SANDIA REPORT

SAND2000-8203

Unlimited Release

Printed January 2000

AVATAR - Adaptive Visualization Aid for Touring and Recovery

RECEIVED
FEB 14 2000
OSTI

L. O. Hall, K. W. Bowyer, N. Chawla, T. Moore, Jr., W. P. Kegelmeyer

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States
Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Prices available from (703) 605-6000
Web site: <http://www.ntis.gov/ordering.htm>

Available to the public from
National Technical Information Service
U.S. Department of Commerce
5285 Port Royal Rd
Springfield, VA 22161

NTIS price codes
Printed copy: A04
Microfiche copy: A01

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

SAND 2000-8203
Unlimited Release
Printed January, 2000

AVATAR - Adaptive Visualization Aid for Touring and Recovery

Lawrence O. Hall, Kevin W. Bowyer, Nitesh Chawla, Thomas Moore, Jr.
Dept. of Computer Science and Engineering, University of South Florida, Tampa, FL.
33620
{hall,kwb,chawla,tmoore4}@csee.usf.edu

W. Philip Kegelmeyer
Sandia National Laboratories, Advanced Concepts Department, P.O. Box 969, MS 9214,
Livermore, CA, 94551-0969,
wpk@sandia.gov
October 31, 1999

Abstract

This document provides a report on the initial development of software which uses a standard visualization tool to determine, label and display salient regions in large 3D physics simulation datasets. This software uses parallel pattern recognition behind the scenes to handle the huge volume of data. This software is called AVATAR (Adaptive Visualization Aid for Touring and Recovery). It integrates approaches to gathering labeled training data, learning from large training sets utilizing parallelism and the final display of salient data in unseen visualization data sets.

CONTENTS

Contents

1	Introduction	8
2	The Visualization Interface	9
2.1	AVATAR and MUSTAFA	9
2.2	Sample Data set	13
2.3	How to train?	13
2.4	Testing	15
2.5	Current Status	19
2.5.1	Current Capabilities	19
2.5.2	Region Display Issues	24
2.6	The Journey Ahead	25
3	Review of Approaches to Distributed or Parallel Learning	26
3.1	Overview of Rule-Based Learning Algorithms	27
3.2	Distributed learning with RL	29
3.3	Distributed learning with SKICAT	29
3.4	Distributed learning utilizing Meta-Learning	30
3.5	Bagging	30
3.6	Unevenly distributed example sets	31
3.7	Correspondence Analysis	31
3.8	Boosting	31
3.9	Combining decision trees	31
4	AVATAR Approach to Parallel Learning	32
4.1	Rule Evaluation Metrics	35
4.2	Conflict Resolution	36
4.3	Weighted voting	37
4.4	Simulated parallel results	37
4.5	Discussion	38
4.5.1	Improving highly parallel performance	40
4.6	Results from weighted voting	41
5	Parallel Learning Implementation Status	42
5.1	Experiments on the ASCI Red	42
6	Experience with ScalParC: Why Its Decision Tree Fails	44
6.1	Introduction to Decision Trees	45
6.2	Claims that ScalParC makes	45
6.3	The ScalParC argument for why SPRINT is not scalable	45
6.4	Examples of what does not work in ScalParC:	45
6.5	Why ScalParC does not work:	47

CONTENTS

6.5.1	Parallel Hashing Tables	47
6.5.2	ScalParC claims to be Scalable	48
6.5.3	Perform-Split-I:	48
6.6	Summary	48
7	Summary	49
8	Bibliography	51

LIST OF FIGURES

List of Figures

1	Pertinent features in a large-eddy simulation.	8
2	Depiction of approaches to labeling and learning potentially large visualization data sets.	10
3	Summary	11
4	AVATAR front panel	12
5	a) First frame and b) Middle frame c) Last frame	13
6	A view of AVATAR features panel	14
7	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.	15
8	A training frame with marked regions.	16
9	A second training frame with marked regions.	17
10	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.	18
11	Node correctly classified as very interesting.	19
12	Node incorrectly classified as very interesting.	20
13	Node correctly classified as interesting.	21
14	Saliency set as interesting	22
15	Region marked as interesting	23
16	An example of the language of the file generated for training	24
17	The attributes file generated for Can Data set	25
18	Options in applying from one to many classifiers on one to many data sets. .	33
19	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.	34
20	Results from PIMA dataset experiments.	39
21	Voting with Ripper on the Pima data set using 3 different approaches vs. conflict resolution.	43
22	Tree for Iris from running Scalparc with the gini metric.	46
23	Tree for Iris from Scalparc using the information gain metric.	46
24	Legal tree from C4.5 for the Iris data.	47

LIST OF TABLES

List of Tables

1	Results on the Iris data set using 10-fold cross-validation for a 2 processor partition. sd - standard deviation.	38
2	Results from voting with Ripper and the Pima data set.	42

1 Introduction

Modern scientific data sets are so large that they can fundamentally disrupt the “one-to-one” principle which previously governed data visualization. It is no longer possible to efficiently view all the data. A pressing example is the 3D high-resolution physics data sets generated by simulation codes under the DOE’s Accelerated Strategic Computing Initiative (ASCI) program.

These simulations are replacing important physical experiments, and so must be conducted in the most exacting detail. But the very quality of that detail poses a conundrum, for if a scientist wants to visualize the data at the finest resolution available, in order to catch important and subtle details, they will not have time to look at all of it, and so may not find all of the areas of interest. And vice versa, of course. If they back off and work at a resolution that insures they will see the full extent of the data set, they will miss small anomalies. An example is in Figure 1, which shows the instantaneous x- and z-vorticity fields for a large-eddy simulation with a Reynolds number of 10,000. This flow exhibits Taylor-type instabilities and Taylor-Göertler 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 straightforward to recognize once located, they are difficult to describe or articulate a priori.

Figure 1: Pertinent features in a large-eddy simulation.

Thus it is necessary to make visualization tools smarter, to let them begin to understand what in the data is of interest to the user, so that the tool can navigate, as well as illustrate, the data. One avenue is to invoke pattern recognition techniques, using a visualization session as a supervised learning session. This will require being able to learn decision rules from very large data sets, which is challenging, as the data will not fit in memory and the amount of time needed to sift the data in a serial fashion will be exorbitant.

In other words, it is desirable to have a tool that will indicate which portions of a data set, when visualized, are likely to have *interesting* information contained within them. Such

2 The Visualization Interface

a visualization tool has the potential to greatly reduce the time needed to find and examine the relevant areas of a data set. One role of such a tool is to learn a model of what makes a region of data interesting from a set of labeled examples. This model may then be applied to new data sets and can provide labels to the elements and regions of these data sets.

Another role of the tool will be the display mechanism. This mechanism can display salient or interesting regions to the user upon request. There may be levels of saliency, such as very salient, salient, somewhat salient, etc.

With very large visualization data sets there are several challenges to effective model learning. The first is that the visualization data is typically not labeled by saliency. Hence, that this tool must enable experts to label *typical* data sets is required. Further, the data sets must be acquired and labeled. It is possible that not all data will be browsed, in which case default labels may be used. Figure 2 summarizes the issues in obtaining training data and learning from it.

After labels are obtained there is the problem of learning with very large data sets. Typically learning algorithms are time consuming even with data sets on the order of several thousand examples. A visualization data set with over a terabyte of data is likely to translate into hundreds of thousands to many millions of labeled examples. It is also possible for visualization data sets to have many descriptive features and most machine learning algorithms do not scale well with the increase in features [1].

The labeled data sets for training may thus take too much time to extract models by learning or they may be too large to fit in main memory of even memory rich modern computers. One approach to learning from very large data sets is to do the learning in parallel on a set of networked computers or on parallel machines. Both the speed and data size issues can be addressed with this approach.

In this document, we provide an interim report on the development of the sort of software discussed above, software which uses a standard visualization tool to label and display salient regions, and parallel pattern recognition behind the scenes to handle the huge volume of data. We call this software AVATAR (Adaptive Visualization Aid for Touring and Recovery). It integrates approaches to gathering labeled training data, learning from large training sets utilizing parallelism and the final display of salient data in unseen visualization data sets.

In the sections to follow we will separately discuss the visualization issues and the parallel pattern recognition issues, in both cases laying out the technical content, the past and future goals, and the current status of the software. We will also discuss a few of the less fruitful directions we explored, and end with a summary of the highlights of the current efforts and the plans for future work.

2 The Visualization Interface

2.1 AVATAR and MUSTAFA

As was just discussed, large scale data sets are becoming ubiquitous. Intelligent visualization is needed to unearth the information embedded in the terabytes of data. To aid intelligent vi-

2.1 AVATAR and MUSTAFA

Requirements for Expert Labeling of Training Data			
		unobtrusive (implicit) labeling	explicit labeling
Required Amount of Labeled Data for Learning	small	ideal situation	ok for expert, ideal for compute power required for learning
	large	ideal for expert, parallel computing needed for learning	taxing on expert, parallel computing needed for learning

Figure 2: Depiction of approaches to labeling and learning potentially large visualization data sets.

2.1 AVATAR and MUSTAFA

sualization, 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 the user 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 a result. An avatar can be unique to each user and attached to a type or category of data set.

The entire procedure behind the visualization and creation of an avatar can be summarized as shown in Figure 3. If a personalized version is desired the visualizer will need to help label training data. It may be necessary for the visualizer to explicitly label data of interest. Alternatively, the system may try to unobtrusively infer interesting regions from

- number of actions taken on regions,
- time spent browsing regions,
- specific types of actions or sequences performed on regions, and
- other metrics that may be developed.

Figure 3: Summary

The MUSTAFA visualization tool [2, 3] has been modified to facilitate the process of creating avatars. (Figure 4 displays the AVATAR front panel.) 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 pattern recognition system (to be discussed in Chapter 4) 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.

Disjoint subsets of labeled data, generated during a training session, will be created. Decision trees will be learned on each of the training sets in parallel and converted to rules. The rules will reflect the saliencies selected by the user. The rules will be combined together into a single model. This rule model will be the learned representation of regions likely to interest the user.

2.1 AVATAR and MUSTAFA

Figure 4: AVATAR front panel

2.2 Sample Data set

2.2 Sample Data set

Experiments were run on the “Can” Exodus [4] data set. This data set has 10088 nodes, 9 data components at each node and 44 time steps. Figure 5 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 5: a) First frame and b) Middle frame c) Last frame

2.3 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 $\rightarrow$ Saliency 1
2. Unseen low $\rightarrow$ Saliency 2
3. Not interesting $\rightarrow$ Saliency 3
4. Probably interesting $\rightarrow$ Saliency 4
5. Interesting $\rightarrow$ Saliency 5
6. Very interesting $\rightarrow$ Saliency 6

There is a separate panel available to the user with all the AVATAR features for training. Figure 6 shows the panel with all the AVATAR features currently available for the training phase. The user can change the current saliency as required. When a user finds something interesting or uninteresting, he/she can define a rectangular geometry of the 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,

2.3 How to train?

the saliency 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[5] input format. (C4.5 is the base pattern recognition engine used in AVATAR; it will be discussed at length later in this report.)

The screenshot shows the 'AVATAR Features' window. At the top is a title bar with the text 'AVATAR Features'. Below the title bar are two buttons: 'Data Query' on the left and 'Help' on the right. In the center, there is a 'Classes' label followed by a dropdown menu currently showing 'Unknown' and a small arrow button. Below this is a 'Selected' button. Further down is the section header 'RECORD SALIENCIES'. Under this header is a 'Browse...' button. Below the 'Browse...' button is a group box containing two checkboxes: 'Append' and 'Record all time steps'. At the bottom of the panel is a 'Write node values..' button.

Figure 6: A view of AVATAR features panel

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 5 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 the C4.5 decision tree learning program 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). Figure 8 and Figure 9 show snapshots of the training frames.

Training output

Figure 7: 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.

The plus in the training images indicates a node within the region of interest. Figure 8 shows the nodes classified as interesting by the user for time steps 3 and 11 respectively, figure.9 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 7 the flowchart of training input and training output is depicted.

2.4 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. This is one way the user can view the resulting saliencies, as 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. This is mainly used as a debugging aid.

2.4 Testing

Figure 8: A training frame with marked regions.

2.4 Testing

Figure 9: A second training frame with marked regions.

Testing output

Figure 10: 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.

A potentially more convenient approach is to note that another output is an Exodus file similar to the one tested is created, with the saliency added as a nodal variable. This can be loaded into MUSTAFA and then saliency viewed as any other nodal variable would be viewed.

A flow chart of the testing session is depicted in figure 10.

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

Figure 11 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 12 shows the incorrect classification of a node as very interesting, since the nodes in the region away from the section of the can being crushed received a saliency of interesting during training.

Figure 13 shows a node correctly classified as interesting, since the nodes at this vicinity

2.5 Current Status

Figure 11: Node correctly classified as very interesting.

were given a saliency of interesting during testing.

2.5 Current Status

2.5.1 Current Capabilities

The MUSTAFA visualization tool [2, 3] is being modified at the University of South Florida to provide an ability for intelligent visualization and creation of avatars. The current version of the MUSTAFA AVATAR software is version 1.1b. This version has the following capabilities:

1. The user has the ability to set the desired saliency at the time of browsing. The saliency value used is displayed in the AVATAR Features panel. Figure 14 shows that the user has set the saliency to be interesting (integer value 5) for browsing.
2. The user can define an arbitrary geometry for the region of his interest on the data set and all the nodes inside that geometry are labeled with the desired saliency. Figure 15 shows a region identified by a user as interesting.

2.5 Current Status

Figure 12: Node incorrectly classified as very interesting.

2.5 Current Status

Figure 13: Node correctly classified as interesting.

<u>D</u> ata Query	<u>H</u> elp
--------------------	--------------

Classes **Interesting**

Selected

RECORD SALIENCIES

Browse...

☐ **Append**
☐ **Record all time steps**

Write node values..

Figure 14: Saliency set as interesting

Figure 15: Region marked as interesting

2.5 Current Status

```
0,0,0,0,0,-5000,-4.66791e-08,1.26859e-07,-8.51591e-07,2.
0,0,0,0,0,-5000,-7.82527e-08,1.69145e-07,1.704e-06,2.
0,0,0,0,0,-5000,7.82527e-08,-1.69145e-07,3.27827e-06,2.
0,0,0,0,0,-5000,4.66791e-08,-1.26859e-07,-1.63954e-06,2.
0,0,0,0,0,-5000,-4.66791e-08,-1.26859e-07,-8.51591e-07,2.
0,0,0,0,0,-5000,-7.82527e-08,-1.69145e-07,1.704e-06,2.
0,0,0,0,0,-5000,7.82527e-08,1.69145e-07,3.27827e-06,2.
0,0,0,0,0,-5000,4.66791e-08,1.26859e-07,-1.63954e-06,2.
0,0,0,0,0,-5000,-4.66791e-08,1.26859e-07,-8.51591e-07,2.
```

Figure 16: An example of the language of the file generated for training

3. At the end of the browsing session, a flat file in the language compatible with C4.5 is generated for training, as was explained in Section 2. Figure 16 shows a snapshot of the data file used for learning and Figure 17 shows the attributes file generated at the end of training session and used for learning.
4. The browsing session can be done in phases. The user can record the nodes and the saliency in a flat file. When he/she restarts the browsing session, the result of the new session can replace or be appended to the original file, as desired.

2.5.2 Region Display Issues

The following AVS/Express modules were tried for the display of salient regions. Discussed below are the problems with incorporating them in AVATAR.

1. **GDroi_2Dclick**: This template defines a module to create a 2D drawing with a series of mouse clicks. The GDroi2d_click module generates a drawing with screen space XY positions provided one at a time typically from a UIonePoint interactor [6]. The routine in MUSTAFA for data probing, captures the X,Y,Z coordinates from the probe point without the call to the interactor routine. The UIonepoint interactor routine overrides the mouse response for the probe point and thus does not allow the capture of the X,Y,Z coordinates.
2. **draw_line module**: The draw_line [6] module draws polylines. This gives the user the choice to draw rectangular shapes, by clicking at points and simultaneously giving the object coordinates to the node selection module. The concern with this approach is, that the region of selection is not completely displayed, if the user is viewing the element blocks. This module draws polylines (in x,y,z coordinates) to connect the cells clicked by the user. The region selected can be displayed if the user is visualizing the mesh/geometry. Element blocks attributes color to variables, making the structure opaque and thus the polylines sometimes pass through the surface to establish the connection and are not displayed to the user.

2.6 The Journey Ahead

```
|classes
1, 2, 3, 4, 5, 6.

|attributes
DISPLX : continuous.
DISPLY : continuous.
DISPLZ : continuous.
VELX : continuous.
VELY : continuous.
VELZ : continuous.
ACCLX : continuous.
ACCLY : continuous.
ACCLZ : continuous.
```

Figure 17: The attributes file generated for Can Data set

3. **AGRectangleObj**: This was incorporated in AVATAR. AGRectangle [6] defines a rectangle. The problem with this is, after the user has defined the region of interest-ness, he/she has to click at the four corners to give the data set coordinates. Also, with this approach the user has the option of a rectangular geometry only.

2.6 The Journey Ahead

Currently, the information that is captured through the visualization tool is in a local context. That is, only the nodes are written. It is desirable to have the global context stored as well. The global context could include the geography of the region, the viz operation performed to reach a particular region in the data set; basically the current state of the data set along with the nodal information should be saved.

The global information is important, for imagine a user who is at time-step 1 and doesn't find anything important. They perform a viz operation (without changing the time-step), and then discover something important. The variables at the nodes will have the same values before and after the operation. The fact which makes the region interesting is the operation performed on the data set.

A way to do this could be to have a menu option available to the user, wherein the user enters the reason for marking some region interesting or not interesting. But, this could be cumbersome for the user if he/she has to do the same for every region he/she marks. Hence, a desirable system would be to have an agent attached to the user's session that saves the state of the data object and maybe some history of actions that lead to that state, when the user identifies something as interesting. The user could have an option of setting an agent or not and also the amount of interaction he/she wants with the visualization tool.

Another desirable feature could be the user enters a property, say Velocity ≤ 100 , and

3 Review of Approaches to Distributed or Parallel Learning

the Visualization tool takes the user to all the regions adhering to that property. Then, the user could assign the desired saliency to that region. This may be useful for an expert who is interested in the influence of a few variables on a simulation.

An important aspect here is— *how to best display salient regions to the user* while the user is trying to indicate what is salient and what is not? Some approaches, to be explored, would be as follows.

1. To have a color code attached to each kind of saliency and as the user selects a particular saliency the corresponding color code spreads in the selected region.
2. To allow the user to draw the region free hand on the data set.
3. As the user marks the points on the data set, polylines are drawn to join the points and thus form a multi-sided polygon.

Note that once the saliency data has been saved, it can be viewed as any nodal variable, or with the HTML interface described earlier. The issue here is how to provide immediate feedback on the regions being selected before any post-processing occurs.

After an avatar is created, the avatar should be able to guide the user it was created by/for, through another data set of the similar domain that the avatar was learned on. For example, it might be another version of the same experiment from which the training data set was created. The avatar should show the user the salient regions based on the experience. There could be an option wherein the user enters the saliency of interest and the avatar walks the user through the regions having that saliency one by one. Each user could have a different avatar attached to a different data set and depending on the domain the user is currently interested in, the related avatar could be brought up. Also, there could be an option for one expert to see another expert's perspective on a problem, by bringing up an avatar generated by that expert. In this aspect, if the global information is saved, that is the actions of the user, another expert's avatar will be self-explanatory to another expert. It will enable a better understanding of a problem.

The desired result would be to have an intelligent visualization system that would learn from a user, adapting dynamically to the user and data set. It should be able to aid the user in visualization and once the user profile (avatar) is created, guide the user using the avatar through an unseen data set.

3 Review of Approaches to Distributed or Parallel Learning

The previous section discussed a user-centric view of AVATAR, focusing on the visualization interface, its functionality and how it can be enhanced. Equally important, but not as readily evident, is the underlying pattern recognition system which creates and interprets the avatars. This learning system must necessarily operate in a parallel fashion, given the huge and already parallelized nature of modern physics datasets.

3.1 Overview of Rule-Based Learning Algorithms

There are two main approaches to doing learning in a distributed or parallel system. For one, an existing machine learning algorithm, such as a decision tree learning algorithm, may be parallelized for distributed computation over a single data set [7, 8, 9]. A second is to apply a set of learning algorithms in parallel to training data sets. These data sets may be disjoint [10] or partially overlapping [11, 12] or one can apply different types of learning algorithms to the same training data in parallel.

In Section 6 we provide a discussion of the first approach (including a scalability analysis) in the context of parallelization. Though such an approach has the conceptual advantage of generating the same model as would have been generated sequentially, our conclusion is that the scalability of parallelizing decision trees has limits that may be problematic for use with very large training data sets.

In this section we concentrate on the second approach, on learning algorithms, approaches to applying the algorithms in parallel on distributed processors, and then combining the resultant learned models. The model for learned knowledge may be a neural network [13], a decision tree, a set of rules, etc. There has been significant work with rule-based systems which are, perhaps, the easiest to understand. In this report, we concentrate on rules as the ultimate learned model.

3.1 Overview of Rule-Based Learning Algorithms

Decision tree learning is fast and produces accurate classifiers [5, 14]. Rules can be created by traversing paths from the root node to each leaf. Rules may be created from a pruned or unpruned decision tree [15]. The rules may also be further processed [16, 5], which has proven to be quite time consuming as the number of training examples increases [9]. In this research we propose evaluating rules taken directly from pruned and unpruned decision trees. Each processor will typically have a large training set and a decision tree will provide a fast mechanism for producing rules.

Attribute tests in the decision tree will use information gain as implemented in C4.5 release 8 [17] where the bias towards attributes with more values is mitigated for both nominal and continuous attributes. The choice of a test to split continuous examples at any node in a tree utilizing information gain is briefly illustrated below.

For simplicity, let's consider only continuous attributes and tests of the form $A \leq t$, for attribute A , which has two outcomes, *true* and *false*. At a node D the training cases are sorted on the values of attribute A to give a set of distinct values $v_1, v_2, \dots, v_m$ ordered from low to high. Then $t = v_i$ produces a corresponding partition of the examples at D . Let c denote the number of classes and $p(D, j)$ the proportion of cases in D that belong to the j^{th} class. The information in D is

$$info(D) = - \sum_{j=1}^c p(D, j) \times \log_2(p(D, j)). \quad (1)$$

3.1 Overview of Rule-Based Learning Algorithms

The information gained by a test T with k outcomes is

$$Gain(D, T) = info(D) - \sum_{i=1}^k \frac{|D_i|}{|D|} \times info(D_i) \quad (2)$$

where D_i is the subset of examples resulting from the i^{th} outcome of the test T . The information gain is biased towards tests with many outcomes, or continuous attributes which have many distinct values, which results in lots of possible tests. This bias may be removed by appropriate modification of the gain for continuous [17] or nominal attributes [5]. The modified gain is used in all experiments reported here.

A node in our decision tree can be recognized as a leaf based upon the application of one of the information tests discussed in the proceeding. Alternatively, for pruned trees, error estimates may be used in the usual way to determine when a node is a leaf [15].

As part of the research a direct rule-generating program based on RIPPER will be utilized [18]. It is approximately as fast as building a decision tree and generating rules from the tree. Briefly, RIPPER works as follows.

Uncovered examples are randomly partitioned into two subsets, a rule *building* subset and a *pruning* subset. Typically, the building set contains 2/3 of the examples. A rule is built up from an initially empty conjunction of conditions. Any condition of the following form can be added to the antecedent of the rule: $A_n = v$, $A_r \leq \alpha$, or $A_r \geq \alpha$, where A_n is a nominal attribute and v is a legal value for A_n , or A_r is a continuous attribute and α is a value of A_r which occurs in the training set. The BuildRule function repeatedly adds the condition that maximizes FOIL's [19, 20] information gain criterion until the rule has no false positives in the building data set.

The rule is pruned immediately after being built. Any final sequence of conditional tests may be deleted from the rule. The deletion that maximizes the following function is selected:

$$v(Rule, PrunePos, PruneNeg) \equiv \frac{p - n}{p + n} \quad (3)$$

where p is the number of examples in the pruning set that are correctly covered by the rule (and $PrunePos$ is the set of positive examples in the pruning set) and n is the number of examples erroneously covered by the rule (and $PruneNeg$ is the set of negative example in the pruning set).

A minimum description length test introduced by Quinlan [21, 18] is used to determine when to stop adding rules. Rule building halts when the total description length of the new rule set and examples is more than d bits larger than the smallest description length obtained so far, or when no more positive examples exist. RIPPER has used $d = 64$. Next the rule set is simplified by examining each rule in turn, beginning with the last one added and deleting rules so as to reduce the total description length. Finally, one or more optimization steps are applied, in which replacement and revision rules are constructed for each rule. One of the original and two new rules are chosen to be in the final rule set based upon the minimum description length heuristic [18].

3.2 Distributed learning with RL

RIPPER orders classes by their cardinality in the training set. It then builds rules for the smallest class first, treating all examples for other classes as negative examples. Next, it builds the rule set for the second largest class and continues this process until all but the largest class have been processed. Any example not covered is by default put in the largest class.

The remainder of this section outlines various approaches to using these basic rule-based learning concepts in various parallel and/or distributed contexts.

3.2 Distributed learning with RL

Provost and Hennessy [10] developed a distributed rule learner which partitions examples across a network of conventional workstations each running an instance of a rule learning program, RL[22]. RL performs a general-to-specific beam search of a syntactically defined space of rules, looking for rules which satisfy a user-specified evaluation criteria. The *cf_normalized* criterion defined in (2) was used. Examples were partitioned into $n \leq 4$ disjoint subsets. This is significantly smaller scale parallelism than will be investigated with AVATAR.

Experiments were done on a single data set describing cellular telephone fraud with each example consisting of 31 attributes. The training data set contained over one million examples. They found that the final rule set produced by the distributed rule learner contained all the same rules as RL applied to the full training set as well as a few other globally satisfactory rules. A separate dedicated processor was used to test learned rules on the full training data. The time to process the million examples in parallel was much faster than with sequential RL. Also, as memory limitations (thrashing) begin to affect the sequential version, the time savings becomes non-linear in the number of processors.

Further time reduction was obtained by allowing processors to broadcast good and bad rules as soon as they were found and allowing evaluation on a dedicated processor. For example, all specializations of a poor rule can be prevented by telling each processor about the rule. Processors can ignore examples covered by good existing rules and focus on other parts of the rule space.

3.3 Distributed learning with SKICAT

Fayyad et. al. [23] built rules to classify objects detected in a sky survey. The objects were represented in over 3 terabytes of image data, but the training data extracted was significantly smaller. They broke the data into 10 subsets of about 1400 objects, each described by more than 40 attributes. The subsets are not disjoint as proposed here but are random subsets of the original data. Rules were learned on each of the subsets and combined. Results improved based on the combination of the independently learned rules. Decision trees were used and given to another program to create rules [24]. There is no discussion of conflicts appearing in the combined rule set. The combined rule set is reported to outperform learning on the whole data set.

3.4 Distributed learning utilizing Meta-Learning

Stolfo et. al. [25, 26] have developed an “agent-based” distributed data mining system. The independent agents utilize their own classifiers on distributed and possibly independent data sets. The results from the independent classifiers are then combined into a single classifier with the use of meta-learning.

A set of validation data is used in the meta-learning process. The meta-classifier is built using the predictions of the n base classifiers on the validation set. Hence, it learns to utilize the outputs of the base classifiers to produce an accurate meta-classifier. A potential drawback of this approach is that as the number of classifiers increases the time to recognize an individual example also increases. There is also premium importance on the validation set being a good representation of the distributed data.

In [27] five base classifiers were used on 0.5 million examples involving credit card fraud. The classifiers included 3 decision tree learning algorithms, a naive Bayes classifier and RIPPER. The data was partitioned into 12 non-overlapping subsets, one per month. A 6-fold cross-validation was utilized where each classifier was trained on each of 10 subsets of the data. Meta-level training was done on one subset and meta-level “testing” for classifier pruning or removal was done on the remaining subset. A performance improvement over the best base classifiers is claimed.

In [28] it is shown that a distributed set of learners can be combined to recognize credit card fraud. The system is shown to save a company money by focusing attention on transactions that are both fraudulent and expensive. No analysis was done on accuracy of distributed learning versus central learning, on speedup as a function of degree of parallelism, or on how degree of parallelism affects accuracy.

In [11] it was shown that partially replicating training data among a set of distributed learners did not generally lead to better performance than using disjoint sets of training data. The learned models were combined in different ways with the conclusion that replication of training data buys nothing over combining models built from disjoint training sets. The study was done with only 2 data domains, and so the conclusion may not be universally applicable.

3.5 Bagging

Classifiers have been created by bagging [29, 30, 31] and then combined, usually with a voting scheme. Bagging involves randomly selecting examples with replacement for each of n training sets. It is possible that some examples will not participate in any training set with this approach and that could be counterproductive in the case of large, diverse training sets. The performance of bagged classifiers has usually exceeded that of a single classifier providing conflicting results with the work discussed in the previous paragraph. It is possible that the particular domains chosen in [11] have features which prevent bagging from helping.

3.6 Unevenly distributed example sets

3.6 Unevenly distributed example sets

When there is a very uneven class distribution of training data, replicating training examples from underrepresented classes with otherwise disjoint data may enable better classifiers to be built. In [12] a model of fraudulent credit card transactions was learned with very few of the training examples actually being fraudulent. To allow each distributed classifier to have a balanced set of training examples, the fraud examples were copied to exist in each otherwise disjoint training set. The distributed classifiers were combined via a variant of voting and the approach produced better performance than even training with the full available data set.

In recent work Domingos [32] showed a classifier independent approach to getting good performance with vastly different class distributions. His approach does potentially require some extra computation and a scheme for weighting errors, but appeared quite good in allowing models of even small classes to be effectively learned.

3.7 Correspondence Analysis

Correspondence analysis [33] has been used with multiple different classifiers trained on the same data. In conjunction with stacking it was found to be competitive with or better than a bagging approach using “plurality voting” to choose an example’s class.

3.8 Boosting

Boosting [34, 35] has been shown to provide excellent performance when utilizing multiple classifiers [30, 31]. The combination of the classifiers is typically weighted based upon their performance. However, in boosting the training set for the k^{th} classifier is based upon the results obtained from the $k - 1$ classifier. Thus it does not lend itself to a parallel approach.

3.9 Combining decision trees

Shannon and Banks [36, 37] have developed a method of creating a single decision tree from n trees created from continuous attributes. The n trees are created on almost completely overlapping training sets. The final tree is a “central tree” which is closest in form to the n learned trees. After the form of the tree has been determined (i.e. attributes for splitting at each level has been chosen), the values to split at each node must be determined from some training set. The resultant tree, apparently with final training done on the full training set, was found to outperform any of the individual trees, as well as a tree created on the full training data. This approach will not work well unless the distribution of tree shapes is unimodal. If more than one shape produces equivalent accuracy trees, the average of the structures might have poor accuracy.

4 AVATAR Approach to Parallel Learning

There are a number of open issues in learning when the training data sets are very large. Perhaps the most basic is whether subsampling of the data set can be used to reduce the time cost of applying a learning algorithm while preserving the accuracy of learning from all available labeled data. It seems certain that in some cases subsampling can be done with no accuracy penalties [38, 39]. However, whether subsampling can be successfully done for all data sets, and for physics mesh data sets in particular, is an open question.

A second issue is whether to parallelize a single algorithm (decision tree learner [7] or rule learner [9]) or to use an approach where models are learned independently and combined. We believe that scalability in time and amount of training data processed will be better with an approach in which learning algorithms operate on separate processors on separate training data sets. The classification models that they learn must then be combined in some way. The models may make conflicting classifications and any individual model may not adequately cover the domain on which learning was applied.

A third issue is unbalanced populations. Learning with equal numbers of examples in each class is often not possible on real world data. In particular, we expect a high ratio of negative to positive examples for physics mesh data. To deal with skewed training data sets, metrics that weight errors based on class size [10] or perhaps cost matrices for error weighting [32] may be used. The best approach to dealing with examples from highly skewed classes remains a topic for research.

Another issue is what type of model to learn: a neural network, a decision tree, a set of rules, a nearest neighbor, a Bayes classifier, etc. Decision trees are fast to learn but can be complicated to combine [16, 5]. For large training sets feedforward, backpropagation neural networks do not seem feasible. Radial basis function networks may be used, but may be challenging to understand. Combining multiple networks may also be challenging. Rules are the easiest models to combine and understand, hence we choose to learn models that can be turned into rules.

An important issue in combining multiple models is whether the accuracy will be equivalent to the accuracy of a single model learned on the full data set. Certainly, this issue argues for the parallelization of a single learning algorithm where we can be sure that the parallel result matches the sequential result. For training data sets that are larger than main memory/swap space it is not possible to determine whether the accuracy of a model created from a distributed set of learners is equivalent to that of a single model learned from the full training data set. Figure 18 shows the different viable approaches to learning with multiple classifier instances (≥ 1) and multiple data sets (≥ 1). In the case of the top right corner there may be no overlap for very large training data sets.

We have demonstrated that rule sets learned from a full training data set and a distributed set of data can consist of different rules [40]. For example, in Figure 19, we show a small illustrative data set for which the rules learned by merging disjoint rule sets built on a disjoint 2 partition of the data do not include any rules learned by training on the full data set. Information gain is used to choose the attribute test for each node in the decision tree

		Data Set Used In Learning	
		single common data set	multiple partially distinct data sets
Classifier(s) Used In Learning	varied types	"combination of classifiers" - demonstrated improvement	??? unknown gains over bagging or combo of classifiers
	one type	traditional pattern recognition approach	"bagging" - demonstrated improvement

Figure 18: Options in applying from one to many classifiers on one to many data sets.

[17] from which the rules were generated.

Figure 19 shows that a merged rule set, with each of the constituent rule sets developed in parallel on a disjoint training set, may in the extreme contain no rules in common with the rules created by training on the union of all the subsets (i.e. the full training set). The final merged rule set will depend upon how the examples are partitioned. The mix of examples needs to reflect the mix of available training examples.

The example data set and results from it shown in Figure 19 suggest that the accuracy of merged rules may well be different from the accuracy of the rules created from the full training set. Our experimental results will examine how different the accuracy may be and how it is affected by the number of partitions made from the training data.

Several approaches to combining models were discussed in the last section. Here we will discuss two approaches, *conflict resolution* and *weighted voting*. The conflict resolution approach tries to produce a single set of rules in which only one rule can fire at a given time. The weighted voting approach will produce a set of weighted rules and more than one rule may fire for a given example. The class of the example will be determined by the sum of the weights of the rules which fired and put the example in that class.

4 AVATAR Approach to Parallel Learning

Figure 19: 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.

4.1 Rule Evaluation Metrics

How should a rule be evaluated? The seemingly obvious answer is to use a measure of the true positive examples and false positive examples covered by the rule. Such a straightforward measure is fine if the training/testing examples are equally distributed among classes. Unfortunately, in real world data sets it is rarely the case that training examples are uniformly distributed across classes. In fact, it is common for items in “important” classes of data to occur less often. Below we examine some candidate metrics for ranking rules.

Some terminology is needed to precisely define an acceptable rule. Let a rule r provide a class description. The rule r will either cover or not cover each example of a data set E . A rule covers an example when it *fires* or is activated by the example. Coverage statistics can be developed for a given r and E .

Let P and N be the numbers of positive and negative examples in E for the class of rule r . The number of true positives, TP , will be the positive examples covered by r and the number of false positives, FP , will be the number of negative examples covered r .

A rule may be evaluated for acceptability based upon a *rule evaluation function*, $g(r, E)$, which takes a rule and an example set and produces an evaluation score. A rule may be deemed acceptable when its evaluation score is above a *threshold* t or $g(r, E) > t$. In particular, the following 3 metrics have been utilized in the literature:

- The positive predictive value [41],

$$g(r, E) = ppv(r, E) = \frac{TP}{TP + FP} \quad (4)$$

- A modification of the certainty factor metric normalized for skewed distributions [10, 14].

$$g(r, E) = cf_normalized(r, E) = \frac{TP - 0.5}{TP + \rho FP}, \quad (5)$$

where ρ is the ratio of positive to negative examples in the training set for the class of r .

- The Laplace accuracy estimate [42, 43, 21].

$$g(r, E) = le(r, E) = \frac{TP + 1}{TP + FP + k} \quad (6)$$

where k is the number of classes in the data.

An important question is whether a rule that will be considered acceptable on the full set of examples E will also be considered acceptable on at least one disjoint subset of examples taken from E for some $g(r, E)$ and $\forall t$, so that it is at least possible that the sequential and parallel processing might generate the same set of rules. Let E be partitioned into N disjoint subsets, $E_i, i = 1 \dots n$. It is clear that, for the positive predictive value, a rule that is acceptable on the full training data set must be acceptable on at least one subset. This

4.2 Conflict Resolution

is called the *invariant partitioning property* [10]. However, for the other 2 metrics it can be shown that this property does not hold.

It is desirable for all rules that would be found acceptable when training on the full data set to also be found when training on disjoint subsets of the training data. In this way it would be possible for the parallel learning approach to produce exactly the same rules as would be produced from learning applied to the entire training set. One way to ensure this is to relax the criterion for rule acceptability on the disjoint subsets. For example, we could let $g'(r, E, n) = le'(r, E, n) = \frac{TP+1/n}{TP+FP+k/n}$ and $g'(r, E, n) = cf_normalized'(r, E, n) = \frac{TP-0.5/n}{TP+p/nFP}$, where N is the number of data partitions. Now, for a given $t = L$ the *invariant partitioning property* will hold. Of course, the cost is that more learned rules will tend to be found acceptable on the subsets of data. Further we have shown earlier that the same set of rules may not be learned in parallel as will be learned sequentially.

4.2 Conflict Resolution

Rules learned in parallel may put the same example into different classes. We call any rules that do so for a specific example *conflicting rules* and the set of such rules is a *conflict set*. When rules are found to be conflicting the conflict must be resolved in some manner.

Rules can be found to conflict via a static analysis procedure [44]. Work in [44] discusses the various types of conflicts that may arise. We have built a rule conflict analysis system that identifies rules in conflict.

It can be helpful in resolving conflicts to have rule performance data on a set of test data. There are two possibilities for obtaining rule results on test data in the case that rule sets are learned on N distributed processors. Each processor on which a rule set is learned may have its own validation set of examples that were not used in the learning process. These examples can be used to provide an indication of the generalization capability of the rules. The accuracy of rules on such validation sets may be used to decide which rules to keep and which to throw away when a conflict has arisen among a set of rules.

Alternatively, rules may be passed on to the $N - 1$ processors on which they were not created and tested against the training data sets on those processors. This type of testing will provide a very good measure of a rule's generality.

Another approach to resolving conflicts is to specialize rules that are in conflict. That is, add conditions to the rule's antecedent so that the rule no longer conflicts with other rules [10, 40]. This requires that a set of conflicting examples be maintained, which would likely degrade scalability. This set of examples will be used to choose one or more attributes to add to a rule found to be in conflict.

The approach to dealing with conflicts that we have been utilizing is, for every set of rules in conflict, to allow only the rules from the class which has the best performing rule to remain. Now the conflict is resolved and only 1 rule will fire or apply for a given example. This approach also has the advantage of reducing the size of the final rule set. It is important that an appropriate metric be used that takes into account the number of examples covered, the proportion of examples in each class, as well as the accuracy of the rule.

4.3 Weighted voting

4.3 Weighted voting

An alternative to conflict resolution is to take rules from all learned models, remove duplicate rules, apply weights to the rules to be used in resolving conflicts among them and use all the rules to classify unseen examples. The key to this approach is to have an effective method of weighting rules.

The weighting can be simple with each rule given a weight of 1. Then in the case of conflicts an example will be assigned to the class for which the most rules fire. Ties might be broken by preferring the minority class in the case of unequal distributions of training examples or simply in an arbitrary fashion. This approach has a relationship to ensembles of classifiers [45, 46] with the major exceptions that in this case the data sets are potentially disjoint and all the classifiers are the same. It is also related to bagging [30, 31, 29] except that the training sets may have no overlapped examples where in bagging different training sets are created via sampling with replacement from the original training set.

In our approach weights can be assigned to rules by applying the rules from a model learned on one processor to the data used to train the rule models on the other $N - 1$ processors. The true positives and false positives from each rule can be utilized in the appropriate metric to get a performance weight in $[0, 1]$. Then when conflicting rules fire for an example, the performance weights for each class can be summed up. The class with the largest sum (the most good performing rules have assigned the example to that class) will be assigned as the class of the example. Ties may be broken arbitrarily.

An alternative to applying the rules learned on a processor to all other train data sets to get a performance weight is to have a validation set on each processor. The rules learned on a processor can be tested against the validation set. The performance on the validation set can be used in a metric formula to provide an individual rule weight for use in voting.

4.4 Simulated parallel results

In this section we discuss experiments done to show what kind of accuracy may be expected as a data set is partitioned into N sets of disjoint training data. The experiments reported here are from two datasets from the UC Irvine database [47] both of which consist of all continuous attributes. The IRIS data set [48] has 150 examples from three classes and the PIMA Indian diabetes data set has 768 examples from two classes. We are interested in how the accuracy is affected by partitioning these small data sets into N disjoint subsets, learning decision trees on the subsets, generating rules from the decision trees and then merging the rules into a final set of rules.

Our experiments are done using 10-fold cross validation [41]. For an individual data set and a given number of disjoint subsets, N , ten partitions of the data are made. Each consists of 90% of the train data with a unique 10% held back for testing. From each fold, N disjoint subsets are created. C4.5 is applied to each of the N subsets and rules are created from the generated decision tree. The rules created from the j^{th} subset of data are then applied to the $N-1$ other subsets. The accuracy of each rule must be greater than the chosen threshold, t , in order for the rule to remain in the final set of rules. The default for the

4.5 Discussion

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

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

threshold t was chosen as 51, which ensures that the developed rule is better than a default rule which assigns every example to the class of the rule. For the Iris data we chose $t=75$ rather arbitrarily. Setting t in an appropriate and systematic way must still be addressed.

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

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

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

A more interesting experiment is to look at a significant number of partitions. With the larger Pima data set experiments were run with $N = 2, N = 4, \dots, N = 10$, and $N = 12$. The results of a 10-fold cross-validation experiment with C4.5 using its default pruning ($cf=25$) were an average accuracy of 73.90% with $sd=4.26\%$ and an average of 23.8 rules. Figure 20 shows plots of accuracy, standard deviation and the number of rules for 10-fold cross validation experiments with each of the above N disjoint partitions. The performance of rules created from the unpruned tree, the pruned tree with the certainty factor of 25 and a certainty factor of 1 are shown. It can be seen that the accuracy of the rule set generally decreases as N increases. The standard deviation tends to get large suggesting that performance on some folds is quite poor. The number of rules that remain in the final set remains fairly constant as N is increased. There are significantly less rules, after conflict resolution, than when training is done on the full data set.

4.5 Discussion

The results obtained here come from small datasets. However, we believe the issue of rule accuracy falling off can also occur with larger datasets. Our results are consistent with those found in [11] where experiments were conducted on data sets related to the human genome project. It was found that more partitions resulted in lower performance on unseen data. Their approach to combining the classifiers was different, relying on voting among the learned set of classifiers.

Some success in mitigating the effect of learning in parallel may be gained by using a combiner or arbitrator approach to integrating multiple classifiers [26, 28]. However, such

4.5 Discussion

Figure 20: Results from PIMA dataset experiments.

4.5 Discussion

approaches entail retaining all N classifiers learned in parallel and may be problematic for large N . There is not a single model of the data either.

In [10] an approach similar to this one was used on a very large dataset (over 1,000,000 examples) and there was no drop off in accuracy for partitions up to $N = 4$. Our results suggest that accuracy would fall off as N increased further.

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

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

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

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

4.5.1 Improving highly parallel performance

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

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

4.6 Results from weighted voting

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

As a simple example of the potential for improvement consider the example of Figure 19. Assume you got the second set of rules using attribute 2 (Att2) from processor 1 (Proc1) and applied them to the examples held by processor 2. The rule

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

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

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

the rule will cover 3/3 examples correctly. Further it essentially matches the second rule obtained from the whole training set in Figure 19. This example shows how specialization would work.

On the Pima data [47], specialization was applied to the two partition case raising the accuracy slightly from 73.38% to 73.77%.

Rule specialization could be decided upon in other ways than in our experiment. One example is that, after learning, a pessimistic estimate of the rule's performance [5] could be generated. For a test subset on which the classification performance of the rule was more than $x\%$ below the estimate, specialization could be carried out. To get a better estimate of the performance of a specialized rule, it might be tested against all the data on which it was not created (e.g. broadcast to all processors). This would make the conflict resolution process more accurate.

4.6 Results from weighted voting

Experiments have been done with the Pima data set and are underway with a subset of the Forest Cover data set [47] (29,053 examples from 581,012). The Forest Cover data set consists of 581,012 examples with 12 attributes some of which have been binary encoded. The attributes relate to cartographic information and the goal is to classify 30x30 meter cells into 1 of 7 classes. The weighted voting results utilize rules generated by the Ripper [18] learning algorithm. The results are preliminary, but we report them because they appear to show that accuracy can be maintained with weighted voting as the number of partitions N gets quite large. Again, 10-fold cross validation is used for all results (hence the use of the subset of the larger data set).

On the Pima data set, using Ripper with optimization level 10, results are shown for power of 2 (and selected other) partitions in Table 2. What is striking about the results is that accuracy does not degrade in any significant way, even with a partition of size $n = 32$, which no person is likely to use on so small a data set (the train sets are only 22 examples on average).

5 Parallel Learning Implementation Status

Table 2: Results from voting with Ripper and the Pima 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 21 shows how voting and conflict resolution approaches compare using Ripper for partitions of $n = 2, n = 4, n = 8, n = 12, n = 16, n = 32$. Voting using the certainty factor metric as the rule weight, TP/FP as the rule weight and equal weights for rules (akin to bagging) are each displayed. In this domain, voting is better for $n \geq 16$. Among voting approaches it is unclear how to differentiate based on this preliminary experiment.

5 Parallel Learning Implementation Status

At the present time we have a distributed learning system implemented on the ASCII Red machine which consists of the following components:

- A version of C4.5 that can be run in parallel on multiple different training data sets. Each instantiation of C4.5 runs on a processor on one data set. Pessimistic pruning and all the usual C4.5 options are available.
- A modified version of C4.5rules which generates rules without modification from a pruned or unpruned decision tree. It can be run in parallel on n distinct trees.
- A program that will test n rule sets on each of the $n - 1$ training sets from which it was not generated. The results are sent to a central source for tabulation.

The developed parallel system also allows for timing of parallel and sequential versions of learning on the ASCII Red to be done. It is possible to learn and test learned rules on large datasets in parallel.

At the present time, we have not ported CLIPS for fast rule testing to the ASCII Red. Nor have we ported Ripper, although it is believed that the same process that is used to allow C4.5 to run in parallel can be applied to Ripper.

An example of a command to run a 10 partition of the pima data if the data sets are named pima0 to pima9 is: `yod -sz 10 c4.5 -f pima`.

5.1 Experiments on the ASCII Red

We have been able to obtain exact replicas of the rules for n partitions of the Iris and Pima data sets which were discussed in the previous sections when using our ASCII Red distributed learning implementation. In the following we provide some timing results from initial experiments with a brief analysis of the results.

Using the Pima data set and a four partition required 13.8 seconds for learning and 14.6 seconds to create rules from the tree. The testing phase in which all rules were tested

5.1 Experiments on the ASCI Red

Figure 21: Voting with Ripper on the Pima data set using 3 different approaches vs. conflict resolution.

6 Experience with ScalParC: Why Its Decision Tree Fails

against the $n - 1$ data sets they were not trained upon took 7.5 seconds. A decision tree build using the Pima data set on a single ASCII Red processor takes 8.7 seconds for training and 12.4 seconds for rule creation. Here the overhead of doing parallel communication and coordinating 4 sets of rules makes the parallel version significantly slower than the sequential version. The message is clearly to not parallelize really small data sets.

We also have run the large (581,012) example forest cover data set as a 10 partition and sequentially. The 10 partition required 295.5 seconds for learning and 10.05 seconds for rule creation. A sequential run required 3807.9 seconds and the rule creation phase failed. Here, we see a nice win for the parallel version (actually a super linear speed-up likely do to with disk swapping).

The rule testing code failed due to the binary attributes that are used in the Forest Cover data set. This can be rectified, we believe, in a fairly simple way. The testing code also had some trouble with the large number of sequential examples (again a code fix that should be simple).

6 Experience with ScalParC: Why Its Decision Tree Fails

In this section we discuss an implementation of parallel decision tree learning, which is claimed to scale well. A discussion of errors in the code is given, as well as a discussion of why scaling appears to be a problem for any parallel decision tree learning algorithm which has a general model like the one shared by ScalParC and Sprint [7].

This discussion requires clear definitions for some basic terminology before getting started:

Record id:	This is a unique identifier assigned to each record in the training set.
Training Set:	This is the set of data that the decision tree is going to be built on.
Continuous Attribute:	This is an attribute which can take on any real value.
Categorical Attribute:	This is an attribute with a finite discrete set of allowed values.
Class:	This is a categorical attribute for each record which is used for classification.
Decision Metric:	This is the function used to determine the split attribute (and value for continuous attributes) in the building of the decision trees.
Gini Index:	This is the decision metric used by ScalParC [8, 49].

6.1 Introduction to Decision Trees

Node Table: This is a mapping between a record id and the node it belongs to in a decision tree.

6.1 Introduction to Decision Trees

A decision tree is built by recursively splitting the training set according to a decision metric until a stopping condition is reached. The stopping condition is based on the class make up of the nodes in the decision tree. Building a decision tree is a two-part process. First, an optimal split point is determined by considering all possible split points and choosing the best one according to the decision metric. The second step is the creation of the child nodes for the recursive next step. The child nodes are created by dividing the training data set according to the chosen *split pointer*.

6.2 Claims that ScalParC makes

ScalParC claims to be scalable in both runtime and memory requirements: “A detailed analysis of applying this paradigm to the splitting phase shows that the overall communication overhead of the phase does not exceed $O(N)$, and the memory required to implement the phase does not exceed $O(N/p)$ per processor” [8]. The claim is that ScalParC has been shown to be scalable by running experiments on “up to 6.4 million records on up to 128 processors of Cray T3D” [8]. The further claim is that the stability is provided by the parallel hash table. ScalParC sorts the dataset at the beginning and the splitting phase needs to maintain this sorted order. ScalParC claims to build the trees in breadth-first order and to perform a single communication phase at each level of the decision tree— “Note that ScalParC could classify 6.4 million records in just 77 seconds on 128 processors. This demonstrates that large classification problems can be solved quickly using ScalParC” [8].

6.3 The ScalParC argument for why SPRINT is not scalable

The parallel SPRINT implementation builds the node table on *all* the processors. This results in the process being unscalable in memory. The large communication overhead needed to produce each of these local copies of the node table also makes the algorithm unscalable in runtime [8, 49].

6.4 Examples of what does not work in ScalParC:

However, as it turns out, ScalParC does not succeed where SPRINT is shown to fail.

Figure 22 shows a tree that results from running ScalParC, using the Gini metric for selecting split points, with data from the Iris data set.

Notice that the tree specifies an impossible split point where it has *petal_width* < 1.4 under a split of *petal_width* >= 1.8. This same split can be seen if the Information Gain measure is used in place of the Gini measure Figure 23:

A legal tree structure for this data set, as computed by C4.5, appears in Figure 24.

6.4 Examples of what does not work in ScalParC:

Figure 22: Tree for Iris from running Scalparc with the gini metric.

Figure 23: Tree for Iris from Scalparc using the information gain metric.

6.5 Why ScalParC does not work:

Figure 24: Legal tree from C4.5 for the Iris data.

The important thing to notice here is the very similar shape of these trees. At the second level of both trees, at the right most branch there are 46 examples in each tree. None of these examples should have *petal_width* < 1.7. But, if we in fact look into the ScalParC data structure at that branch of the tree we will find that it does contain a number of elements with *petal_width* < 1.7.

The conclusion is that the decision tree resulting from ScalParC's computation is *not* a valid decision tree. It sometimes specifies splits that could not happen, and it sometimes contains data at a node that should not be there. Examination reveals that the problem is not a simple programming bug, but rather a design error in which the data structures used are simply not capable of allowing the computation of a valid decision tree.

6.5 Why ScalParC does not work:

6.5.1 Parallel Hashing Tables

ScalParC uses a parallel hashing paradigm in the splitting phase of tree generation. The goal of this parallel hashing is to allow the algorithm to be scalable. This process can be described as follows: each value, *V*, on each processor has a given key associated with it. During the splitting phase each key, *K*, is hashed to produce both the processor, *P*, and local address, *L*, for each key. Each processor needs to hash each of the keys. As it is doing this it maintains a separate buffer for each of the other processors, into which it places the (*L*,*V*) pairs to be communicated for hashing. These buffers will be of very different sizes, from empty to size N/p , where p is the number of processors being used. Each of these buffers is communicated in a one step all-to-all personalized communication step. Each processor is communicating $p - 1$ buffers and also receives $p - 1$ buffers. Each processor then extracts the pairs from received buffers and either searches for or stores the values. The claim here is that this algorithm is scalable i.e. the all-to-all personalized communication is $O(m)$ as

6.6 Summary

long as m is $\Omega(p)$, the order of the processors. That is as long as $\Omega(p^2)$ keys are hashed at the same time the parallel hashing step is scalable.

6.5.2 ScalParC claims to be Scalable

In ScalParC the all-to-all personalized communication is done on a tree level basis. That is at each level of the tree all of the split determining phase is complete before any splits are performed. All processors therefore must wait on the slowest processor at each level of the tree.

Since the node table is divided evenly among the processors, a processor can receive at most $O(N/p)$ updates at each level. So, the receiving section should be scalable in both runtime and memory. But, the sending section of the process is often not scalable. Since the update set, examples to be split, at each processor is not even at each level of the tree both the memory and run-time of the all-to-all personalized communication will be controlled by the processors with the most updates to communicate. This step is not scalable. In the paper ScalParC admits to this, but goes on to say: "Although the formulation becomes runtime unscalable in such a scenario, it can be seen that no parallelization approach which sorts continuous attributes only once, can avoid this unscalability." They then claim: "Also, from our experience, because of the relative randomness among the attribute list, on an average, any given processor will not send more than 5-10% extra elements than N/p . Hence, such scenarios are rare and do not impact runtime scalability."

Next they claim: "Even in the worst case, our formulation still ensures memory scalability by dividing the updates being sent into blocks of N/p , hence requiring only $O(N/p)$ memory for the hash buffers."

Consider that claim: If the communication has a maximum size of N/p and at each step 1 processor has any extra to send, then a second set of all-to-all communications must be performed. This is a likely event and this also effects the scalability of the algorithm.

6.5.3 Perform-Split-I:

"For a continuous splitting attribute, each processor does this trivially by just updating the pointers into the list to demarcate the records belonging to each of the nodes at the next level" [8]. This is a second problem with ScalParC, the process of performing split I is not a trivial update of pointers. This trivial update method leads to problems when a second split is performed on the same attribute.

The update stores only the relationship of a node to its parent, so when a second split is performed on the same attribute the node table cannot store the additional information needed (from further up the tree) to correctly perform the split.

6.6 Summary

Here, we argue that ScalParC does not produce correct decisions in parallel trees under all circumstances. Our experiments were run on the ASCI Red [50] parallel computer using

7 Summary

ScalParC code modified from the original to run on this machine. We also added the ability to print out trees and use information gain to determine the best attribute to use in splitting examples at a node. We find that maintaining the parallel hash table correctly will require additional communication. This will negatively impact the scalability. In effect, we conjecture it will be more like that of Sprint [7].

7 Summary

This report contains the summary of our results in the development of an intelligent aid for visualization of large-scale data sets. Highlights from this work are as follows.

- We have modified the MUSTAFA visualization tool to enable visualization tool users to provide saliency labels to regions of data. This allows training data sets to be easily created.
- We have taken an Exodus format data set (several hundred thousand labeled data samples) of a can being crushed, labeled regions of interest in MUSTAFA and trained a set of rules from a C4.5 decision tree. The rules have been utilized to successfully label regions of interest from untrained time steps during the crushing of the can.
- A scalable, distributed approach to learning from very large data sets has been implemented on the ASCI Red machine.
- A method of resolving conflicts among rule sets learned on disjoint sets of data has been developed. The conflict resolution process allows a single rule model to be built from a training data set that may (greatly) exceed the memory/swap capacity of a given machine.
- Several approaches to utilizing voting with rules created on disjoint data sets have been explored. Preliminary results show that accuracy remains high as the number of partitions increases. An advantage of the proposed voting procedure over building a meta-learner is that the meta-learning stage is avoided, N classifiers do not need to be tested directly, and only a few rules will be fireable per example.

There are several directions for this research to proceed towards enabling the easy building of personal avatars for use in large-scale data visualization. The first is a careful evaluation of the accuracy of learning in parallel with disjoint data sets utilizing (1) conflict resolution and (2) rule voting. The conflict resolution approach provides a single model which is normally more interpretable than other approaches. However, preliminary work suggests that data may need to be presented to the learning algorithms in a different way or specialization may be required as the partitions become larger to ensure no loss in accuracy. The rule voting approach seems promising from an accuracy standpoint, but needs to be fully evaluated. It suffers from a potential overabundance of rules, which may slow down the process of labeling unseen data sets.

7 Summary

Rule voting and conflict resolution may also utilize multiple different classifiers on each processor which may enable higher accuracy as seen via other combination of classifier approaches. A similar accuracy effect may be seen by distributing the data to be learned with overlaps along the lines of what is done with bagging [29].

The implementation of parallel rule testing and specialization needs refinement. Whether all rules get broadcast to all processors or whether a systolic process of rule transmission and testing should be followed needs to be evaluated. Specialization must be implemented in the parallel environment. It requires that a mechanism for mixing in newly created rules be effectively used.

It is also possible that subsets of very large data sets may be utilized in applying learning algorithms [38]. This is possibly a way to save learning time while maintaining accuracy if effective subsampling can be developed. However, the conditions under which subsampling may be effective are not clear. The idea of subsampling for learning from very large training sets merits further investigation.

MUSTAFA needs to be enhanced in two directions. The first is to better enable the capture of labeled training data. The drawing of bounding boxes is desirable. Unobtrusive measures of region salience would greatly ease the burden on the system user and have been discussed (such as time spent browsing a region or number of operations performed on a region of data).

Second, more effective methods of displaying regions labeled with high salience are desirable. Currently, regions of interest are listed in a web interface. It might be useful to allow users to click on a button and be taken to a potentially interesting view. We have discussed providing colors or some other marker around areas of high salience. The danger is that a data visualizer might never look near these areas. Another approach is to allow a user to ask for views of all regions with specific salience levels. An issue is that we may not give them exactly the salient view and some searching may still be required.

Tests of different approaches to viewing and labeling large data sets with salience are necessary to determine the best approaches and important differences between approaches. This also leads to the need to gather and label some very large visualization data sets.

The above directions define at a broad brush a course of future research and development for the development of intelligent visualization tools.

Acknowledgments

This research was partially supported by the United States Department of Energy through the Sandia National Laboratories LDRD program, contract number DE-AC04-76DO00789. Some of this material is contained in the following papers:

Hall, L.O., Chawla, N. and Bowyer, K.W., Decision Tree Learning on Very Large Data Sets, International Conference on Systems, Man and Cybernetics, Oct, 2579-2584, 1998.

Hall L.O., Chawla N., Bowyer K.W. and Kegelmeyer W.P.K., Learning Rules from Distributed Data, Proceedings of the Workshop on Large-Scale Parallel KDD Systems, Technical Report 99-8, Rensselaer Polytechnic Institute, CS Dept., Zaki M.J. and Ching-Tien H.,

1999, KDD99, 77-83.

8 Bibliography

- [1] T.M. Mitchell. *Machine Learning*. McGraw-Hill, N.Y., 1997.
- [2] Michael W. Glass. MUSTAFA tutorial. Albuquerque, NM 87185.
- [3] Michael W. Glass. MUSTAFA user's guide. Albuquerque, NM 87185.
- [4] Larry A. Schoof Victor R. Yarberry. *EXODUS II: A Finite Element Data Model*. Sandia National Labs, Albuquerque, NM 87185 and Livermore, CA 94550, 1994.
- [5] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1992. San Mateo, CA.
- [6] AVS/Express. AVS/Express visualizaion tool, developers edition. <http://www.avs.com>.
- [7] John Shafer, Rakesh Agrawal, and Manish Mehta. Sprint: A scalable parallel classifier for data mining. In *Proceedings of the 22nd VLDB Conference*, 1996.
- [8] M. V. Joshi, G. Karypis, and V. Kumar. ScalParC: A new scalable and efficient parallel classification algorithm for mining large datasets. Technical report, University of Minnesota, 1998.
- [9] R. Kufrin. Generating C4.5 production rules in parallel. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 565–570, July 1997.
- [10] F.J. Provost and D.N. Hennessy. Scaling up: Distributed machine learning with cooperation. In *Proceedings of AAAI'96*, pages 74–79, 1996.
- [11] P.K. Chan and S.J. Stolfo. Scaling learning by meta-learning over disjoint and partially replicated data. In *Proceedings of the Florida Artificial Intelligence Society*, 1996.
- [12] P.K. Chan and S.J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *AAAI'98*, 1998.
- [13] P.D Wasserman. *Neural Computing, Theory and Practice*. Van Nostrand Rheinhold, N.Y., 1989.
- [14] J.R. Quinlan. Generating production rules from decision trees. In *Proceedings of IJCAI-87*, pages 304–307, 1987.
- [15] J. Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4(2):227–243, 1989.

8 Bibliography

- [16] J.R. Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27:227–248, 1987.
- [17] J.R. Quinlan. Improved use of continuous attributes in C4.5. *Journal of Artificial Intelligence Research*, 4:77–90, 1996.
- [18] W.W. Cohen. Fast effective rule induction. In *Proceedings of the 12th Conference on Machine Learning*, 1995.
- [19] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5(3), 1990.
- [20] J.R. Quinlan and R.M. Cameron-Jones. FOIL: A midterm report. In Pavel B. Brazdil, editor, *Machine Learning: ECML-93*, Springer-Verlag Lecture Notes in Computer Science #667, 1993.
- [21] 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.
- [22] S. Clearwater and F. Provost. RL4: a tool for knowledge-based induction. In *Proceedings of the 2nd International IEEE Conference on Tools for AI*, pages 24–30, 1990.
- [23] U. Fayyad and et. al. Skicat: A machine learning system for automated cataloging of large scale sky surveys. In *Proceedings of the tenth International Conference on Machine Learning*. Morgan Kaufmann, 1993.
- [24] J. Cheng, U.M. Fayyad, K.B. Irani, and Z. Qian. Applications of machine learning techniques in semiconductor manufacturing. In *Proceedings of the SPIE Conference on Applications of Artificial Intelligence VIII*, pages 956–965, Orlando, Fl., 1990.
- [25] W. Lee, S.J. Stolfo, and K. Mok. A data mining framework for building intrusion detection models. In *Proc. 1999 IEEE Symposium on Security and Privacy*, 1999.
- [26] S.J. Stolfo, A.L. Prodromidis, S. Tselepis, W. Lee, D. Fan, and P.K. Chan. JAM: Java agents for meta-learning over distributed databases. In *Proc. KDD-97*, 1997.
- [27] A. L. Prodromidis, S. J. Stolfo, and P. K. Chan. Pruning classifiers in a distributed meta-learning system. In *Proc. of First National Conference on New Information Technologies*, pages 151–160, 1998.
- [28] P. K. Chan and S. J. Stolfo. Toward scalable learning with non-uniform class and cost distributions: A case study in credit card fraud detection. In *Proc. KDD-98*, 1998.
- [29] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.
- [30] L. Breiman. Pasting bites together for prediction in large data sets. *Machine Learning*, 36(1,2), 1999.

8 Bibliography

- [31] E. Bauer and R. Kohavi. An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36(1,2), 1999.
- [32] P. Domingos. Metacost: A general method for making classifiers cost-sensitive. In S. Chaudhuri and D. Madigan, editors, *Proceedings of the Fifth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 155–164. Association for Computing Machinery, 1999.
- [33] C. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1,2), 1999.
- [34] R.E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197–227, 1990.
- [35] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth National Conference*, pages 148–156, 1996.
- [36] W.D. Shannon and D. Banks. A distance metric for classification trees. In *Sixth International Workshop on Artificial Intelligence and Statistics*, Fort Lauderdale, Fl., 1997.
- [37] W.D. Shannon and D. Banks. Combining classification trees using mle. *Statistics in Medicine*, 1999.
- [38] F. Provost, D. Jensen, and T. Oates. Efficient progressive sampling. In S. Chaudhuri and D. Madigan, editors, *Proceedings of the Fifth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pages 23–32, N.Y., N.Y., 1999. ACM.
- [39] J. Ke. Fast accurate fuzzy clustering through reduced precision. M.S. thesis, University of South Florida, Tampa, Fl. 33620, 1999.
- [40] Hall L.O., Chawla N., Bowyer K.W., and Kegelmeyer W.P. Learning rules from distributed data. In Zaki M.J. and Ching-Tien H., editors, *Proceedings of the Workshop on Large-Scale Parallel KDD Systems*, pages 77–83. KDD99, Technical Report 99-8, Rensselaer Polytechnic Institute, CS Dept., 1999.
- [41] S.M. Weiss, R.S. Galen, and P.V. Tadepalli. Maximizing the predictive value of production rules. *Artificial Intelligence*, 45:47–71, 1990.
- [42] 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.
- [43] R. Segal and O. Etzioni. Learning decision lists using homogeneous rules. In *Proceedings of the AAAI-94*, pages 619–624, 1994.
- [44] L.O. Hall, N. Chawla, and K.W. Bowyer. Decision tree learning on very large data sets. In *International Conference on Systems, Man and Cybernetics*, pages 2579–2584, Oct 1998.

8 Bibliography

- [45] 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.
- [46] C. Merz. Using correspondence analysis to combine classifiers. *Machine Learning*, 36(1,2), 1999.
- [47] C.J. Merz and P.M. Murphy. *UCI Repository of Machine Learning Databases*. Univ. of CA., Dept. of CIS, Irvine, CA. <http://www.ics.uci.edu/~mlearn/MLRepository.html>.
- [48] R.A. Fisher. The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7, 1936.
- [49] 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.
- [50] Sandia National Labs, <http://www.sandia.gov/ASCI/Red/UserGuide.htm>. *ASCI Red Users Manual*, 1997.

Distribution

- 1 Kevin Bowyer
University of South Florida
Department of Computer Sciences and Engineering
4202 East Fowler Avenue, ENB 118
Tampa, FL, 33620
- 1 Nitesh Chawla
University of South Florida
Department of Computer Sciences and Engineering
4202 East Fowler Avenue, ENB 118
Tampa, FL, 33620
- 1 Larry Hall
University of South Florida
Department of Computer Sciences and Engineering
4202 East Fowler Avenue, ENB 118
Tampa, FL, 33620
- 1 Thomas Moore, Jr.
University of South Florida
Department of Computer Sciences and Engineering
4202 East Fowler Avenue, ENB 118
Tampa, FL, 33620
- 1 Cathy Smith
University of South Florida
Department of Computer Sciences and Engineering
4202 East Fowler Avenue, ENB 118
Tampa, FL, 33620

- 1 MS 0561 Donna Chavez, for LDRD Office, 01401
- 1 9004 John Vitko, 8100
- 1 9214 L.M. Napolitano, 8130
- 10 9214 W.P. Kegelmeyer, Jr., 8130
- 3 9018 Central Technical Files, 8940-2
- 1 0899 Technical Library, 4916
- 1 9021 Technical Communications Department, 8528/
Technical Library, MS 0899, 4916
- 1 9021 Technical Communications Department, 8528
For DOE/OSTI