not involve data communication. This algorithm only entails rule communication and is discussed in the next chapter.

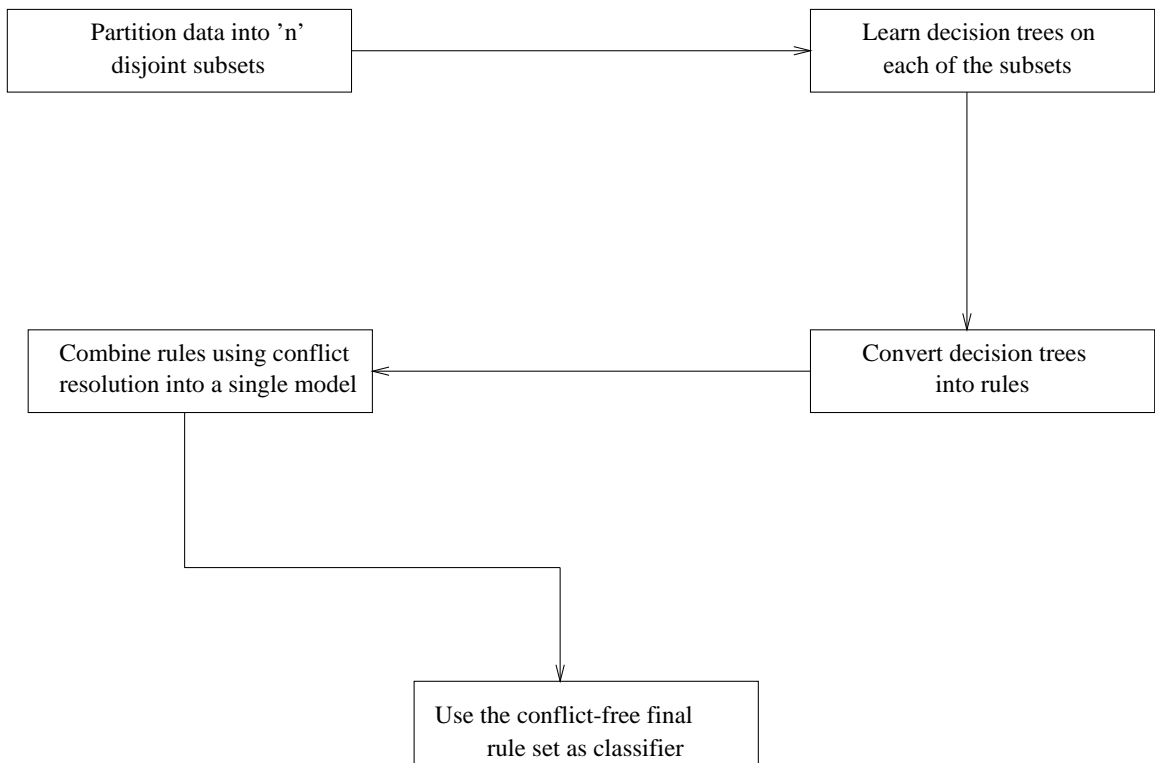


Figure 8. Summary of the procedure.

CHAPTER 5

RULE COMBINATION II

5.1 Introduction

With the conflict resolution strategy proposed in the previous chapter, the drawback limiting scalability is the amount of data communication. In the approach focused upon here, rules are created on N processors. Each of the N processors contains a disjoint set of labeled training data. These distributed rules are combined using some heuristic into a single rule set. Rules are created directly by traversing pruned decision trees on each processor (with the obvious optimization of removing redundant tests). The rule combination is related to work by Williams [Wil90] and Provost [FJP96].

Each rule that is created will have associated with it a measure of "goodness," which is based on its accuracy and the number and type of examples it covers. The normalized version of the certainty factor metric [Qui92, FJP96] to determine the accuracy of a Rule R over an example set E is :

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

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 learned 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 testing time on other processors and some communication time required to send it and its current TP/FP count to another processor.

5.2 Algorithm details

1. Partition the data and distribute disjoint subsets of data over the N processors.
2. Learn a decision tree on each of the partitions and prune it.
3. Convert the learned decision tree into rules.
4. Eliminate the conditions which are subsumed by the other conditions.
5. Test the rules on the other processors' data set and update E and calculate $\text{acc}(R,E)$ for each rule R based on metric (1). This is an estimate of the global performance of each rule.
6. Set a threshold t as discussed in the previous section. If the performance of the rule is less than the threshold of the class it classifies, the rule is removed.
7. Identify Conflicts. The theory behind conflict identification is discussed in Chapter 4.
8. For every pair of rules identified to be in a conflict, remove the worse performing of the pair. Any uncovered examples will be assigned to the majority class.

Provost and Hennesey in [FJP96] have shown that any rule which is acceptable by certain metrics on the full data set will be acceptable on at least one disjoint data set.

They found that the set of rules created after merging the rule sets built on different processors was a superset of the set of rules created on the full data set.

In [HCBK99], it is illustrated that rules learned over two partitions of the data set and then merged do not necessarily contain any rule learned over the full data set. Figure 9 shows an example. It suggests that the accuracy of the merged rule sets may differ from the rules learned on the complete data set.

Conflict resolution is needed for the rules having an overlap in the example space. As rules are tested against examples on the processors, their calculated accuracy under the chosen metric may change. It may just happen that of the two conflicting rules, one may have its performance estimate less than the chosen threshold t and thus the conflict is resolved. However, partially overlapping rules may stay and will potentially cause misclassifications. They need to have the conflict resolved. The conflict may be resolved as explained in Chapter 4. Another possibility is to remove the lower performing of the two, which is how it is done in our current system. Two approaches were experimented with for discarding the lower performing rule.

1. In one approach rules are ordered by performance estimates and conflicts resolved in a pairwise fashion. In this approach, as soon as a rule is in a conflict and has a worse estimate than another rule, it is removed from the rule set. This is called the *pairwise conflict resolution approach*.
2. The second conflict resolution strategy is based on the heuristic that every rule is compared against every other rule before the worse performing rule is discarded and the final rule set is formed. This is called the *global conflict resolution approach*.

In the pairwise approach, the rules were removed immediately on conflict identification and didn't remain during examination of the other rules. This way some other poor performing rules remained in the final classification model, which really didn't

```

Example label: A B C D E F
Att1:          -----
              1 4 3 2 5 6
Att2:          1 3 2 4 2 1
Class:         C1 C1 C1 C2 C2 C2

```

```

Sorted Class Label      Class  Label

Att1: 1  C1  A  Att2: 1  C1  A
      2  C2  D           1  C2  F
      3  C1  C           2  C1  C
      4  C1  B           2  C2  E
      5  C2  E           3  C1  B
      6  C2  F           4  C2  D

```

Processor 1 Gets the examples BCD and produces rules:

```
Att1 > 2 then C1
```

```
Att1 <= 2 then C2
```

OR

```
Att2 > 3 then C2
```

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

Processor2 get the examples AEF and produces rules:

```
Att1 <= 1 then C1
```

```
Att1 > 1 then C2
```

From the full train set we get:

```
Att1 > 4 then C2
```

```
Att1 <= 4 and Att2 <= 3 then C1
```

```
Att1 <= 4 and Att2 > 3 then C2
```

Figure 9. 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.

deserve to stay. In the current model every rule is compared against every other rule even if it is identified to be in a conflict and deserves removal. The following example will explain the global conflict resolution strategy.

Rule 1: $X \rightarrow \text{class A}$ [96]

Rule 2: $Y \rightarrow \text{class B}$ [90]

Rule 3: $Z \rightarrow \text{class C}$ [80] where X, Y, Z are the variable placeholder for conditions and the values in [] are the performance estimates of the rule based on test examples for a chosen metric.

First let us consider the pairwise conflict resolution when the rules are ordered by performance. The final rule set based on conflict resolution will be :

Rule 1: $X \rightarrow \text{class A}$ [96]

Rule 3: $Z \rightarrow \text{class C}$ [80] Rule 1 and Rule 2 conflict, and Rule 2 also Rule 3 conflict. Rule 1 has a higher performance than Rule 2, thus Rule 2 fails in comparison with Rule 1. Rule 2 has a better performance than Rule 3. Thus, Rule 3 fails in comparison with Rule 2. Finally, Rules 2 and 3 will be removed. The final rule set will look like

Rule 1: $X \rightarrow \text{class A}$ [96] Another example to illustrate the global conflict resolution strategy.

Rule 1: $X \rightarrow \text{class A}$ [96]

Rule 2: $W \rightarrow \text{class B}$ [65]

Rule 3: $Y \rightarrow \text{class A}$ [60]

Rule 4: $Z \rightarrow \text{class B}$ [55] In the above scenario, rules 1, 2 and 4 conflict with each other and rules 2, 3 and 4 also conflict with each other. Rule 1 is the only one that survives, Rule 3 gets killed as Rule 2 performs better. Thus the final rule set looks like,

Rule 1: $X \rightarrow \text{class A}$ [96] Considering another set of rules as follows

Rule 1: $\text{cond1 and cond2} \rightarrow \text{class A}$ [96]

Rule 2: $\text{cond1 and cond3} \rightarrow \text{class B}$ [65]

Rule 3: $\neg \text{cond1 and cond4 and cond5} \rightarrow \text{class A}$ [70]

Rule 4: $\neg \text{cond1 and cond4 and cond6} \rightarrow \text{class B}$ [75] Rules 1 and 2 form one conflicting pair of rules and rules 3 and 4 form another. Rule 1 performs better than rule 2 and rule 3 performs better than rule 4. Both rules 1 and 4 stay and the final rule set looks like:

Rule 1: $\text{cond1 and cond 2} \rightarrow \text{class A}$ [96]

Rule 4: $\neg \text{cond1 and cond 4 and cond 6} \rightarrow \text{class B}$ [75] The next section contains some experimental results based on the strategies discussed above.

In the process of rule removal, some coverage may be lost. The examples unclassified by any rule in the final rule set will be given the default class. The default class would be chosen based on the following conditions:

1. The class with the majority representation in the training set becomes the default class.
2. In the case of classes having an equal representation in the train set, the following could be the criteria.
 - (a) The class that got completely eradicated in conflict resolution.
 - (b) If none got eradicated, the class is based on representation in the final rule set. The class with a minimum coverage in the final rule set becomes the default class. If M_i is the coverage of a rule and N_i is the number of misclassifications, the class with the least $M_i - N_i$ becomes the default class, where i is number of the rule.

This way the class with a minimum/no representation in the rule set becomes the default class as the other classes are assumed to have good coverage in the rule set.

Figure 10 shows a pair of rules induced from two partitions of the Iris data set. The numbers shown in the figure are the local and global estimates of each rule. The

```

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]

```

Figure 10. 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.

convention followed is $[x/y]$ where x is the false positives and y is the total classified. The first bracket is the local estimate and the second bracket is the global estimate. Both rules perform well, have performance greater than the chosen threshold t , but are in conflict. The rule ordering here will make a difference. So, the conflict needs to be resolved.

5.3 Experiments and discussions

5.3.1 Conflict resolution approaches

Experiments were done using 10 fold cross validation [WGT90]. The experiments on the Iris Data set and Pima Indian Diabetes Data set were performed using the conflict resolution strategies discussed in the previous section. The results on the Iris data set are tabulated below in Tables 5 and 6. The second column in the tables is the result of 10-fold cross-validation using C4.5 decision trees on the entire data set, the third column is the train set split into 'n' number of partitions and the conflicts resolved in a pairwise manner and the fourth column is the train set split into 'n' number of partitions and the conflicts resolved globally.

Table 5. Results on the Iris data set using 10-fold cross-validation for a 2 processor partition, using Pairwise and Global conflict resolution.

Results	C4.5 %	Pairwise %	Global %
Accuracy(unpruned)	95.33	94.67	95.33
Standard Deviation(Unpruned)	± 6.01	± 5.21	± 4.99
Accuracy (Pruned with cf = 1)	94	95.33	95.33

Table 6. Results on the Iris data set using 10-fold cross-validation for a 4 processor partition, using Pairwise and Global conflict resolution and pruning with certainty factor = 1.

Results	C4.5 %	Pairwise %	Global %
Accuracy(Unpruned)	95.33	94	94
Standard Deviation(Unpruned)	± 6.01	± 6.29	± 6.29
Accuracy(Pruned with cf = 1)	94	94	94

Table 7 shows the C4.5 10-fold cross-validation results and Tables 8 and 9 show the comparative results of the two conflict resolution strategies on the Pima Indians Diabetes data set, for the trees grown with a certainty factor of 1.

The results indicate that the global conflict resolution strategy performed the best. Pairwise conflict resolution performed worse since the rules didn't stay until the end to be compared with the other lower performing rules, thereby allowing bad, lower performing rules to survive which caused misclassifications in the test set.

Figure 11 shows the accuracy result of the global conflict resolution strategy on the Pima Indians Diabetes data set for a 10-fold cross-validation.

Figure 12 depicts the standard deviation for 2,4,6,8,10,12 partitions of the Pima Indians Diabetes data set using 10-fold cross-validation.

Figure 13 shows the number of rules on average generated for 10-fold cross-validation on 2,4,6,8,10,12 partitions of the Pima Indians Diabetes data set.

Table 7. C4.5 10-fold cross-validation Accuracy results.

Unpruned %	Default pruning %	Pruned with cf = 1 %
73.38	73.90	73.77

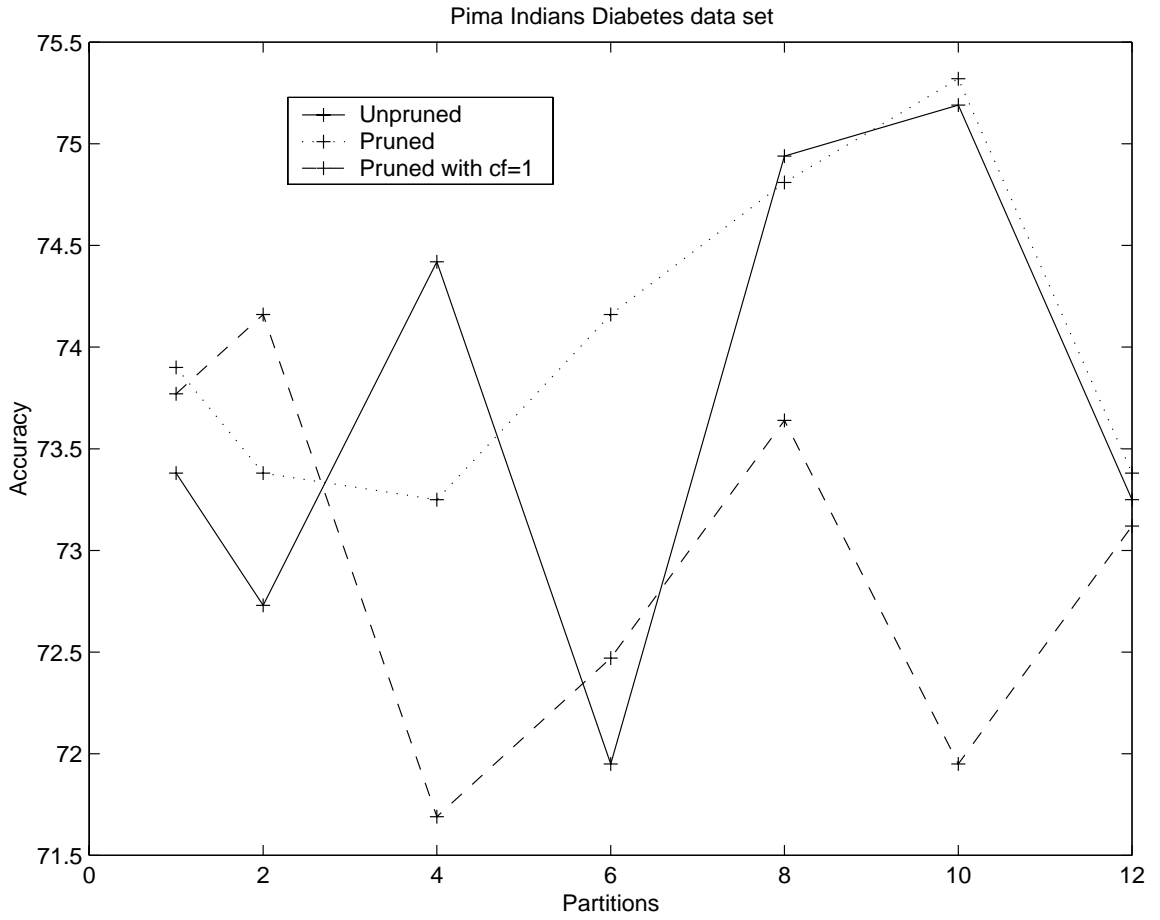


Figure 11. Accuracy results for Pima Indians Diabetes data set using global conflict resolution.

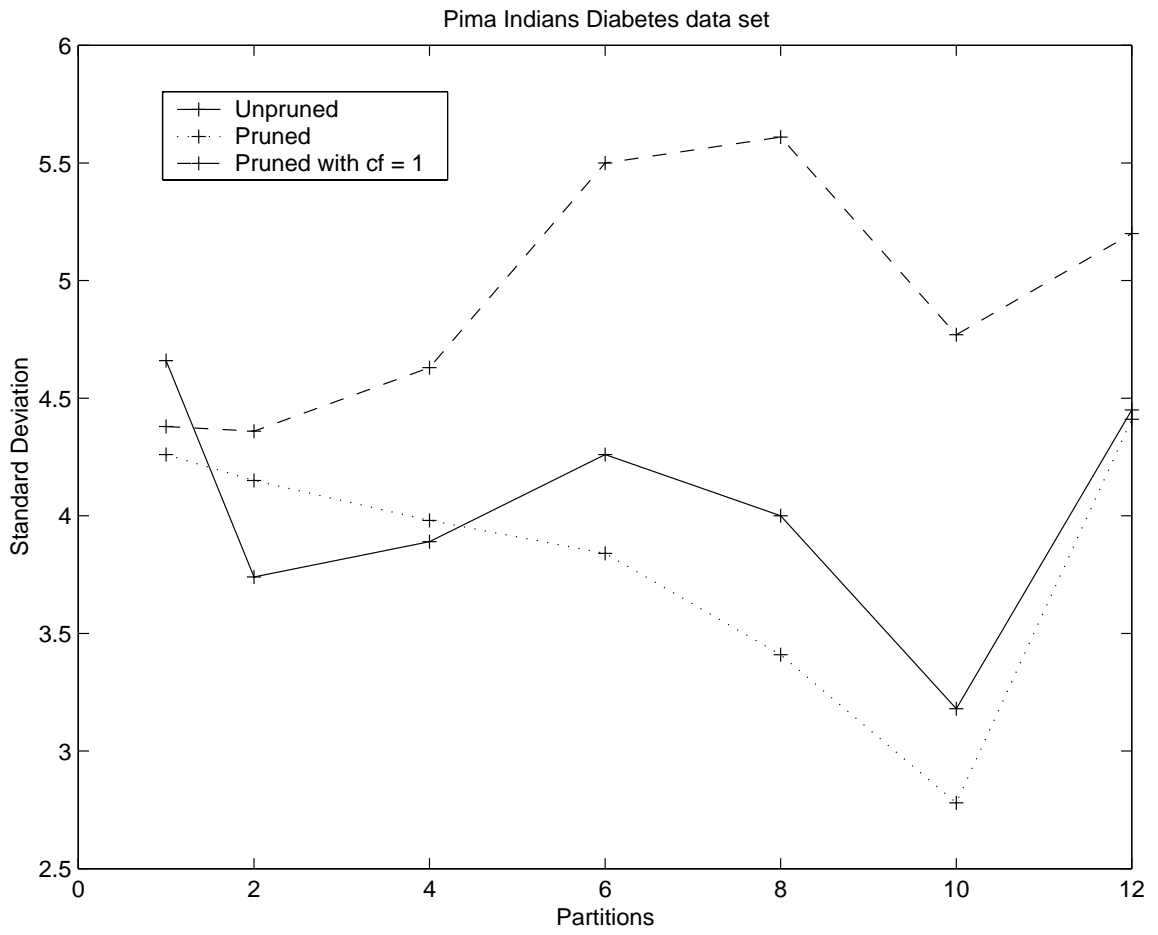


Figure 12. Standard Deviation results using global conflict resolution.

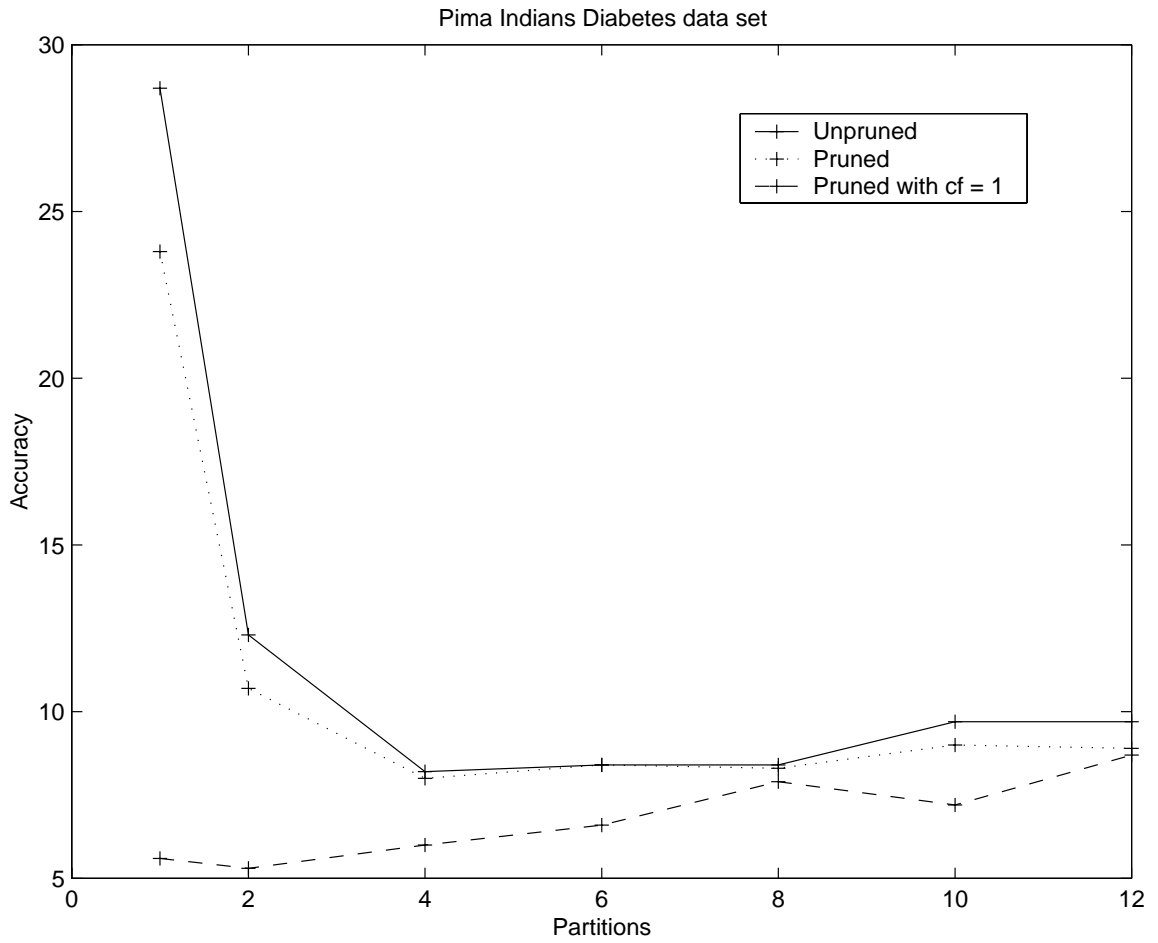


Figure 13. Number of rules graph for the Pima Indians Diabetes data using global conflict resolution.

Table 8. Results on the Pima Diabetes data set using 10-fold cross-validation for a 4,6,8,10,12 processor partition, using Pairwise and Global conflict resolution.

Number of Processor partitions	Pairwise(Pruned cf = 1) Accuracy %	Global(Pruned cf = 1) Accuracy %
4	69.48	71.69
6	72.08	72.47
8	70.00	73.64
10	70.13	72.60
12	67.40	73.25

Table 9. Standard Deviation results on the Pima Diabetes data set using 10-fold cross-validation for a 4,6,8,10,12 processor partition, using Pairwise and Global conflict resolution.

Number of Processor partitions	Pairwise(Pruned cf = 1) Standard Deviation %	Global(Pruned cf = 1) Standard deviation %
4	± 3.27	± 4.63
6	± 5.54	± 5.50
8	± 6.07	± 5.61
10	± 3.68	± 4.77
12	± 6.60	± 5.20

Figures 14 to 19 show the percentage accuracy spread across the folds for 2,4,6,8,10 and 12 partitions of the Pima Indians Diabetes data set using global conflict resolution strategy.

Experiments were also run on a Mammography data set provided by Kevin Woods. This data set has 11183 instances, eight predictive attributes and two classes. The class is binary valued for predicting whether the pixel is normal or has microcalcifications. The predictive attributes represent feature vectors from the image, which are ASCII floating-point numbers. The target attributes (classes) are '1' for normal and '2' for microcalcification. The majority class in this case is class '1', which holds for almost 98% of the entire data set.

Figure 20 depicts the accuracy trend using 2,4,6,8 partitions of the mammography data sets, for a 10-fold cross-validation. The trees were pruned with the default certainty factor and a certainty factor of 1. The partition 1 is the accuracy by C4.5 on the complete data set. The experiments were run for both the global and local

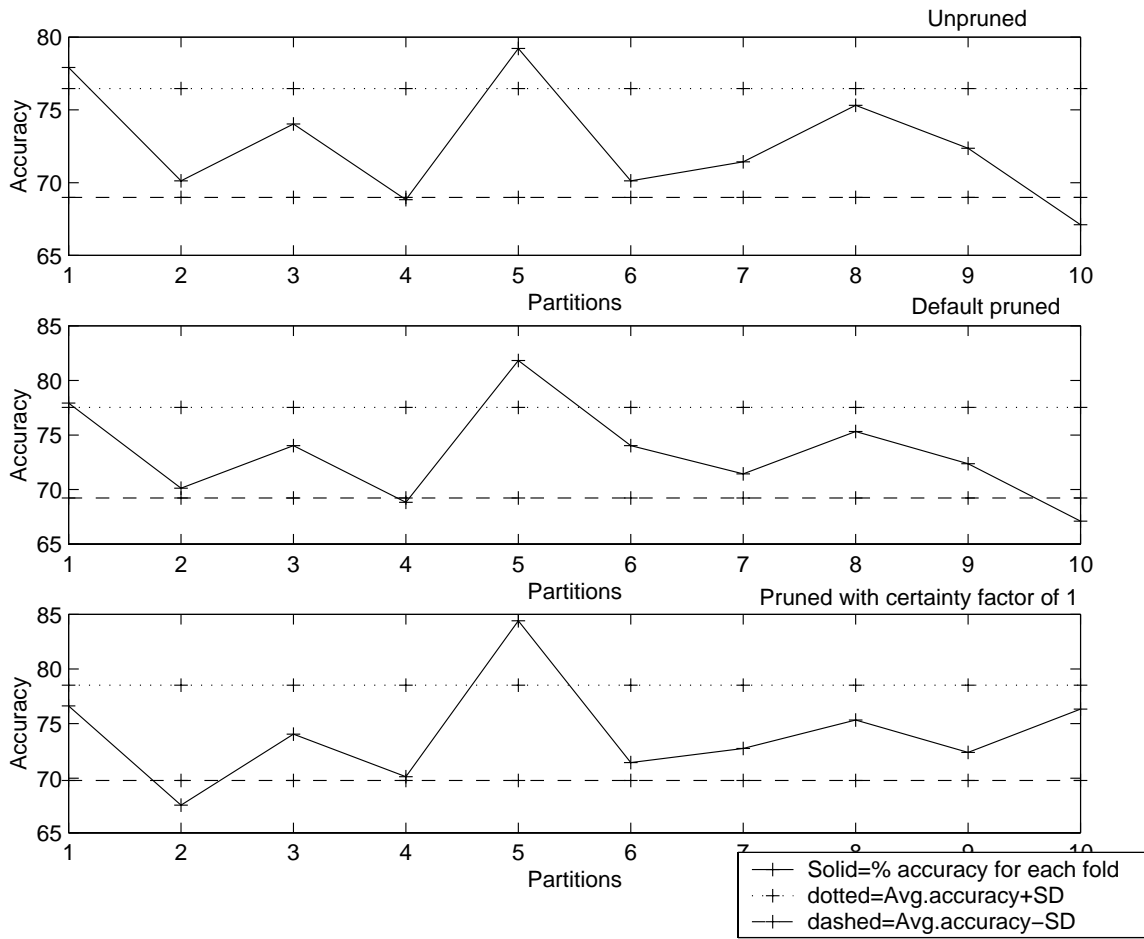


Figure 14. Percentage accuracy spread for 2-processor partition of the Pima data set.

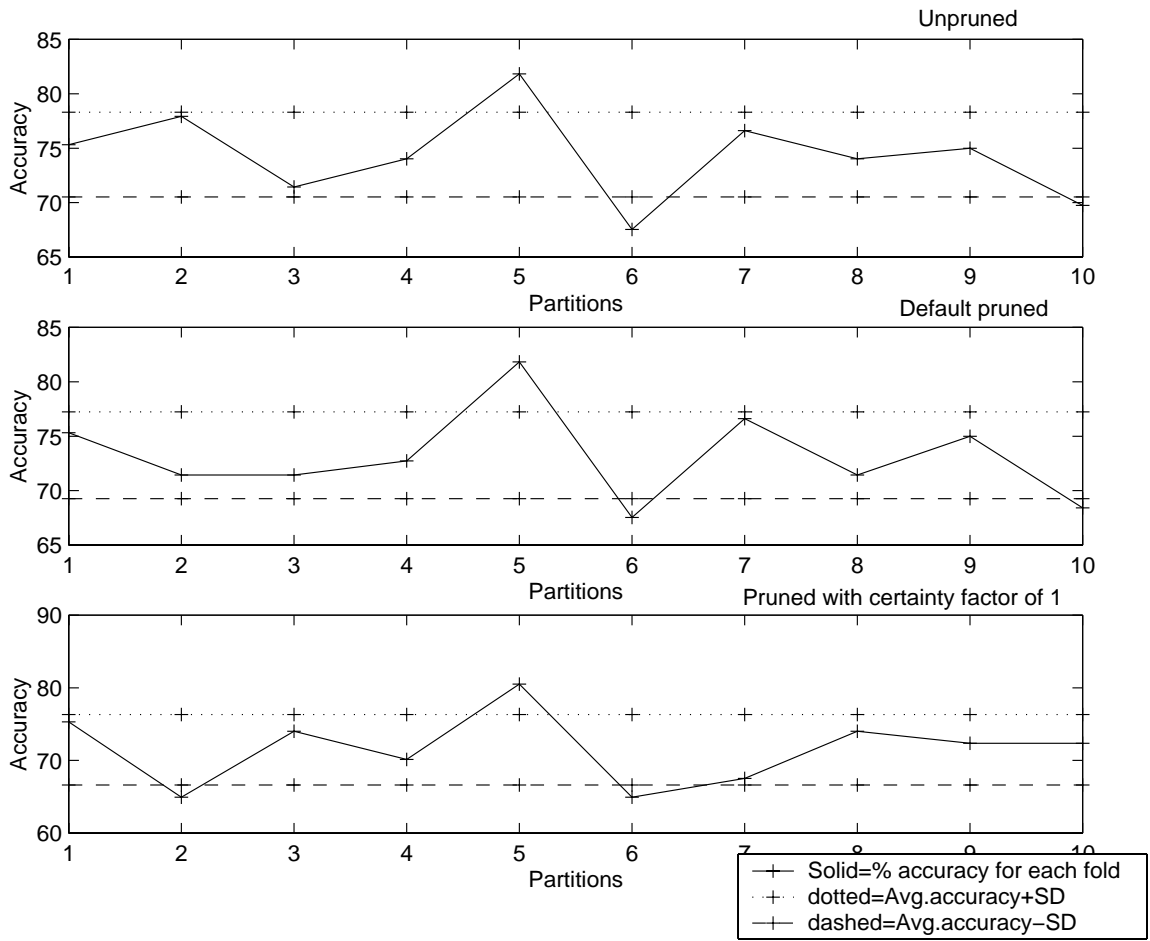


Figure 15. Percentage accuracy spread for 4-processor partition of the Pima data set.

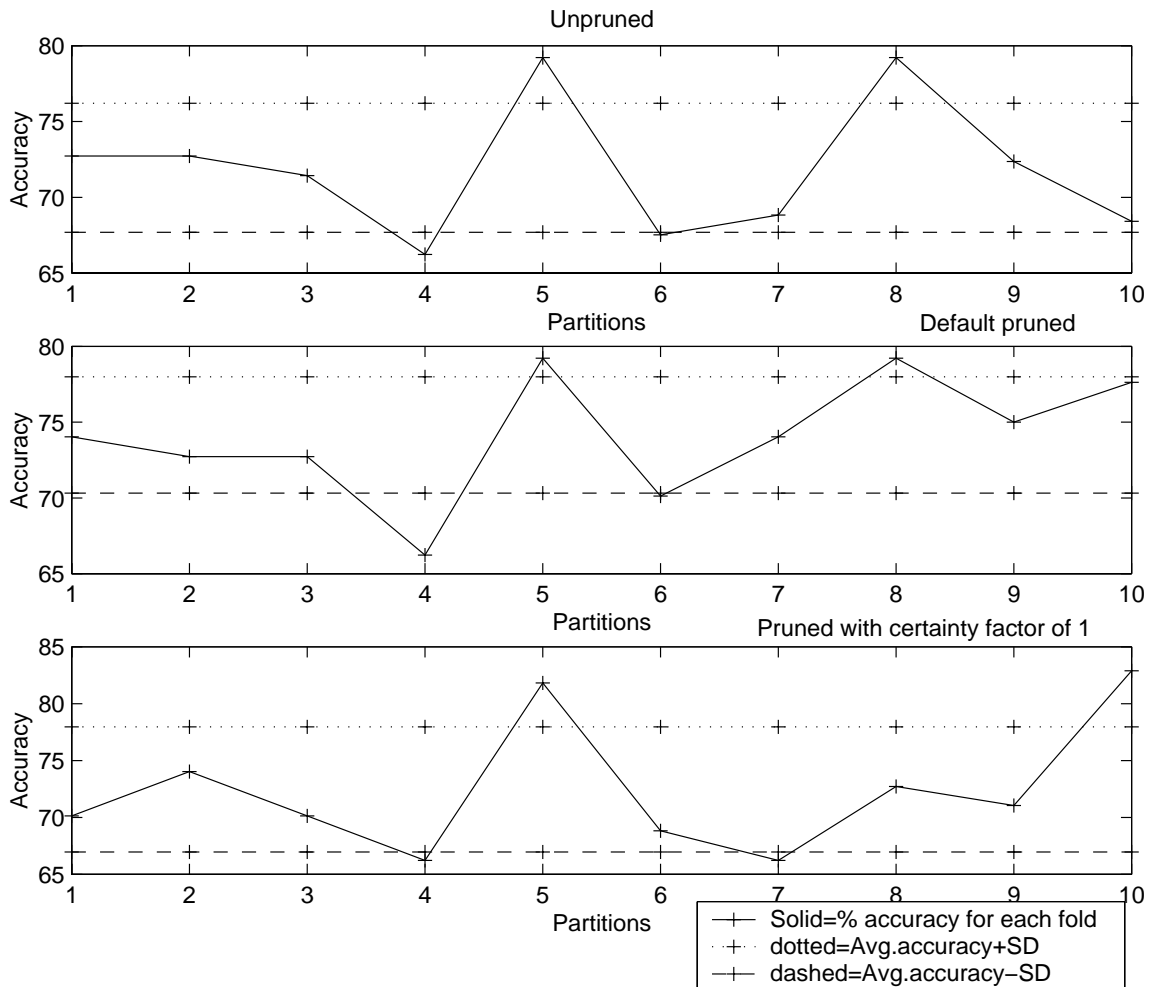


Figure 16. Percentage accuracy spread for 6-processor partition of the Pima data set.

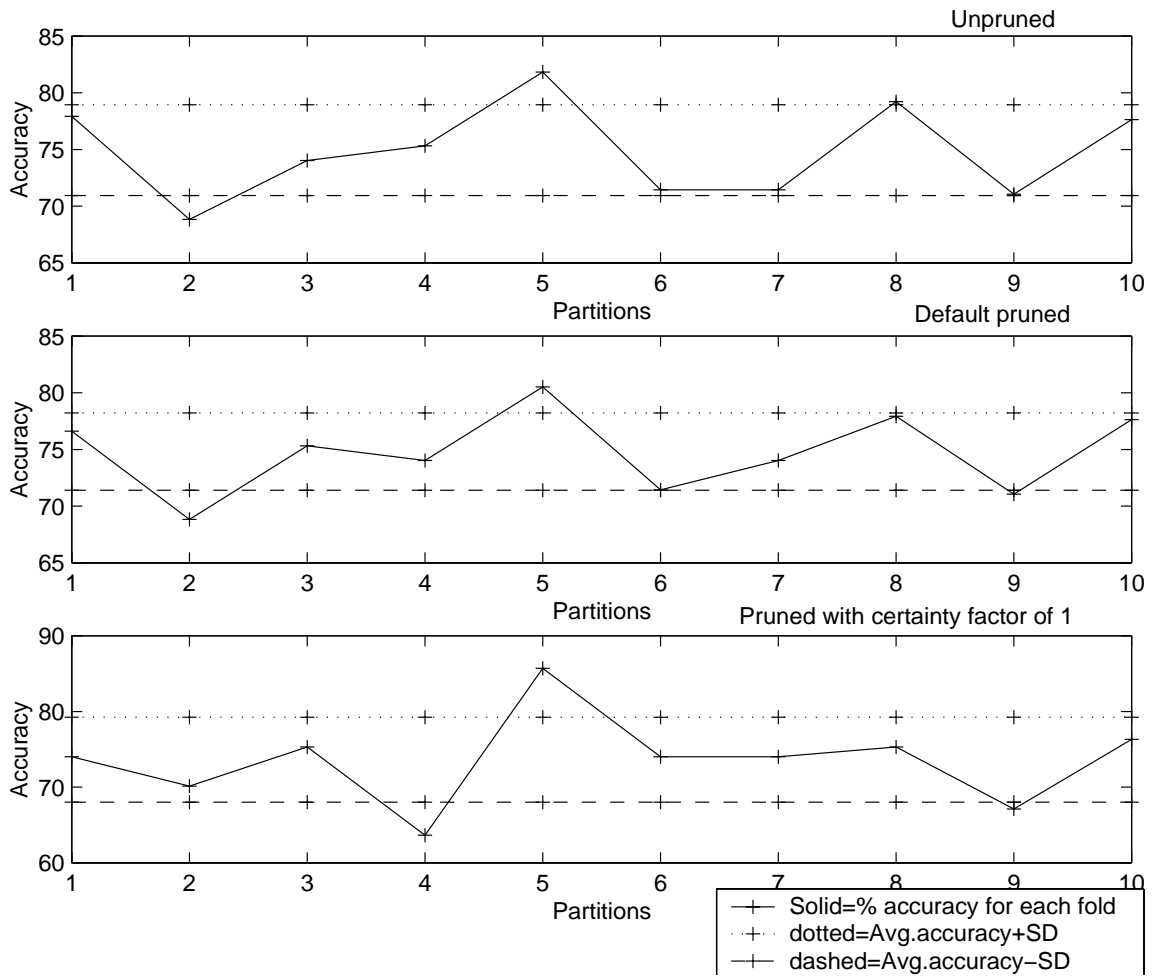


Figure 17. Percentage accuracy spread for 8-processor partition of the Pima data set.

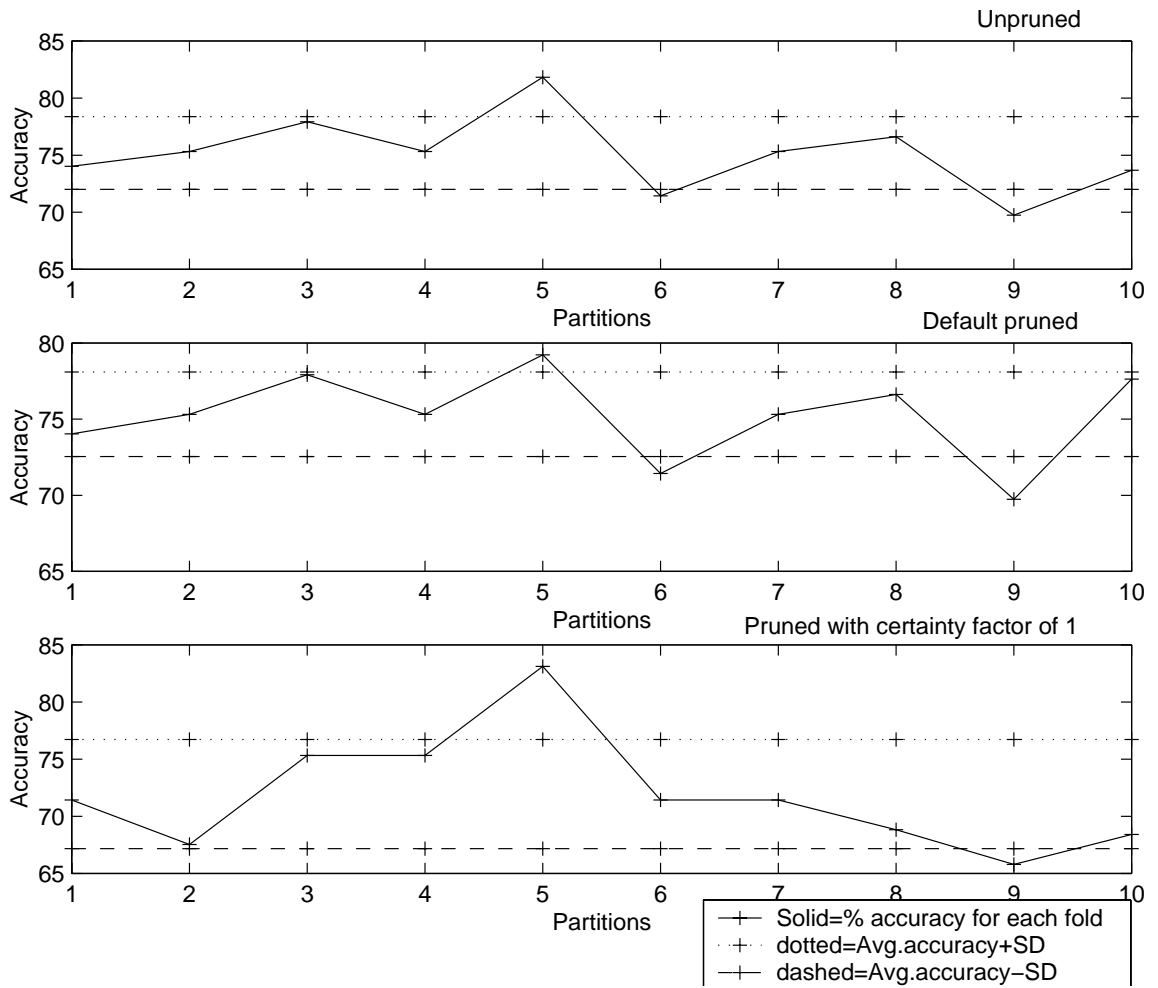


Figure 18. Percentage accuracy spread for 10-processor partition of the Pima data set.

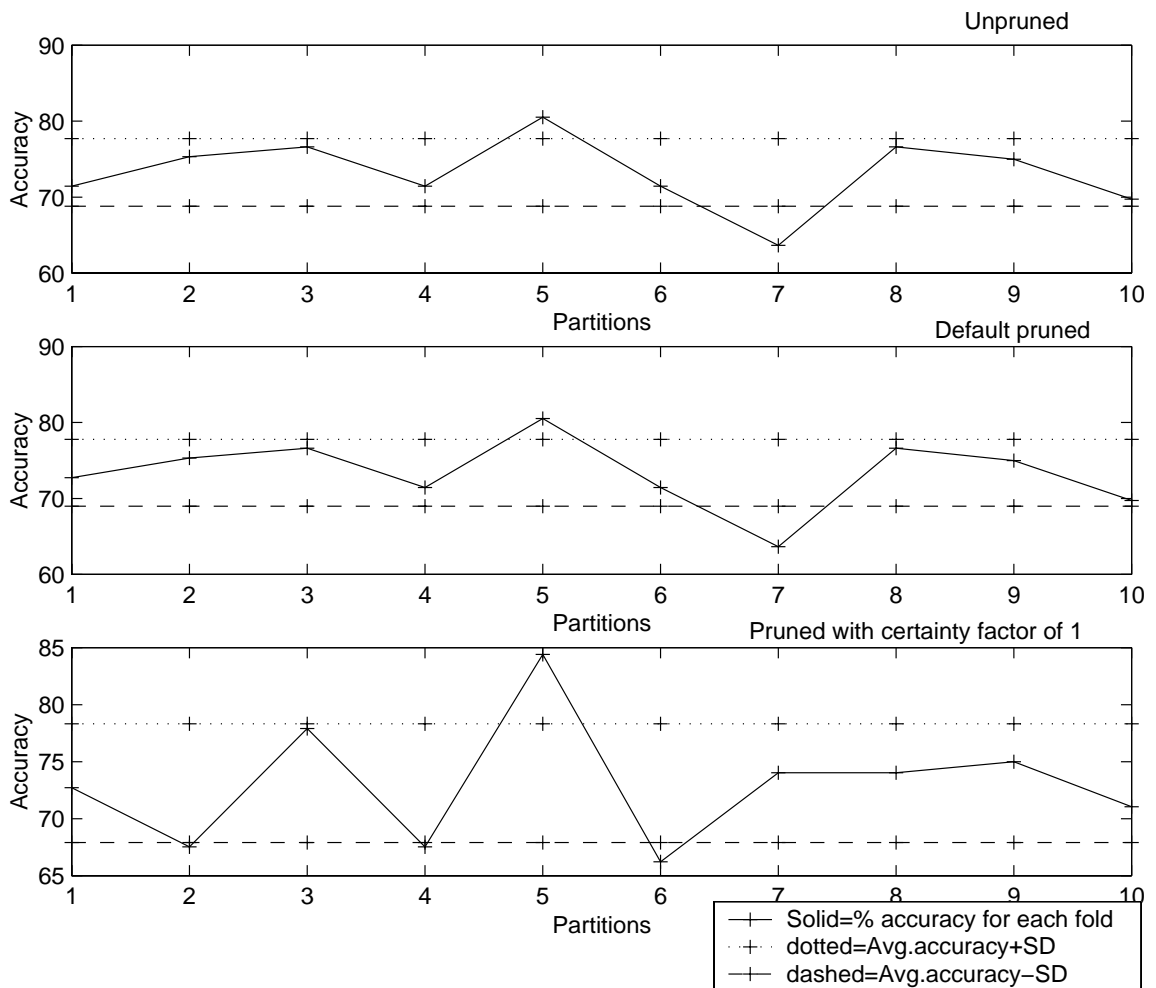


Figure 19. Percentage accuracy spread for 12-processor partition of the Pima data set.

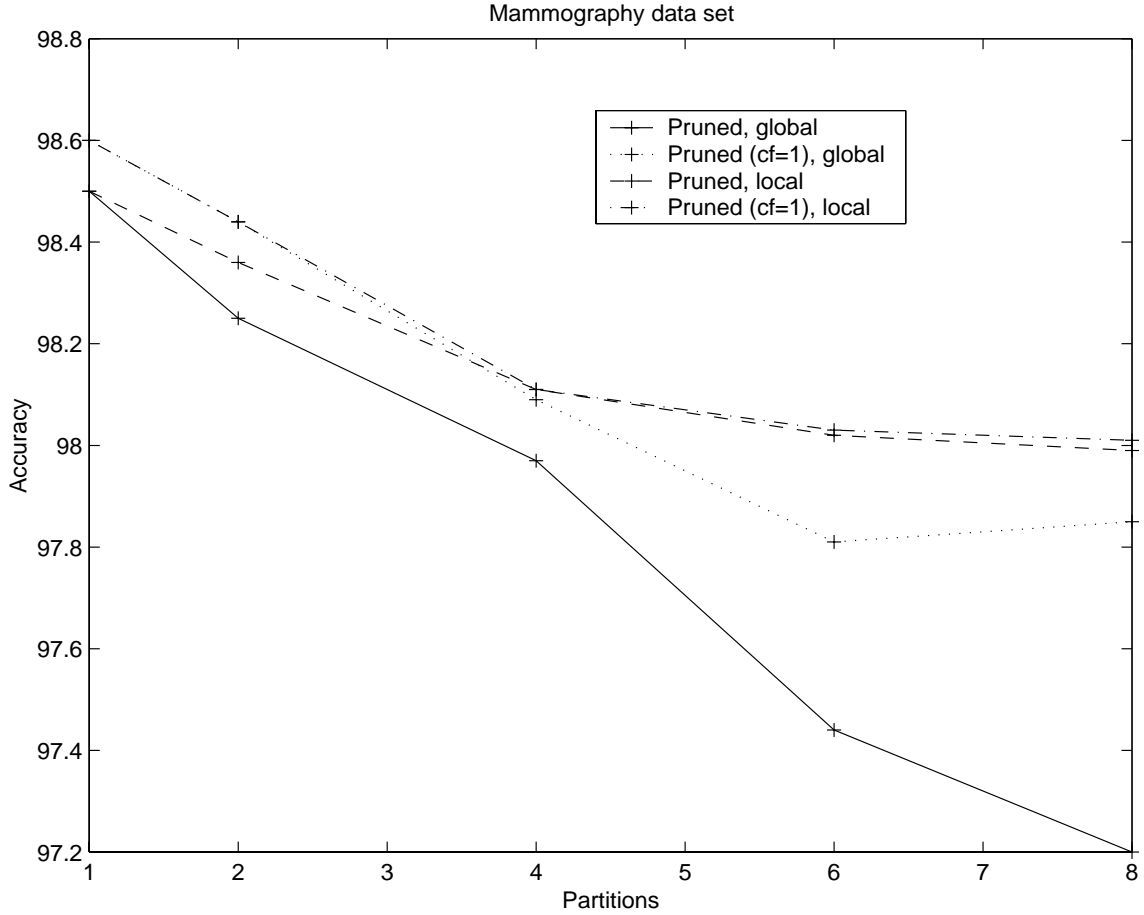


Figure 20. Accuracy results for Mammography data set using local and global performance estimates.

performance estimates on the data set. A local performance estimate is when the rules were assigned the certainty factor based on the performance only on their own data set. For a global performance estimate the rules are evaluated against every other partition before conflict resolution.

The mammography accuracy graph depicts a behavior which is very counter intuitive. The classifier performs better using the local performance estimate than the global performance estimate. This is discussed more in the next section.

The experiments indicate that the global conflict resolution algorithm seems to perform well for the data sets tested except for the mammography data set. The results for the Iris and Pima Indian Diabetes data sets were very close to the results obtained by C4.5 sequentially. The differences have no statistical significance though. We believe that the issue with the mammography data set is more of a metric issue than an algorithmic feature, as will be discussed in the next section.

5.3.2 What is the right metric?

With the mammography data set, the rules combined based on the local performance estimate performed better than the rules combined based on the global performance estimate. Also, in some cases the performance is less than that of guessing the majority class. Guessing the majority class is when a classification is assigned to an example based on the class distribution in the data set. For example, assuming a 2-class data set with one class contributing 66% of the data set and the other class contributing 34% of the data set, if the majority class is guessed by a classifier all the time, the predictive accuracy of the classifier will be 66%.

Careful analysis of the result revealed the problem with metrics for a dataset with a skewed distribution such as the mammography dataset.

The majority class, class '1', has a ρ of 42 and the other class, class '2' a ρ of 0.023.

Consider a case of a class '1' rule having a local performance of 0/57 convention $x/y \rightarrow x$ incorrect(FP) and y total coverage(TP+FP) .

The performance would be

$$(57-0.5)/(57+42*0) = 99.1$$

This rule using a global performance has a 5/565 performance (5 errors covering 565 examples). Since, the ρ is high, this makes the dividing factor big, even for the small number of misclassifications, Giving a performance of,

$$(560-0.5)/(560+42*5) = 72.7$$

Thus the performance decreases for this apparently good class 1 rule. This allowed the other class '2' rules to remove this rule globally. Under local performance considerations the situation was reversed.

On the other hand consider the minority class, class '2' rule. The local performance of a rule is 1/4 using the certainty factor metric, so the performance assigned is

$$(3-0.5)/(3+0.023*1) = 82.7$$

The same rule has global performance of 18/30. Using the certainty factor metric the performance estimate becomes

$$(12-0.5)/(12+0.023*18) = 92.6$$

This rule survived in the global evaluation. This rule actually caused misclassifications in the test set when considered globally but was removed when using local performance. The global performance caused an increase in the TP, thereby increasing the numerator, and even if there were subsequent misclassifications, more FP, the performance was not affected because of the low ρ .

The metric proposed in [Coh95] was also experimented with. This metric is used to guide pruning in Ripper.

$$v * (Rule, PrunePos, PruneNeg) \equiv \frac{p - n}{p + n} \quad (2)$$

where p is the number of positive examples, PrunePos is the number of positive examples in the prune set, n is the number of negative examples and PruneNeg is the number of negative examples in the prune set. It performed better for the mammography data set in a few folds and made the global performance better than the local performance in all but for the 6th fold's case.

The graph in Figure 21 illustrates the result for the mammography data set.

Pima Indian Diabetes data set was also experimented with using Ripper metric. Table 10 compares the result of Pima Indians Diabetes data set using C4.5, the

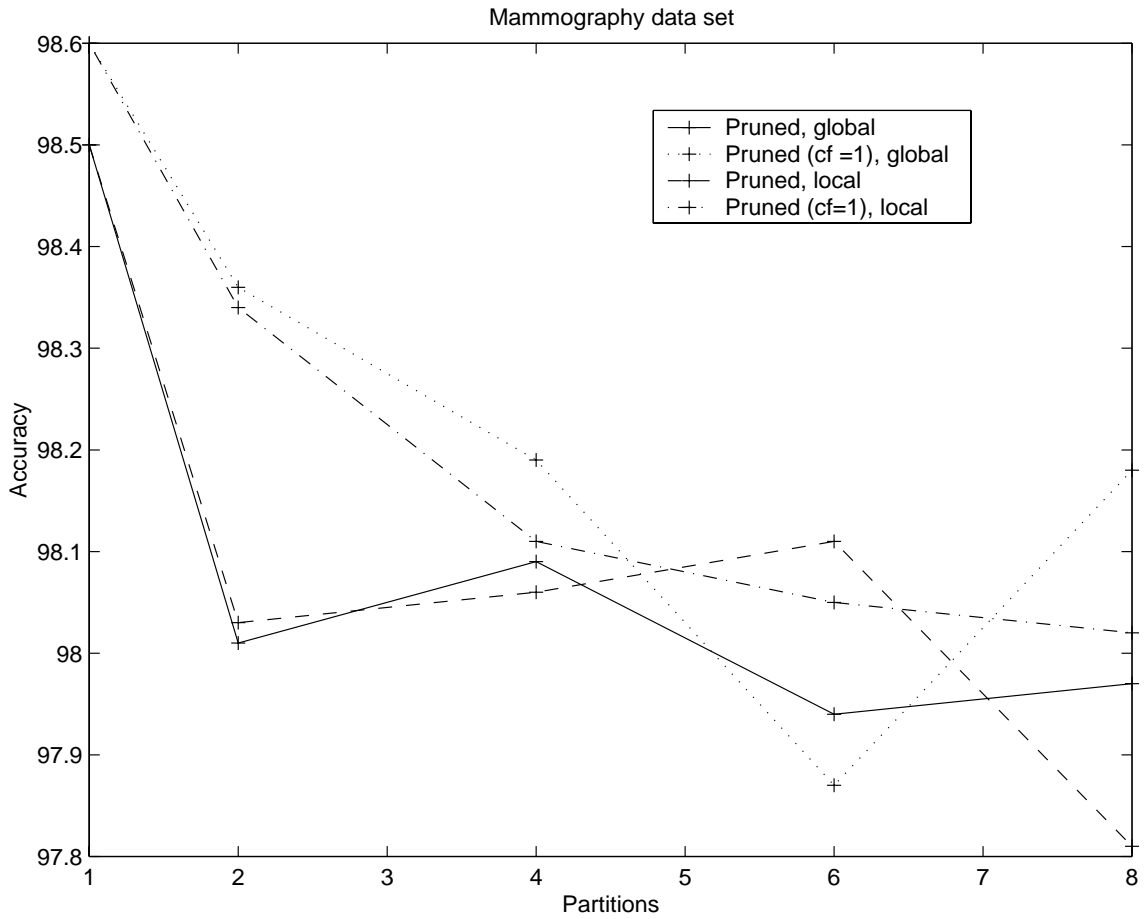


Figure 21. Accuracy results for Mammography data set for local and global performance estimates assigned by Ripper metric.

Table 10. Results on the Pima Diabetes data set comparing 10-fold cross-validation on the 12-partition with c4.5.

C4.5(Pruned cf = 1) Accuracy %	Certainty factor Accuracy %	Ripper metric Accuracy %
73.38	73.12	68.83

certainty factor metric and the Ripper metric. The experiments were only run on the 12-partition of Pima data set, to observe any differences.

The Ripper metric prefers the rules with less relative misclassifications. It does not take into account the actual coverage.

If a rule performs 0/5 (0 errors over 5 examples covered), the ripper metric gives it a performance measure of

$$(5-0)/(5+0) = 100.$$

A rule performing 1/100, a seemingly more reliable rule by the base coverage, has a performance of

$$(99-1)/(99+1) = 98.$$

The Ripper metric performed well with the mammography data set, as the high ρ also couldn't account for the better coverage rule, rather it further magnifies the difference.

$$(5-0.5)/(5+0.023*0) = 90$$

$$(99-0.5)/(99+42*1) = 70$$

With the Pima Diabetes data set, since the ρ of the majority class is small, 1.86, the rule with 1/100 would be given precedence.

$$(5 - 0.5)/(5 + 0.54*0) = 90, \text{ Minority class rule.}$$

$$(99 - 0.5)/(99 + 1.86*1) = 97.66, \text{ Majority class rule.}$$

Table 11. Results on the Pima Indian Diabetes data set comparing 10-fold cross-validation on the 12-partition with C4.5.

C4.5(Pruned cf = 1) Accuracy %	Certainty factor Accuracy %	LaPlace metric Accuracy %	Ripper metric Accuracy %
73.38	73.12	68.18	68.83

The LaPlace metric [PB91, SE94, QCJ95] was also tried with the same data sets. The LaPlace metric is

$$acc(R, E) = (TP + 1)/(TP + FP + k) \quad (3)$$

where k is the number of classes.

The LaPlace metric tends to favor the majority class. This can be attributed to the higher TP associated with the majority class rule due to more coverage.

For instance, consider the following rules derived from the mammography data set.

Rule 1:

Eattr > 100.926

Dattr > 5.312

→ class 2 [12/94, 99.0, 86.46]

Rule 7:

Eattr <= 55.160

→ class 1 [81/9418, 73.4, 99.13]

where $[x/y, \mathbf{a}, \mathbf{b}]$ represent x is incorrect examples (FP), y is total coverage (TP+FP), \mathbf{a} is the certainty factor, \mathbf{b} is the LaPlace estimate

Figure 22 illustrates the results on the Mammography data set. The experiments were also run for a 12-partition of the Pima Indian Diabetes data set. Table 11 tabulates the results of C4.5 on the entire data set, the Certainty factor metric, the Ripper metric and the LaPlace metric for the 12-partition.

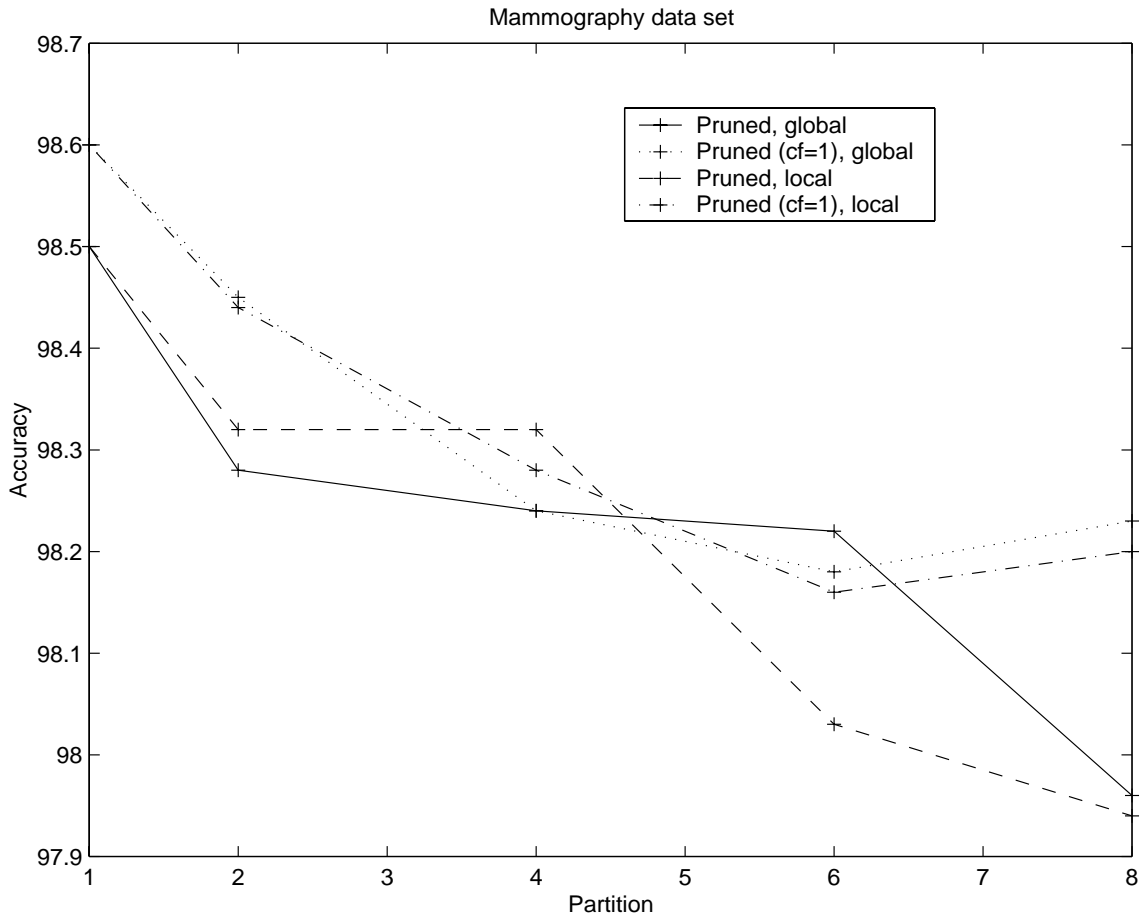


Figure 22. Accuracy results for the Mammography data set using local and global performance estimates assigned by LaPlace estimate.

A situation similar to the mammography rules was observed with some of the Pima rules, not as widespread though, because of the less skewed nature of the data set.

With mammography the performance was not affected, as even if all the class 2 (minority class) rules were killed because of conflict resolution, the performance was still close to 98% for that fold. This is because of the small percentage of examples in class 2. In the test set of 1119 there are only 25 or 26 examples of class 2. In some folds if any class 2 rules did survive, they helped improve the performance over just guessing the majority class. Now, with the Pima Indian Diabetes data set, the minority class is a significant portion of the test set, 27 out of 77, so elimination of class 1 (minority class) rules didn't help performance. The performance decreased because of less rule representation of the minority class in the final rule set.

The graph in Figure 23 is an accuracy comparison of the three metrics for the mammography data set. The experiments were run with decision trees pruned with a certainty factor of 1. Based on the discussion above, the graph shows the most improved results using the LaPlace metric. However, they still are not as good as C4.5 sequentially.

5.4 Weighted voting

The conflict resolution approach discussed in the previous section results in a single, conflict-free model as a classifier. A dynamic method of conflict resolution would have rules vote on an example, with the majority winning. In the voting approach, the data sets are distributed across processors, resulting in disjoint partitions. Rules are learned on each of the partitions and then used to classify unseen examples. Each rule could be assigned a weight. The approach would resolve conflicts at the time of classification, classifying the example as per majority vote.

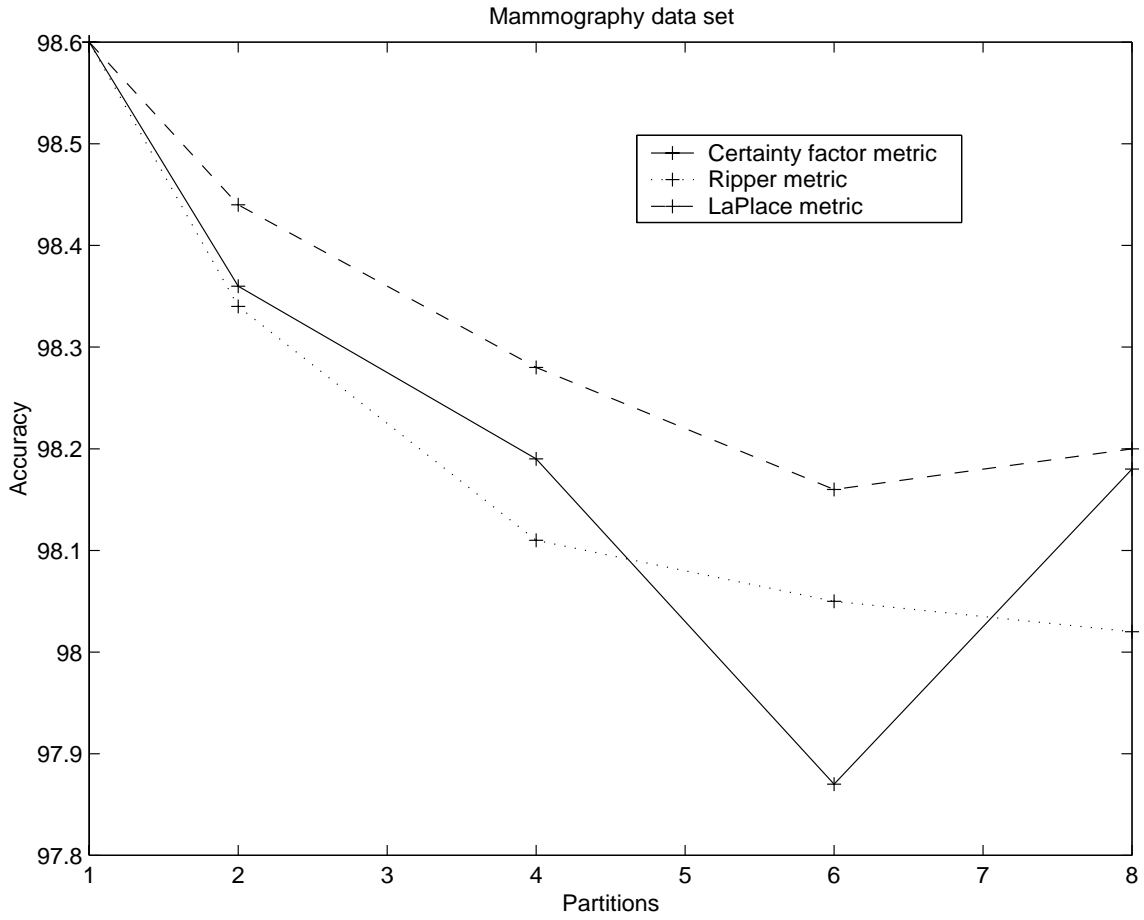


Figure 23. Accuracy comparison between Certainty factor metric, Ripper metric and LaPlace metric.

A simple way of applying weights is giving each rule a weight of 1, that is uniform weighted voting approach. Then, in the case of conflicts an example is given a class that majority rules claim. A similar approach has been tried by Woods et al [WKB97] and by Merz in [Mer99]. The difference is that the data sets are not disjoint and the voting is from an ensemble of classifiers that could be different. Another approach to voting would be assigning weights to the rules based on a performance metric.

Experiments were run on the Pima Indians Diabetes data set to compare with the conflict resolution discussed in the previous section. Decision trees were learned on disjoint subsets of data and converted to rules. The experiments were run for the two voting approaches, uniform voting and weighted voting. The weights were the certainty factor metric, evaluated using the TP and FP for each rule. The TP and FP for each rule were calculated by evaluating the rule on partitions other than what was learned on, to eliminate any bias whatsoever. The majority weight count was used to decide the classification of an unseen example. Figure 24 shows a comparison graph of the conflict resolution approach and voting.

In a similar approach being developed, the rules are learned using Ripper on disjoint partitions. The TP and FP from each rule, after it is applied to all examples on which it was not trained, are combined into a metric to give a performance weight. When conflicting rules fire for an example, the weights of the fired rules are summed up and the example assigned to the class with the highest performance weight. The experiments reported using Ripper are from Jimmy Chao's thesis.

The graph in Figure 25 shows an accuracy comparison of the conflict resolution approach and voting strategies using Ripper for a 2,4,8,10,12 partition on the Pima Indian Diabetes data set.

Using weighted voting, the experiments were also run for 16 and 32 partitions and the accuracy did not drop. Table 12 tabulates the accuracy results with Ripper as the learner.

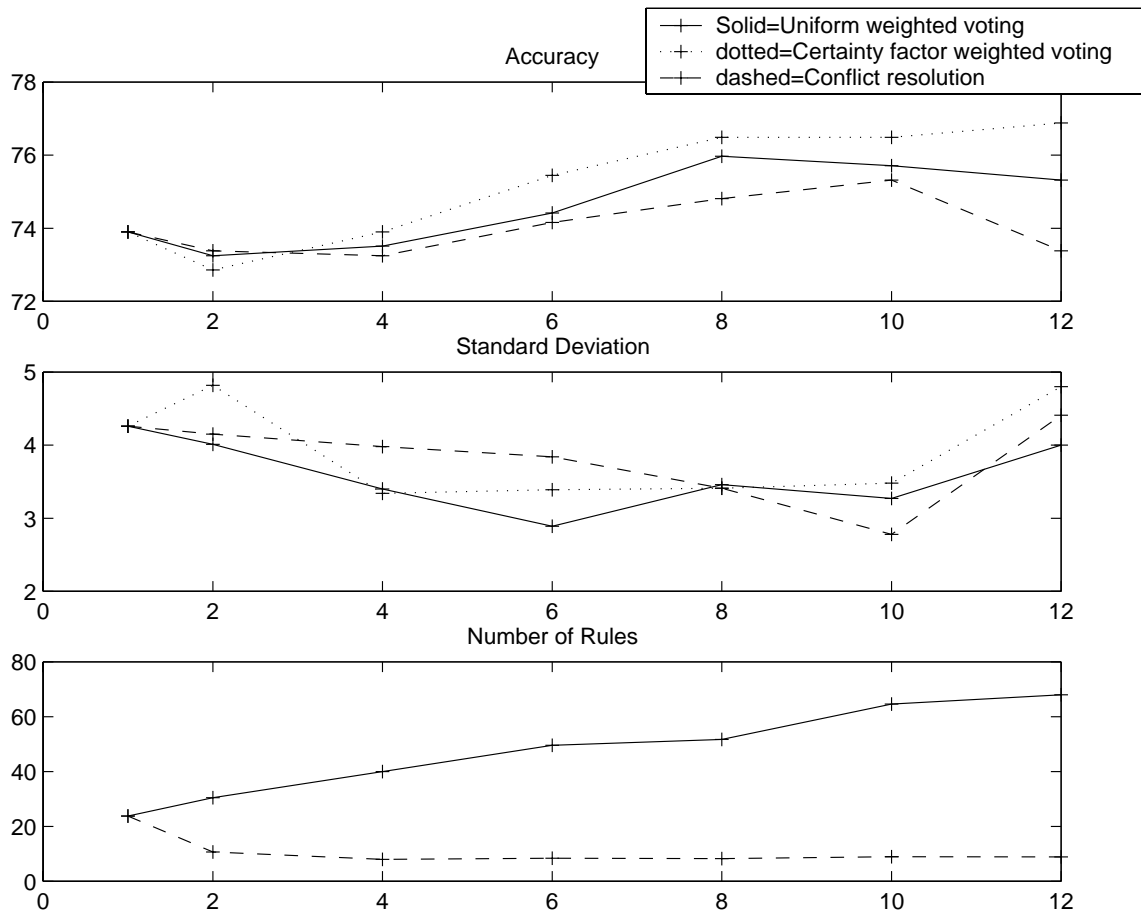


Figure 24. Comparison of Voting strategies against conflict resolution for the Pima Indians Diabetes data set.

Table 12. Results from voting with Ripper on the Pima Indian Diabetes data set.

Partitions	2	4	8	10	12	16	32
Accuracy	73.05	75.658	76.314	76.181	74.488	75.273	76.314
Std. Dev.	4.74585	4.24303	5.14239	4.31052	4.06352	4.16724	4.86807

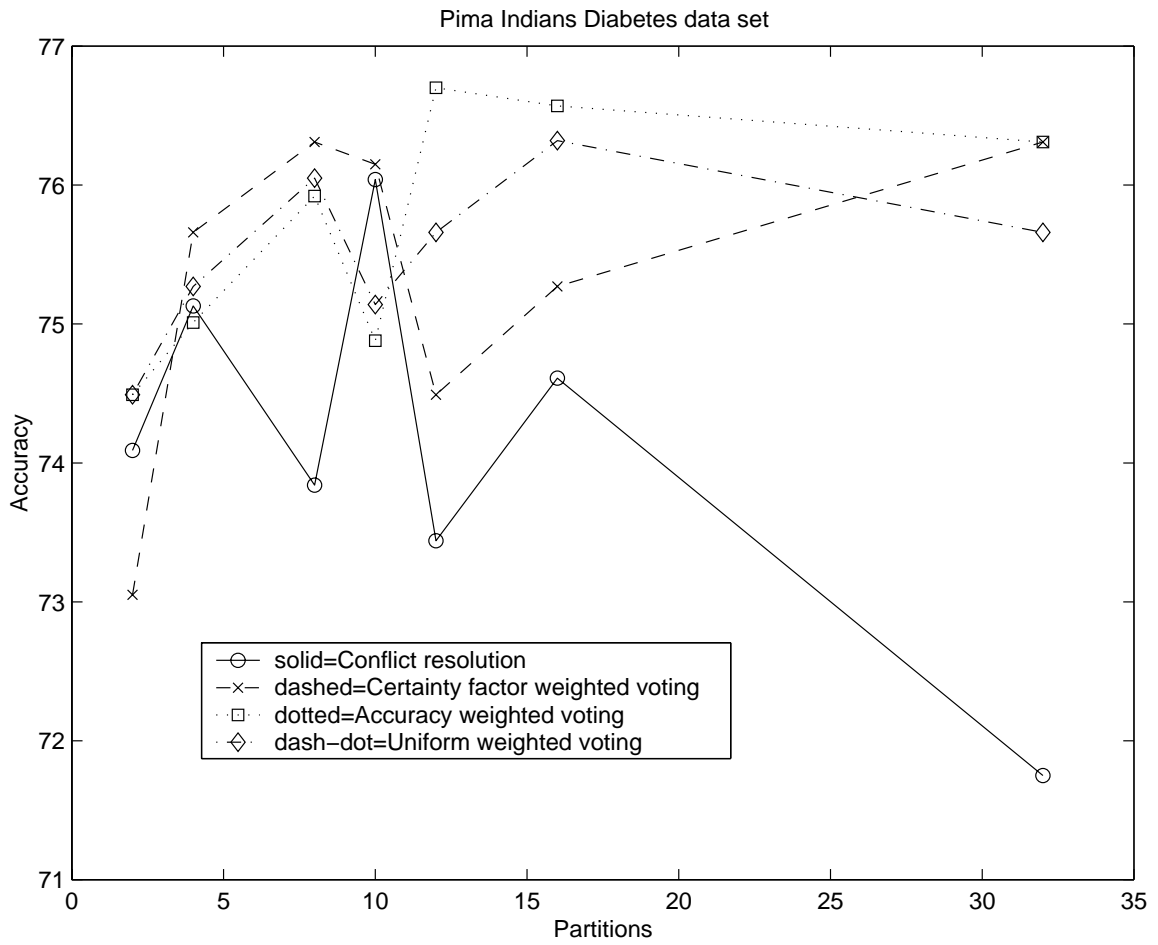


Figure 25. Accuracy comparison between conflict resolution and voting using Ripper.

The graph shows that voting using Ripper to learn models outperforms conflict resolution using C4.5 to learn models. With Ripper, the nature of the rules is more general and thereby less rules are generated. The way Ripper learns is by aiming to maximize the coverage of one rule, which would arise through more generality. Thus rules have a high TP+FP. With Ripper there were not any rules produced with extremely low coverage. With C4.5, there were leaves with a low coverage such as 0/4. These leaves cause problems, as they do not carry enough information since they are too specific, and at the same time are not assigned a bad certainty factor metric. That is a difference between decision tree learning and Ripper. Decision trees keep partitioning till a stage is reached where the best partition is made, and this stage is overfitting leading to some very specific decision rules.

In an experiment run, C4.5 was also forced to have a higher coverage at the leaves by setting the weight 'm' to 5. That is, at least two branches must have a minimum number of 5 objects. There was an improvement in the accuracy observed as shown in Figure 26. There was a drop after the 8th partition, because with the increase in the partitions the size of data set reduces, the rules produced are of a very general nature, in some cases just a couple of rules are learned for a partition.

The experiment was run on the Pima Indian Diabetes data set for a 2,4,6,8,10,12 partition. The trees were grown with default pruning and setting $m = 5$.

5.5 Learn again?

The rules not performing well on one partition may be associated with a second learning phase. The rules falling within a $\pm x\%$ range (x is definable) of a threshold t may be associated with a second learning phase. As these rules are passed across the processors, identify the coverage of the rule over the data set, and grow decision trees on these data sets. The processor creates a copy of the rule and appends the best performing branch(es) (leading to a leaf) of the tree to the rule, thus creating

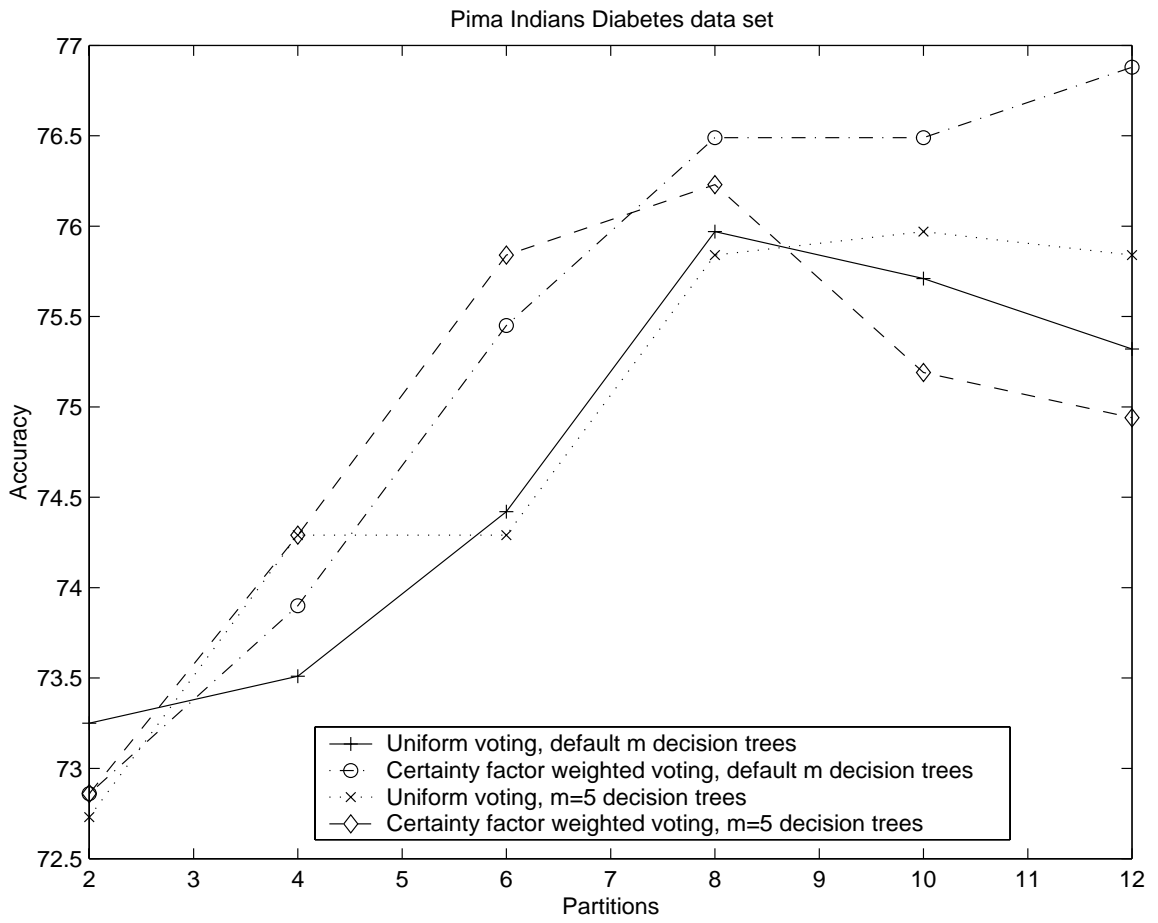


Figure 26. Accuracy results for decision trees grown with $m=5$ and default pruning.

a specialized version of the rule. The processor then sends out the original rule and the specialized version to the next processor. The next processor also evaluates the original rule and the specialized version of it, and potentially creates another specialization of the original rule and further specializes the specialized rule. This procedure is repeated until the performance worsens with specializations. The performance for a rule is measured by applying the rule's TP and FP to one of the metrics discussed in section 5.3. With this approach, there will not be a decrease in the performance as long as the original rule remains in the final rule set, but has a lower priority than its specialized version. Any examples left uncovered by the specialized rule will still be classified by the more general rule.

As a simple example of the potential for improvement consider the example of Figure 9. Assume the second set of rules was obtained using attribute 2 (Att2) from processor 1 (Proc1) and then applied 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 9. 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%.

Figure 27 summarizes the learning discussion presented in this chapter.

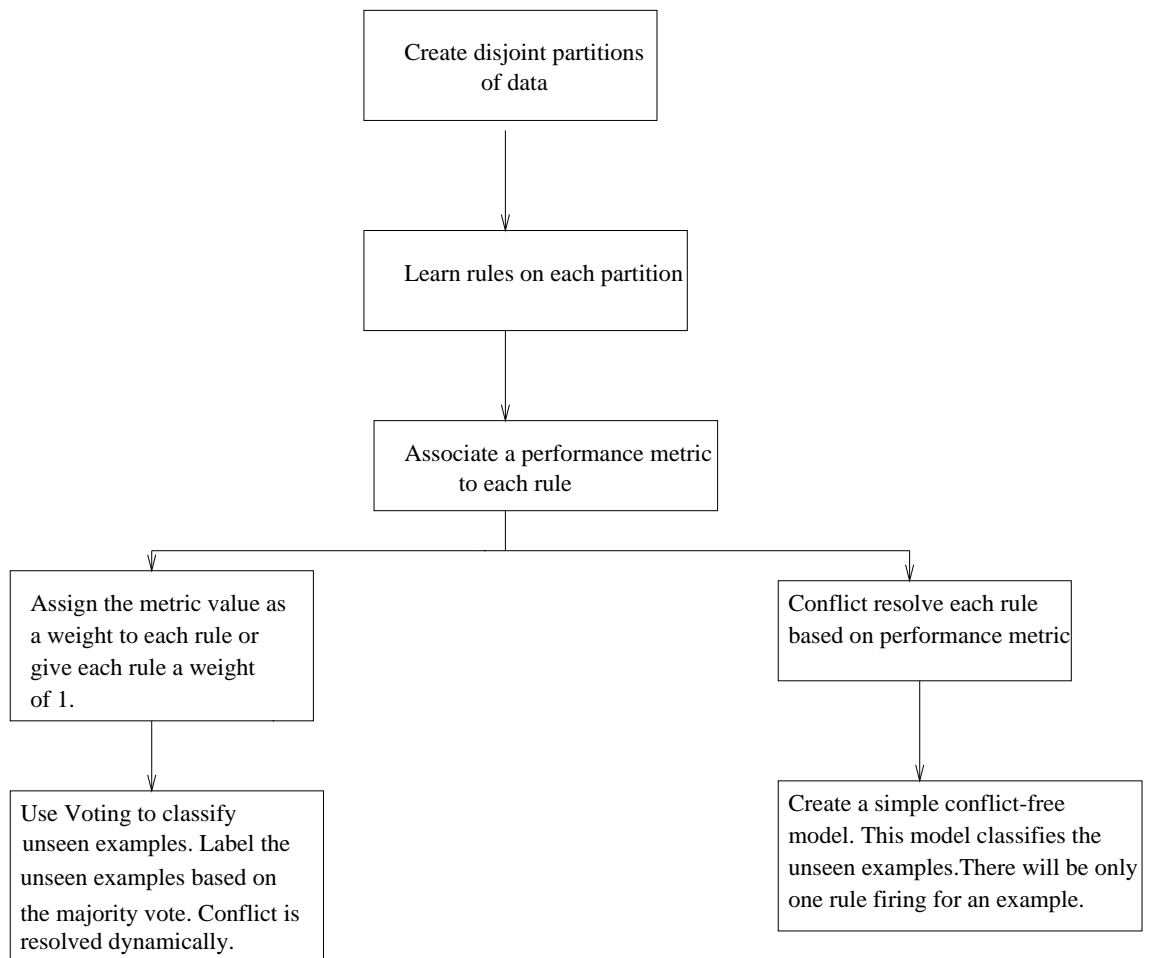


Figure 27. Summary.

CHAPTER 6

AVATAR DISCUSSION: A REAL-WORLD EXAMPLE

6.1 Introduction

Large scale data sets are becoming ubiquitous. Intelligent visualization is needed to unearth the information embedded in the terabytes of data. To aid intelligent visualization, efficient and scalable parallel and/or distributed learning algorithms are needed. The visualization tool should be able to guide the user intelligently through the data set without him or her missing the relevant portions of the data set.

Large data sets can be distributed and learning applied in parallel to develop a model of user interests. User profiles, called AVATARS are created as result. AVATAR is an acronym for Adaptive Visualization Aid for Touring and Recovery. An AVATAR can be unique to each user and attached to a type or category of data set.

The procedure behind the visualization and creation of AVATAR can be summarized as shown in Figure 28. If a personalized version is desired the visualizer will need to help label training data [CH99].

The MUSTAFA visualization tool [Glaa, Glab] has been modified at University of South Florida to facilitate the process of creating AVATARS. During a training session, the user browses a data set using MUSTAFA and identifies salient or interesting regions. A labeled data set is generated at the end of the training session. This data set can be used by a data mining system to create AVATARS. This AVATAR is the user profile and can be used to classify new data sets in domains similar to its training set. The AVATAR will then be able to guide the user through the interesting regions of the new data sets.

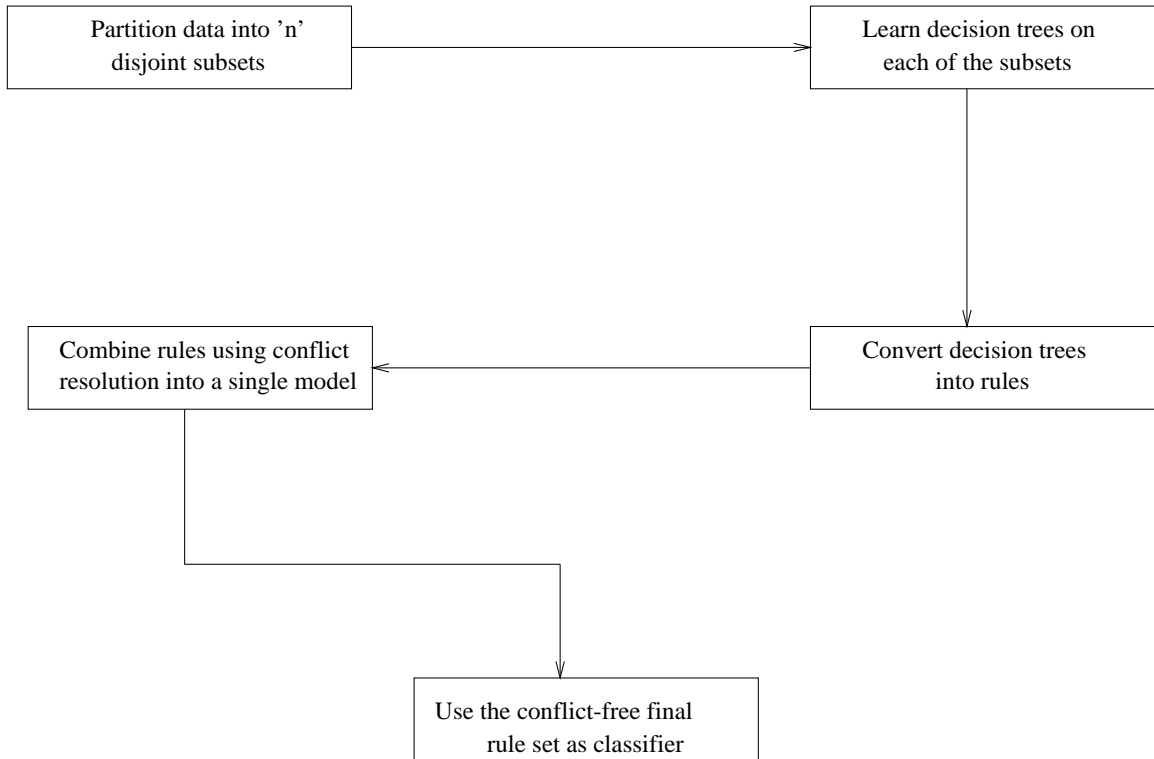


Figure 28. Summary.

Disjoint subsets of labeled data are generated during a training session. Decision trees are learned on each of the training sets in parallel and converted to rules. The rules reflect the saliencies selected by the user. The rules are combined together into a single model. This rule model is the learned representation of regions likely to interest the user. Figure 29 displays the AVATAR front panel.

6.2 Sample data set

Experiments were run on the “Can” Exodus [YS94] data set provided by W. Philip Kegelmeyer. This data set has 10088 nodes, 9 data components at each node and 44 time steps. Figure 30 depicts the Can Exodus data set at the various stages of visualization. The figure depicts the first, middle and last frame of the data set.

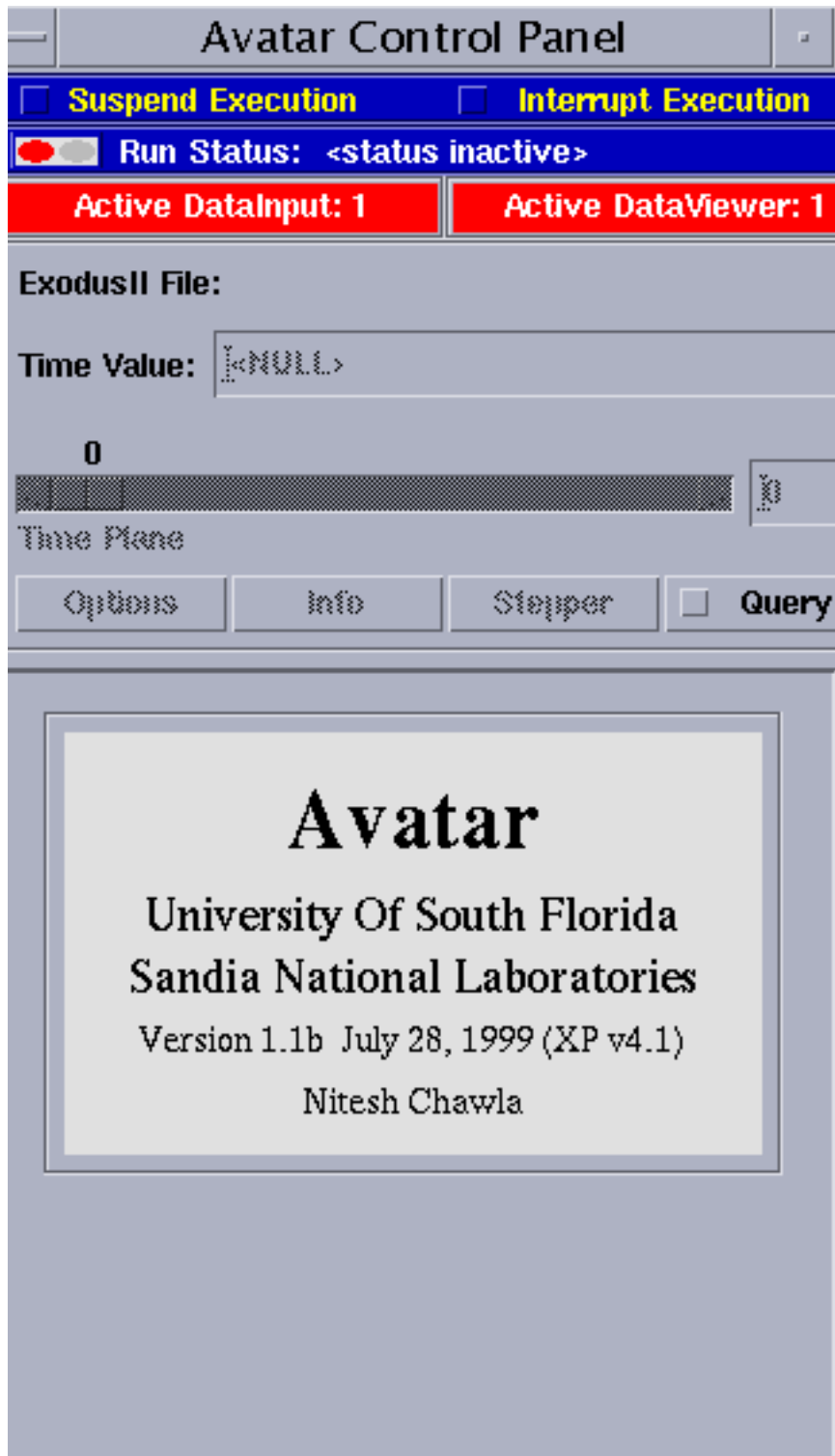


Figure 29. AVATAR front panel.

6.3 How to train?

The MUSTAFA visualization tool has been modified to be used for intelligent visualization. The user has an option of setting the desired saliency at the time of browsing.

The saliencies offered to the user currently are:

1. Unknown → Saliency 1
2. Unseen low → Saliency 2
3. Not interesting → Saliency 3
4. Probably interesting → Saliency 4
5. Interesting → Saliency 5
6. Very interesting → Saliency 6

There is a separate panel available to the user with all the AVATAR features for training. Figure 31 shows the panel with all the AVATAR features currently available for the training phase. The user can change the saliency as required. When a user finds something interesting or uninteresting, he/she can define a rectangular region by marking four nodes to form a rectangle and the nodes within that rectangular region will be labeled by the currently selected saliency value. After the user is done browsing, the salience information that was captured can be saved in a file specified by the user. At the same time a version of the Exodus data set, which the user trained on, will be created with saliency added as a nodal variable in the mesh. For training, a flat file is generated which is a listing of feature vectors followed by a label. This format is compatible with the C4.5 input format.

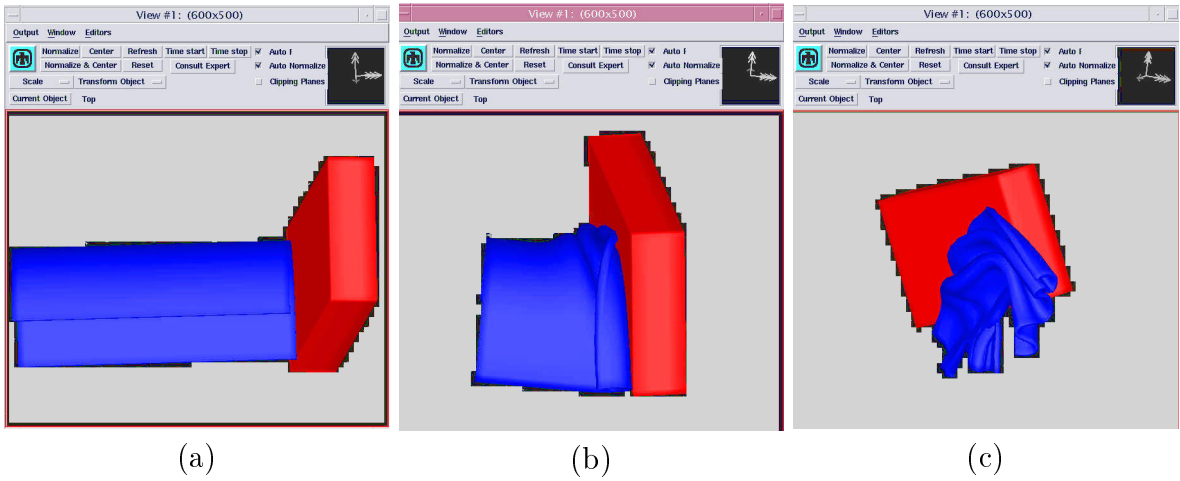


Figure 30. a) First frame. and b) Second frame. c) Third frame.

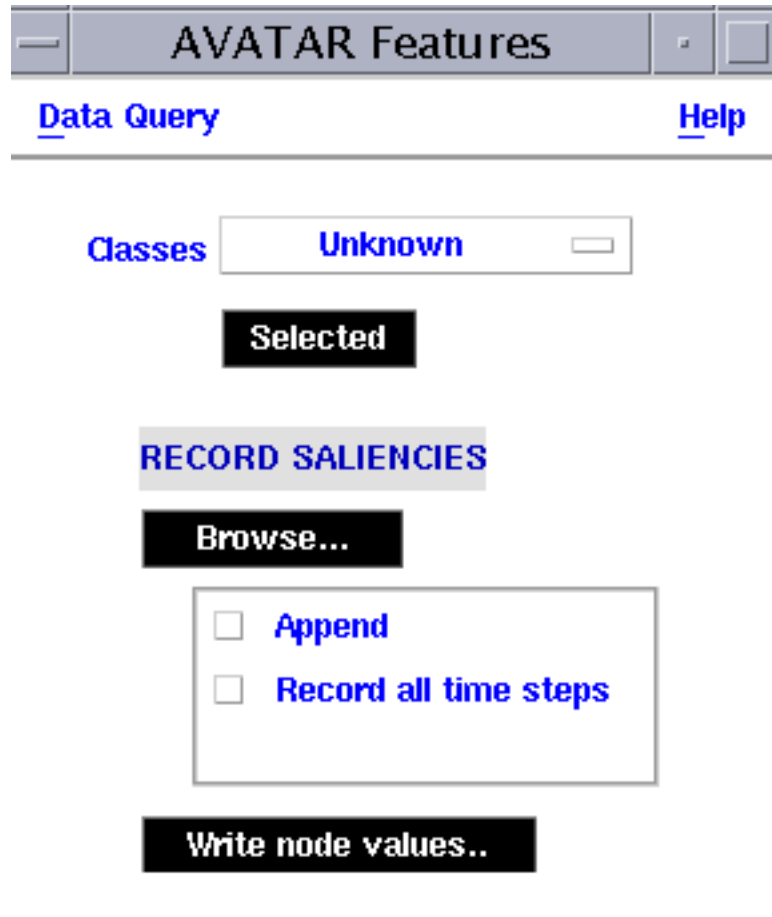


Figure 31. A view of AVATAR features panel.

The format of the flat file used for training is as follows -

$v_1, v_2, \dots, v_m, \text{class}$

|

|

|

$v_1, v_2, \dots, v_m, \text{class}$

where v_1, v_2, \dots, v_m are the nodal variables and class is the saliency attached to the node.

6.3.1 Training experiment

In our experimental setup, for training, the user browsed through odd time steps and identified regions as interesting or very interesting. All unlabeled regions were given a saliency of unseen low. The size of the labeled data generated from the can data set shown in Figure 30 was 221,936 examples each with 9 numeric features. There were approximately 1300 rules created. The rules were created from a pruned (default pruning) decision tree which resulted from applying C4.5 [Qui92] to a training set. The training set consisted of the 22 odd-numbered time steps of the sequence, and the test set consisted of the even numbered time steps. The saliencies that were marked were interesting (saliency 5) and very interesting (saliency 6). The rest of the examples were given a saliency of unseen low (saliency 2) by default. Figures 33 and 34 show snapshots of the training frames. The plus in the training images indicates a node within the region of interest. Figure 33 shows the nodes classified as interesting by the user for time steps 3 and 11 respectively, figure 34 shows the nodes classified as very interesting for the time steps 23 and 39 respectively. The nodes near the portion of can being crushed were given a saliency of very interesting and the nodes slightly away were given a saliency of interesting.

In Figure 32 the flowchart of training input and training output is depicted.

Training output

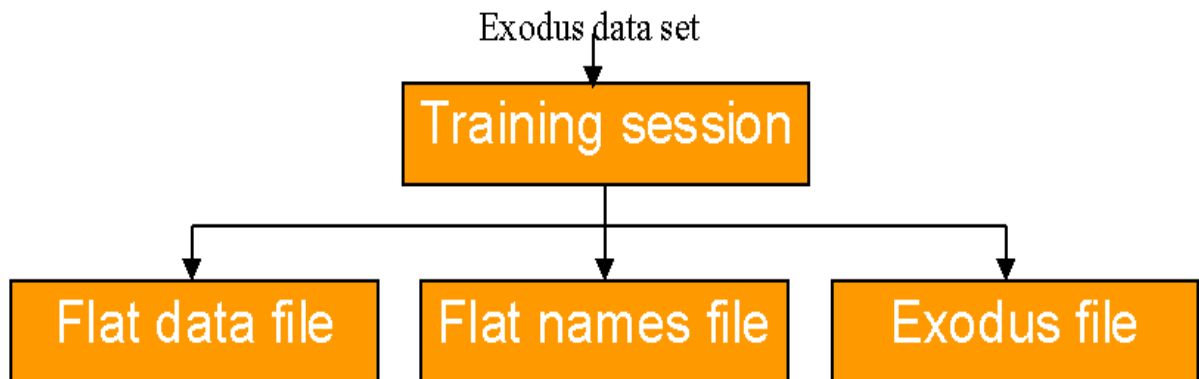


Figure 32. The above figure shows that the input to the training session is the Exodus data set. At the end of the train session the output produced is as follows: **Flat data file**: A file comprising of all the nodes data and the corresponding saliencies. **Flat names file**: A file comprising of all the attributes listing and classes(saliency). **Exodus file**: The training Exodus file recreated with saliency added as another nodal variable.

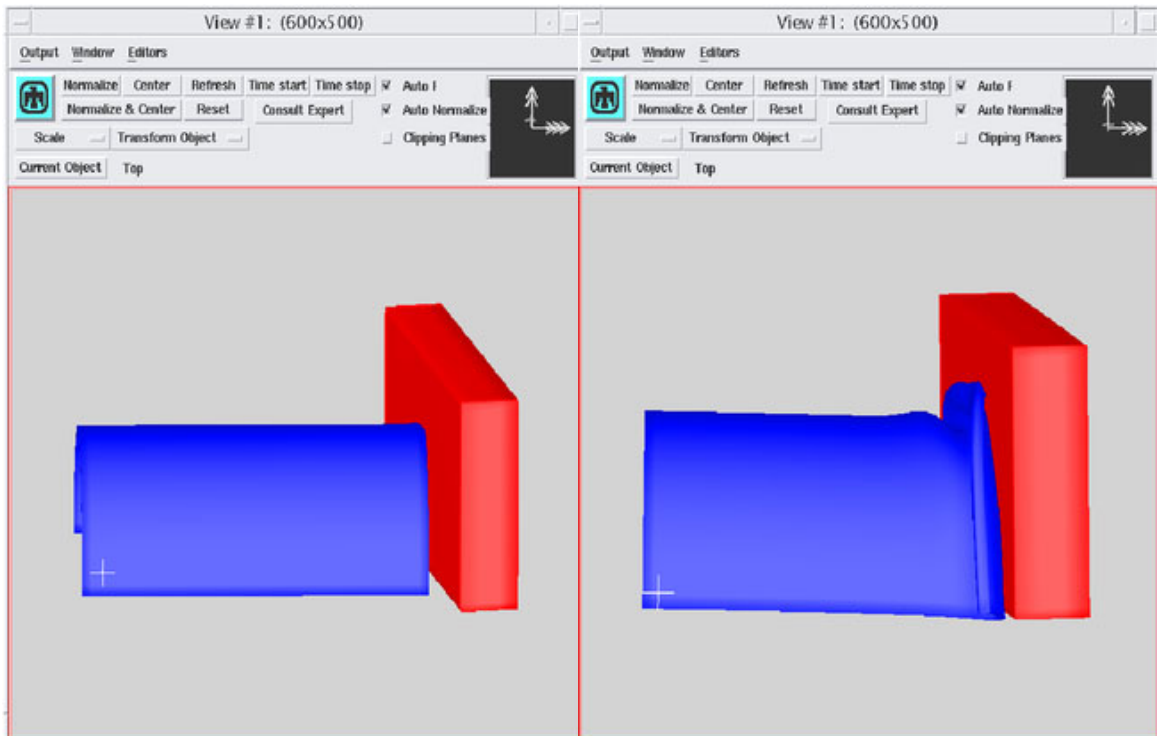


Figure 33. A training frame with marked regions.

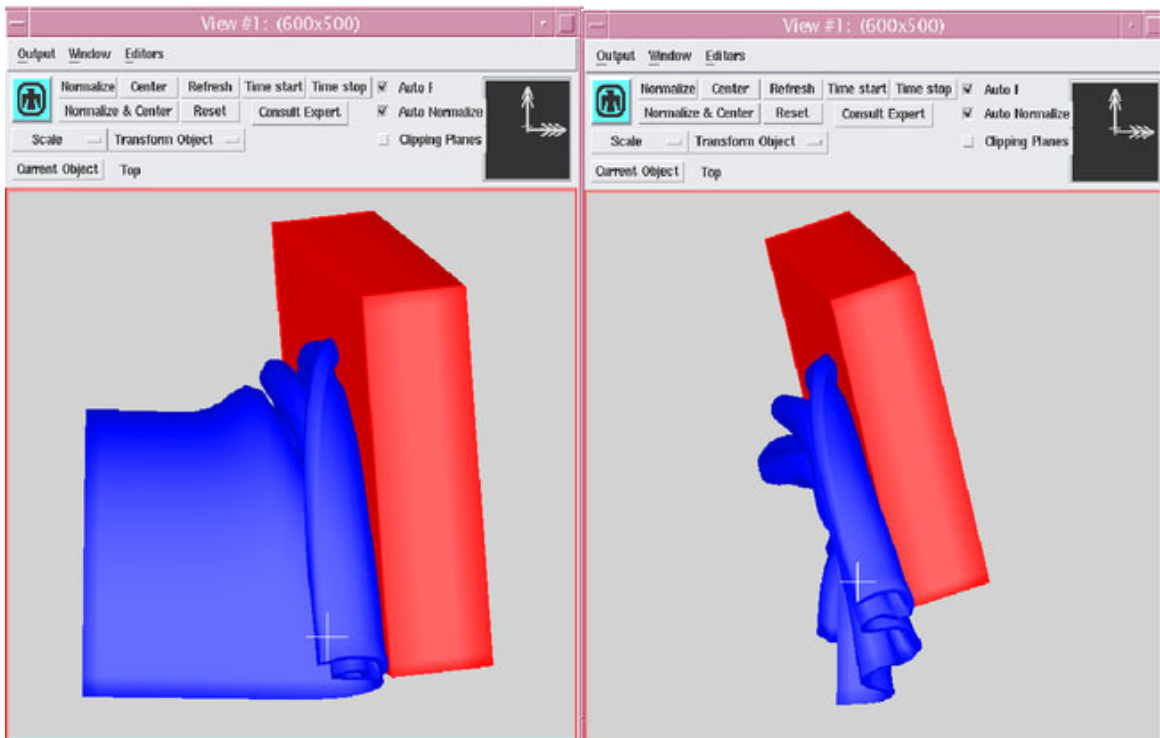


Figure 34. A second training frame with marked regions.

Testing output

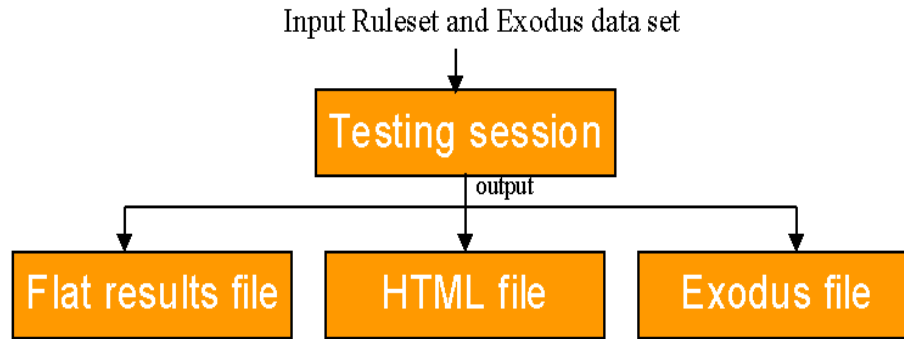


Figure 35. The above figure shows that the input to the training session is the Exodus data set. At the end of the train session the output produced is as follows: **Flat results file**: A file containing grouping of nodes by time-step and their corresponding saliency. **HTML file**: An HTML file providing a web interface for web query. **Exodus file**: A similar Exodus file is created with saliency added as a feature to each node.

6.4 Testing

The set of rules generated from training was used to classify the unseen even numbered time steps of the can exodus data set. After all the nodes have been tested against the rules, an HTML file containing a grouping of all the nodes by the time-step and the corresponding saliency is created, which can be used by the user to view the saliencies. The user can choose a node from the web interface and using the Data Query option in MUSTAFA can visualize the node of the data set. Also, an exodus file similar to the one tested is created, with the saliency added as a nodal variable. A flow chart of the testing session is depicted in figure 35.

The following figures display the result of testing on the even numbered time steps of the can Exodus data set.

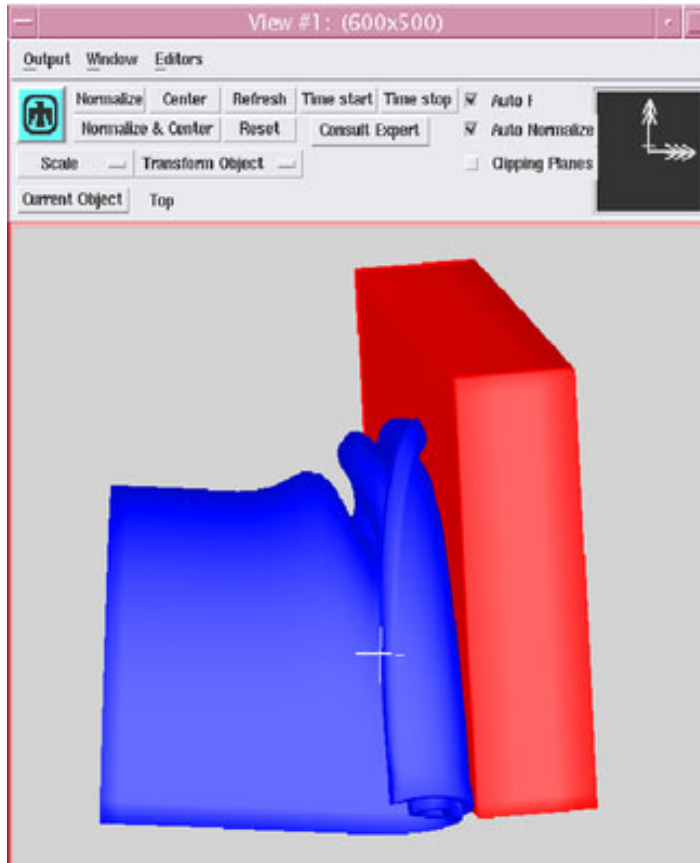


Figure 36. Node correctly classified as very interesting.

Figure 36 shows the correct classification of a node/region as very interesting (saliency 6) which is correct classification, as the nodes near the region of the can being crushed were marked as very interesting.

Figure 37 shows the incorrect classification of a node as very interesting, since the nodes in the region from the section of the can being crushed received a saliency of interesting during training.

Figure 38 shows a node correctly classified as interesting, since the nodes at this vicinity were given a saliency of interesting during testing.

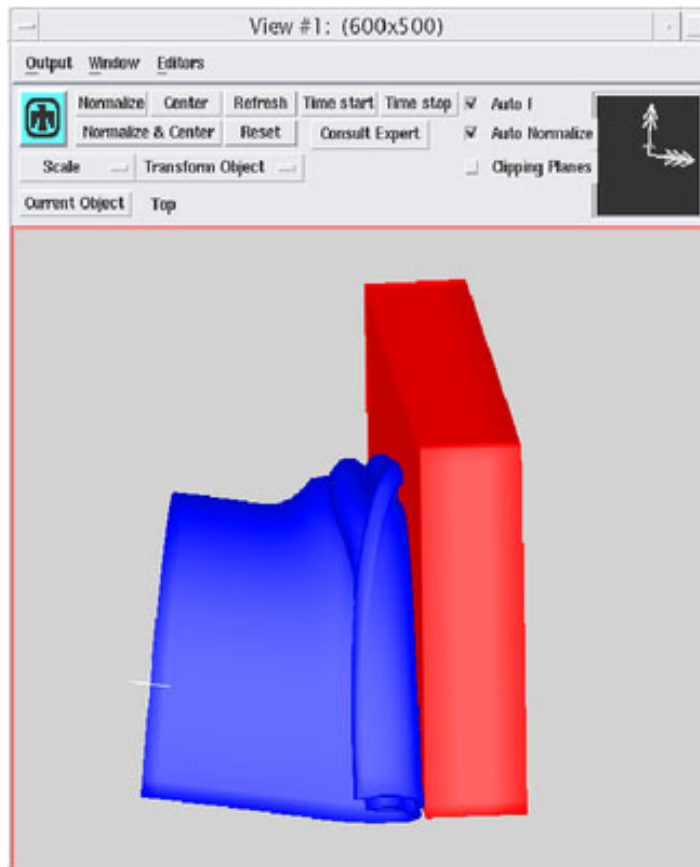


Figure 37. Node incorrectly classified as very interesting.

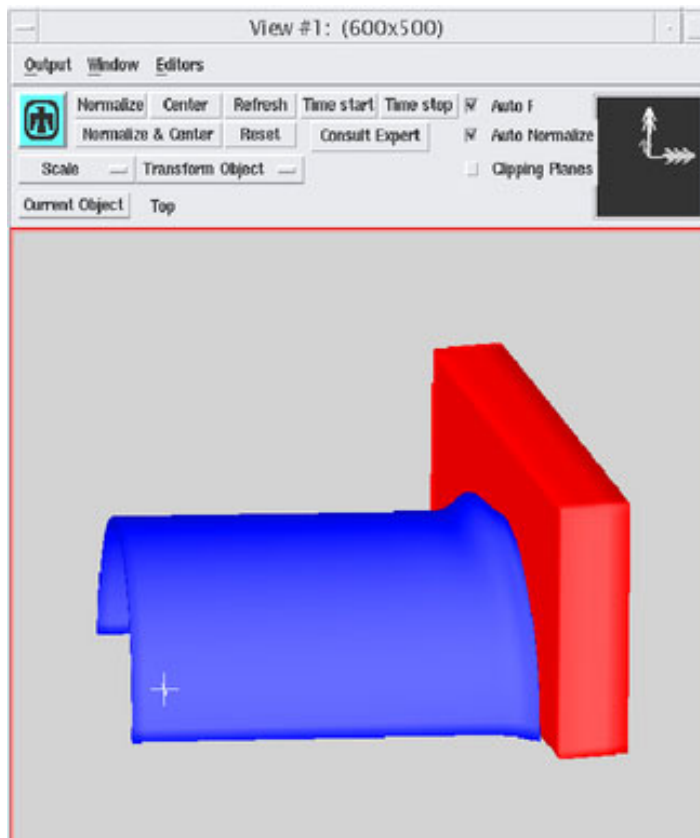


Figure 38. Node correctly classified as interesting.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Summary

In this thesis, learning rules in a distributed environment was discussed. The data sets for learning were disjoint. New approaches to rule combination were presented. The first approach of conflict resolution had a prohibitive cost of data communication, though it was promising in terms of accuracy. The second approach, in which rules are assigned a performance based on a metric and conflicts resolved by discarding the worse performing rule, had promising results as well and did not have the drawback of data communication. Another comparison of the conflict resolution approach against the voting approach was presented. The voting approach performed better in accuracy for the data set they were tested on. The approaches had interesting results to further pursue the distributed learning on disjoint data sets.

7.2 The journey ahead

This thesis leads to some more avenues for research in learning rules in a distributed environment. The following could be the possible extensions to the thesis.

1. Tests on larger data sets.
2. The current algorithm only works for continuous attributes. The ability to deal with categorical attributes could be incorporated and the feasibility of the algorithm tested with this feature.
3. Specialization and voting using decision rules tested in depth.

4. Utilization of the Metacost [Dom99] variation in the metric. In our set up, a validation set could be distributed across every other processor too. This validation set should be the same on every processor but different from any of the training partitions. When the rules are learned on the training partition, the rules could be evaluated on the validation set. Since this validation set is the same across every processor, the probability of each class for every example could be calculated by having each classifier vote on them. These probability distributions could then be associated with the corresponding rules by replacing the error-based estimates.

REFERENCES

- [BA99] Roberto Bayardo and Rakesh Agarwal. Mining the most interesting rules. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154, 1999.
- [BCG⁺99] S. Bailey, E. Creel, R. Grossman, S. Gutti, and H. Sivakumar. A high performance implementation of the data space transfer protocol (dstp). In *Workshop on Large-Scale Parallel KDD Systems*, pages 6–12, 1999.
- [BFOS84] L. Breiman, J.H. Friedman, R.A. Olshen, and P.J. Stone. *Classification and Regression Trees*. Wadsworth International Group, Belmont, CA, 1984.
- [BK99] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting and variants. *Machine Learning*, 36(1,2), 1999.
- [BM98] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.
- [Bre96] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.
- [CH99] N. Chawla and L.O. Hall. Modifying MUSTAFA to capture salient data. Technical Report ISL-99-01, University of South Florida, Computer Science and Eng. Dept., 1999.
- [Coh95] W. Cohen. Fast effective rule induction. In *Proceedings of the 12th Conference of Machine Learning*, 1995.
- [CS93a] Philip Chan and Salvatore Stolfo. Experiments on multistrategy learning by meta-learning. In *Proc. Second Intl. Conf. Info. Know. Manag.*, pages 314–323, 1993.
- [CS93b] Philip Chan and Salvatore Stolfo. Towards parallel and distributed learning by meta-learning. In *Working Notes AAAI Work. Know. Disc. Databases*, pages 227–240, 1993.
- [CS95] Philip Chan and Salvatore Stolfo. Learning arbiter and combiner trees from partitioned data for scaling machine learning. In *Proc. Intl. Conf. on Knowledge Discovery and Data Mining*, pages 39–44, 1995.

- [CS96] Philip Chan and Salvatore Stolfo. Scaling learning by meta-learning over disjoint and partially replicated data. In *9th Florida Artificial Intelligence Research Symposium*, pages 151–155, 1996.
- [DGST97] J. Darlington, Y. Guo, J. Sutiwaraphun, and H. To. Parallel induction algorithms for data mining. In *Advances in Intelligent Data Analysis Reasoning about Data, Second International Symposium, IDA-97. Proceedings*, pages 437–445, 1997.
- [DNS91] D. J. DeWitt, J. F. Naughton, and D. A. Schneider. Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Proc. of the first Int'l Conf on Parallel and Distributed Information Systems*, pages 280–291, 1991.
- [Dom99] Pedro Domingos. Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 155–164, 1999.
- [FB85] Li-Min Fu and Bruce G Buchanan. Learning intermediate concepts in constructing a hierarchical knowledge base. In *International Joint Conference on Artificial Intelligence*, 1985.
- [FJP96] Daniel N. Hennessy Foster John Provost. Scaling up: Distributed machine learning with cooperation. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence, AAAI '96*, pages 74–79, 1996.
- [FPO99] David Jensen Foster Provost and Tim Oates. Efficient progressive sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 23–32, 1999.
- [Fre95] Y. Freund. Boosting a weak learning algorithm by majority. *Information and Computation*, 121(2):256–285, 1995.
- [FS95] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, 1995.
- [FW94] J. Furnkranz and G. Widmer. The effects of training set size on decision tree complexity. In *Machine Learning: Proceedings of the Eleventh Annual Conference*, 1994.
- [Glaa] Michael W. Glass. Mustafa tutorial. Albuquerque, NM 87185.
- [Glab] Michael W. Glass. Mustafa user's guide. Albuquerque, NM 87185.
- [Gre84] Michael J. Greenacre. *Theory and Application of Correspondence Analysis*. Academic Press, London, 1984.

- [HCB98] L. Hall, N. Chawla, and K. Bowyer. Decision tree learning on very large data sets. In *IEEE conference on Systems, Man and Cybernetics*, pages 2579–2584, 1998.
- [HCBK99] L.O. Hall, N. Chawla, K.W. Bowyer, and W.P. Kegelmeyer. Learning rules from distributed data. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 1999.
- [HMS66] E.B. Hunt, J. Martin, and P.J. Stone. *Experiments in Induction*. Academic Press, New York, NY, 1966.
- [JKK98] M. Joshi, G. Karypis, and V. Kumar. Scalparc: A new scalable and efficient parallel classification algorithm for mining large datasets. In *12th. International Parallel Processing Symposium*, pages 573–, April 1998.
- [Kuf97] R. Kufirin. Generating c4.5 production rules in parallel. In *The Fourteenth National Conference on Artificial Intelligence*, pages 565–570, 1997.
- [Mer99] C. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1,2), 1999.
- [MHB] Thomas Moore, Larry Hall, and Kewin Bowyer. Experience with a parallel decision tree builder. <http://www.csee.usf.edu/tmoore4>.
- [Min89a] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2), pages 227–243, 1989.
- [Min89b] J. Mingers. An empirical comparison of selection methods for decision tree induction. *Machine Learning*, 3(4), pages 319–342, 1989.
- [Mit97] T.M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, 1997.
- [OJ97] T. Oates and D. Jensen. The effects of training set size on decision tree complexity. In *Proceedings of the 14th International Conference on Machine Learning*, pages 254–262, 1997.
- [OJ98] T. Oates and D. Jensen. Large datasets lead to overly complex models: an explanation and a solution. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining. August 1998*, 1998.
- [PB91] P.Clark and R. Boswell. Rule induction with cn2: Some recent improvements. In *Proceedings of the Fifth European Working Session on Learning*, pages 151–163, 1991.
- [QCJ95] J.R. Quinlan and R.M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the IJCAI-95*, pages 1019–1024, 1995.
- [Qui87] J.R. Quinlan. Simplifying decision trees. *International Journal of Man Machine Studies*, V.27, pages 227–248, 1987.

- [Qui92] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Mateo, CA, 1992.
- [Qui96] J.R. Quinlan. Improved use of continuous attributes in c4.5. *Journal of Artificial Intelligence Research*, pages 77–90, 1996.
- [SAM96] J. Shafer, R. Agrawal, and M. Mehta. Sprint: A scalable parallel classifier for data mining. In *Proceedings of the 22nd VLDB Conference, Mumbai (Bombay), India*, pages 1–12, 1996.
- [SB99] W.D. Shannon and D. Banks. Combining classification trees using mle. *Statistics in Medicine*, 1999.
- [SE94] R. Segal and O. Etzioni. Learning decision lists using homogeneous rules. In *Proceedings of the AAAI-94*, pages 619–624, 1994.
- [WGT90] S.M. Weiss, R.S. Galen, and P.V. Tadepalli. Maximizing the predictive value of production rules. In *Artificial Intelligence*, pages 47–71, 1990.
- [Wil90] G.J. Williams. *Inducing and Combining Multiple Decision Trees*. PhD thesis, Australian National University, Canberra, Australia, 1990.
- [Win92] P.H. Winston. *Artificial Intelligence (3rd ed.)*. Reading. Addison-Wesley, MA, 1992.
- [WKB97] K. Woods, W.P. Kegelmeyer, and K.W. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):405–410, April 1997.
- [Wol92] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.
- [YS94] Victor R. Yarberry and Larry A. Schoof. *EXODUS II: A Finite Element Data Model*. Sandia National Labs, Albuquerque, NM 87185 and Livermore, CA 94550, 1994.