

Inductive Learning on Partitioned Data

A Thesis Presented

by

Qijun Chen

to

The Faculty of the Graduate Colledge

of

The University of Vermont

In Partial Fulfillment of the Requirements
for the Degree of Master of Science
Specializing in Computer Science

February 2004

Accepted by the Faculty of the Graduate College, The University of Vermont, in partial fulfillment of the requirements for the degree of Master of Science, specializing in Computer Science.

Thesis Examination Committee:

_____ Advisor

Xindong Wu, Ph.D.

_____ Co-Advisor

Xinquan Zhu, Ph.D.

_____ Chairperson

Adel W. Sadek, Ph.D.

_____ Vice President for Research and
Dean of the Graduate College

Frances E Carr, Ph.D.

Date: November 13, 2003

ABSTRACT

Inductive Learning is a typical learning task in machine learning. Given a data set, inductive learning aims to discover patterns in the data and form concepts that describe the data. Research in inductive learning has sustained for decades. However, much of the existing work focuses on a relatively small amount of data, which will be infeasible in large, realistic situations.

With the rapid advancement of information technology, scalability has become a necessity for learning algorithms to deal with large, real-world data repositories. This thesis aims to design some scalable inductive learning algorithms. Scalability is defined as the ability to process large data sets or handle data sets that are distributed at different sites. In our work, scalability is accomplished through a data reduction technique, which partitions a large data set into subsets, applies the learning algorithm on each subset sequentially or concurrently, and then integrates the learned results.

Five strategies to achieve scalability (Rule-Example Conversion, Rule Weighting, Iteration, Good Rule Selection, and Data Dependent Rule Selection) have been identified and their corresponding scalable schemes have been designed and developed. A substantial number of experiments have been performed to evaluate these schemes. Experimental results demonstrate that through data reduction some of our schemes can effectively generate accurate classifiers from inaccurate classifiers generated from data subsets. Furthermore, our schemes require significantly less training time than that of generating a global classifier.

Among the five investigated strategies, Iteration and Data Dependent Rule Selection are the two most effective strategies in respect to the classification accuracy of the generated classifiers and the variety of the data sets that can be dealt with. These two strategies, combined with a Voting strategy, can generate schemes, which outperform Voting consistently.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis committee members Dr. Xindong Wu, Dr. Xinquan Zhu, and Dr. Adel W. Sadek for their guidance towards my thesis. Specifically I am grateful to Dr. Wu, my thesis advisor, for his enlightening suggestions and guidance. I am also grateful to Dr. Zhu, who gave me continual support in fulfilling this thesis work. I would like to thank Dr. Sadek, who kindly helped me in the final defense.

This study is supported by EMCF (Engineering and Mathematics Computer Facility, College of Engineering, University of Vermont), a NASA EPSCoR grant, and a DOD grant (DAAD19-02-1-0178).

TABLE OF CONTENTS

ACKNOWLEDGEMENTS	ii
LIST OF TABLES	vi
LIST OF FIGURES	viii
Chapter 1 INTRODUCTION.....	1
1.1 Problem Description.....	1
1.2 Summary of Results and Contributions	3
1.3 Organization of Thesis.....	5
Chapter 2 INDUCTIVE LEARNING AND RELATED WORK.....	6
2.1 Inductive Learning.....	6
2.1.1 Tree Based Induction.....	7
2.1.2 Rule Based Induction.....	9
2.2 Data Reduction	11
2.3 Sampling	11
2.4 Processing Subsets Sequentially.....	13
2.4.1 Windowing	15
2.4.2 Integrative Windowing.....	16
2.4.3 Multiple Layer Induction.....	18

2.5	Process Subsets Concurrently	19
2.5.1	Voting	19
2.5.2	Meta-Learning.....	21
Chapter 3	INDUCTIVE LEARNING ON PARTITIONED DATA	26
3.1	Introduction	26
3.2	Rule-Example Conversion	26
3.3	Rule Weighting	29
3.4	Iteration	31
3.4.1	Iteration.....	31
3.4.2	Iteration & Voting	34
3.5	Good Rule Selection	36
3.5.1	Good Rule Selection	37
3.5.2	Good Rule Selection & Voting	41
3.6	Data Dependent Rule Selection	43
3.7	Summary	45
Chapter 4	EXPERIMENTS	46
4.1	Test Methodologies	46
4.2	Rule-Example Conversion	49
4.3	Rule Weighting	50

4.4	Iteration.....	51
4.5	Iteration & Voting.....	52
4.6	Good Rule Selection.....	54
4.7	Good Rule Selection & Voting.....	55
4.8	Data Dependent Rule Selection.....	57
4.9	Time Complexity Analysis.....	58
4.10	Experiment Summary.....	59
Chapter 5	SUMMARY AND FUTURE WORK.....	62
REFERENCES	65

LIST OF TABLES

TABLE 1 SOLVENT AND INSOLVENT CUSTOMERS.....	6
TABLE 2 A PARTITIONED DATA SET ON DISEASE A.....	26
TABLE 3 EXPANDED DATA SET THROUGH RULE-EXAMPLE CONVERSION	29
TABLE 4 EXPANDED P_1 ’ WITH MISCLASSIFIED EXAMPLES FROM P_2	34
TABLE 5 EXPANDED SUBSETS AFTER FORWARDING MISCLASSIFIED EXAMPLES	36
TABLE 6 CHARACTERISTICS OF 7 SCALABLE SCHEMES	45
TABLE 7 DATA SET PROPERTIES	47
TABLE 8 SCHEME ACRONYMS	48
TABLE 9 RESULTS (RULE-EXAMPLE CONVERSION)	50
TABLE 10 RESULTS (RULE WEIGHTING, HCV)	51
TABLE 11 RESULTS (ITERATION, HCV)	52
TABLE 12 RESULTS (ITERATION, C4.5RULES).....	52
TABLE 13 RESULTS (ITERATION, NOISY DATA SET)	52
TABLE 14 RESULTS (ITERATION &VOTING)	53
TABLE 15 RESULTS (ITERATION &VOTING, LARGE SET).....	53
TABLE 16 RESULTS (ITERATION & VOTING, NOISY DATA SET)	54
TABLE 17 RESULTS (GOOD RULE SELECTION).....	55
TABLE 18 RESULTS (GOOD RULE SELECTION & VOTING)	56
TABLE 19 RESULTS (GOOD RULE SELECTION & VOTING, LARGE SETS)	56
TABLE 20 RESULTS (GOOD RULE SELECTION & VOTING, WITH DIFFERENT SIZES OF EVALUATION SET) ...	57

TABLE 21 RESULTS (DATA DEPENDENT RULE SELECTION)	58
TABLE 22 TRAINING TIME OF GRSV, DDRS, IV AND C4.5 RULES (UNIT SECONDS)	59
TABLE 23 RESULTS (7 SCHEMES COMPARISON ON CAR DATA SET)	60
TABLE 24 RESULTS (5 SCHEMES COMPARISON ON SPLICE DATA SET)	60
TABLE 25 RESULTS (5 SCHEMES COMPARISON ON IBM DATA SET).....	61

LIST OF FIGURES

FIGURE 1 DECISION TREE GENERATED BY C4.5 ON CUSTOMER DATA SET	9
FIGURE 2 RULES GENERATED BY HCV ON CUSTOMER DATA SET	10
FIGURE 3 MODEL-GUIDED INSTANCE SELECTION.....	14
FIGURE 4 INCREMENTAL BATCH LEARNING.....	14
FIGURE 5 RULE-EXAMPLE CONVERSION	28
FIGURE 6 RULE WEIGHTING	30
FIGURE 7 ITERATION.....	33
FIGURE 8 ITERATION & VOTING.....	35
FIGURE 9 RULES INDUCED BY C4.5RULES ON MONK1 DATA SET	39
FIGURE 10 GOOD RULE SELECTION	40
FIGURE 11 GOOD RULE SELECTION & VOTING	42
FIGURE 12 DATA DEPENDENT RULE SELECTION.....	44

Chapter 1 INTRODUCTION

1.1 Problem Description

For a human being, learning is the action of acquiring knowledge or skills as a result of study, experience, or teaching. Such knowledge will be useful when a similar, or the same situation happens again. Researchers in the field of machine learning (Carbonell, 1989) attempt to endow the computers with such intelligence. This thesis research is based on a special category of machine learning, called inductive learning (Michalsky, 1983). Inductive learning can be defined as the task of identifying regularities in some given set of training examples (Chan 1996). Such examples have a feature vector and a class label (concept). Inductive learning is the procedure to form a concept description for each class (concept). These generalized descriptions can be used to discriminate between different concepts and predict the label of some unseen instance.

Research in inductive learning has made substantial progress over the years and a number of algorithms have been developed. AQ (Michalski et al., 1986), C4.5 (Quinlan, 1993), CART (Breiman et al., 1984), CN2 (Clark & Niblett, 1989), HCV (Wu, 1995) are the most popular ones. Usually two criteria are used to evaluate the performance of one specific algorithm: accuracy and efficiency. Accuracy means how well the learned concepts match the reality. It is hoped the learned rules can correctly predict the concept labels of the unseen instances. Efficiency means how fast these concept patterns can be

learned. With the advancement of information technology, enormous amount of data can be easily generated. So it becomes increasingly important that a learning system can process examples in an efficient way.

Improving the accuracy and efficiency of existing learning algorithms has been a central issue for most inductive learning research efforts. However, much of the existing research concentrates on problems with relatively small amount of data. Meanwhile learning systems are confronted with vast amounts of information nowadays. Common databases of customers, operations, scientific data, and other sorts of data are growing rapidly (Fayyad, Piatetsky-Shapiro, and Smyth 1996). It would be desirable that useful knowledge can be mined from these ever-growing sets. The question is can we apply our learning algorithms directly to these real-world sets. Unfortunately, few of the existing algorithms are designed to handle large or distributed sets. Furthermore, many of them require all the training data to be resident in main memory before being processed. Even with the fast development of the hardware technology, such a requirement is still untenable in most real-world cases. It is found that both manual data mining and the direct application of today's mining techniques can be problematic when data sets exceed 100 megabytes (Provost and Kolluri 1997; Huber 1997). In certain cases, data is inherently distributed and can only be localized, it would be infeasible to bring all the data together and compute one primary "global" classifier.

The above observations motivated this thesis research in scalable learning algorithms. Here scalability is defined as the ability to process large example set or handle data sets

that are distributed at different sites. With the advent of the age of information explosion, scalability has become a necessity for learning algorithms to be used out of the research lab.

As many of the existing learning algorithms have a solid foundation, it would be inadvisable to design the scalable learning schemes from the scratch. So rather than changing the underlying principles of those algorithms, our approach seeks to solve the scalability problem through data reduction, which partitions the data set into subsets, applies a learning algorithm on each subset sequentially or concurrently, and then integrates the learning results. Each subset is sized to fit into the memory so that it can be processed efficiently. It is anticipated that the speed improvement would be more substantial when the computation complexity is more than linear to the data set size. As in the distributed environment, each subset is disjointed with each other. Accuracy will suffer as a result of data partitioning (Chan 1996). Our approach seeks to minimize the impact of information loss due to partitioning.

1.2 Summary of Results and Contributions

This thesis aims to address the problem of efficiently and accurately analyzing massive amounts of data using inductive learning algorithms. Here we briefly summarize the contributions:

- Five strategies to achieve scalability (Rule-Example Conversion, Rule Weighting, Iteration, Good Rule Selection, Data Dependent Rule Selection) have been identified and the corresponding schemes have been designed and developed. A

substantial number of evaluation based on different inductive learning algorithms have been performed.

- Our study shows that classifiers trained individually from random subsets of a large data set are not as accurate as integrating a collection of separately learned classifiers.
- All the schemes evaluated here cannot maintain the accuracy of the global classifier (trained from the global set) as the number of subsets increases.
- Generally, the accuracy of the final classifier drops as the number of subsets increases. Such accuracy degradation can be ascribed to information loss caused by partitioning.
- Among all the schemes we designed and evaluated, Iteration&Voting and Good Rule Selection–Data Dependent Rule Selection are the two most promising schemes, which can outperform Voting consistently. So incorporating knowledge from other learning processes can improve local learning.
- Iteration&Voting outperforms both Iteration and Voting. So integrating these two strategies can help generate a stronger classifier. Furthermore, Iteration & Voting can overcome the noise-intolerant weakness of the Iteration strategy.
- Compared with Good Rule Selection, Data Dependent Rule Selection is more capable when the number of subsets is large and each subset is small.

- Iteration & Voting, Good Rule Selection&Voting, and Data Dependent Rule Selection schemes can save significant training time when the global data set is large and the underlying learning algorithms' time complexity is more than linear of the number of the examples

1.3 Organization of Thesis

This thesis is organized as follows. In Chapter 2, we review the concepts of inductive learning. Following that, three generalized categories of data reduction techniques to achieve scalability are introduced. We elaborate some common approaches in each category.

In Chapter 3, we describe the different schemes we designed and investigated. To evaluate the different schemes, we did a substantial amount of experiments. The detailed experimental results are shown in Chapter 4. We also provide our findings for each experiment after the results.

We conclude, in Chapter 5, by discussing the contributions and research directions of this work.

Chapter 2 INDUCTIVE LEARNING AND RELATED WORK

2.1 Inductive Learning

Inductive learning can be described as a process that systematically produce general descriptions from the specific knowledge provided by domain examples. In inductive learning, knowledge is compiled by generalizing patterns in factual experience. For example, Table 1 displays a set of solvent and insolvent customer records. Through analyzing features of many solvent and insolvent customers, you may learn to recognize, on the basis of the feature values, which customers appear to be solvent and which are not. This type of task would be referred to as inductive learning.

Age	Income	Education	Solvability
72	18	L	N
14	85	L	N
98	18	L	N
16	76	H	Y
65	98	H	Y
32	82	H	Y
28	75	H	Y
19	62	H	Y
32	73	H	N
65	36	H	N
56	66	H	N

Table 1 Solvent and Insolvent Customers

The above table may tell you that

IF education = H AND (Income > 75 OR age < 30) THEN solvability = Y
ELSE solvability = N

The learned concepts can then be used to predict the concept label of the future instances (e.g. age=50, income = 70, education = L).

Inductive learning can be supervised or unsupervised. In supervised inductive learning, the class labels of training examples are supplied and learned concepts describe these class labels (or learning from examples (Dietterich & Michalski, 1983)). However, in unsupervised learning (or learning from observations (Michalski & Stepp, 1983)), the class labels are not known. The learning algorithm induces clusters, which can later be identified as individual concepts. CLUSTER/2 (Michalski & Stepp, 1983), COBWEB (Fisher, 1987), and AUTOCLASS (Chessemann et al., 1998) are the typical conceptual clustering algorithms.

Inductive learning algorithms are utilized to generate the concept descriptions (classifier) automatically. Decision trees and rules are two common representations used for the generated classifiers.

2.1.1 Tree Based Induction

Decision trees are used in ID3, CART, and C4.5 where each concept is represented as a conjunction of terms on a path from the root. Tree structure is declarative, comprehensible, and thus commonly used.

ID3

ID3 is a top-down algorithm developed by Quinlan out of the Concept Learning System (CLS) by Hunt (Hunt, et al. 66). CLS is a learning mechanism that accepts a set of training pairs and constructs a representation in the form of a decision tree, which is equivalent to a disjunctive rule. The decision tree is structured so that each leaf node has a target output associated with it.

Quinlan modified the CLS algorithm in two ways. First, he added a process known as windowing (Quinlan, 1983). This was designed to enable the algorithm to cope with very large training sets. Second, Quinlan devised an information theoretic heuristic (the entropy or information gain measure) that decided how to split the inputs at each stage of the tree-growing process, thus enabling smaller and therefore more efficient decision trees to be constructed.

C4.5

C4.5 is evolved from ID3. Several new features are added to extend the capacities of the original ID3 algorithm.

- Dealing with continuous attribute values.
- Grouping nominal (discrete) values of a single attribute together to support more complex tests
- Incremental learning
- Noise Handling
- Post-pruning after induction of trees
- Dealing with incomplete information (missing attribute values).

Figure 1 shows the output of C4.5 on the customer data set (Table 1).

```

Read 11 cases (3 attributes) from Customer.data

Decision Tree:

education = L: N (3.0)

education = H:
| income <= 73 : N (4.0/1.0)
| income > 73 : Y (4.0)

Tree saved

Evaluation on training data (11 items):

      Before Pruning      After Pruning
      -----
Size  Errors  Size  Errors  Estimate
  5   1(9.1%)  5   1(9.1%)  (40.6%) <<

```

Figure 1 Decision Tree Generated by C4.5 on Customer Data Set

2.1.2 Rule Based Induction

Rules are used in AQ, C4.5Rules, CN2, HCV where if-then expressions are used to describe the concept patterns.

HCV

HCV is an extension matrix based rule induction algorithm, which is Heuristic, Attribute-based. It divides the positive examples into intersecting groups and then uses HFL

heuristics to find a conjunctive formula, which covers each intersecting group. Figure 2 shows the output of HCV on the customer data set (Table 1).

```
Rules for the 'Y' class (Covering 5 examples):  
The 1st conjunctive rule:  
  [ education = { H } ]  
  --> the 'Y' class  
  (Positive examples covered: 5)  
  
Rules for the 'N' class (Covering 3 examples):  
The 2nd conjunctive rule:  
  [ education = { L } ]  
  --> the 'N' class  
  (Positive examples covered: 3)  
  
The total number of conjunctive rules is: 2  
The default class is: 'N' (Examples in class: 6)  
Time taken for induction (seconds): 0.0 (real), 0.0 (user), 0.0 (system)
```

Figure 2 Rules Generated by HCV on Customer Data Set

C4.5Rules

C4.5Rules is a component algorithm of the C4.5 Package. It generates rules through decompiling decision trees into production rules. It contains three basic steps [Corlett 83, Quinlan 87a]:

1. Traverse a decision tree to obtain a number of conjunctive rules. Each path from the root to a leaf in the tree corresponds to a conjunctive rule with the leaf as its conclusion.
2. Check each condition in each conjunctive rule to see if it can be dropped without more misclassification than expected on the original training examples or new test examples.

3. If some conjunctive rules are the same after Step 2, then keep only one of them.

2.2 Data Reduction

Data reduction techniques are used to handle large or distributed sets in inductive learning. The data reduction approach introduced here involves partitioning the initial data set up into subsets, learning from one or more of the subsets, and possibly combining the results from each separate learning process. Data reduction is useful to avoid thrashing of memory management systems. Also, if a learning algorithm's time complexity is more than linear in the number of examples, processing small, fixed size samples sequentially can reduce the complexity to linear (with the constant term dependent on the size of the subsets).

Data reduction techniques can be generalized into 4 categories: sampling, feature selection, processing subsets sequentially, and processing subsets concurrently (Provost, 1999). As we only build our schemes on the principles of sampling, sequential processing of subsets and concurrent processing of subsets, feature selection is omitted in the introduction.

2.3 Sampling

The most common approach for coping with the infeasibility of learning from very large data sets is to select a single, small sample from the large data set. Random sampling, stratified sampling, and duplicate compaction are the three representative sampling

methods. Random sampling means to select a subset of examples randomly from the whole training set. Because of the randomness, it is hard to assess the performance of the learning algorithm through the learned classifier. Duplicate removal tries to compact the database through removing duplicates. The computational effort is proportional to the degree of completeness desired. Stratified sampling is the commonly used and most scientific method. It tries to sample a subset, which maintains the same class distribution of the original training set. So one scan of the database to get the statistics of the original database is needed before sampling.

People often doubt whether sampling would be an effective solution for the scaling problem. Catlett (Catlett, 1991a) studied a variety of sampling methods. His work showed that learning from subsets of data through sampling decreases accuracy. Despite the advantages of certain sampling strategies (speedups and improving the accuracy of the classifier through random sampling in noise free domains), Catlett concluded that they are not a solution to the general problem of scaling up to very large data sets.

Some people may question Catlett's conclusion, as what he claimed to be a large set in his experiment has only fewer than 10,000 examples. In a more recent study, Harris-Jones and Hains (1997) analyze the relationship between data set size and accuracy for two large business sets (up to 300,000 instances). They found while some algorithms level off quite early, in some cases algorithms (such as C4.5) did show accuracy increases across the entire range of data sizes. However, the improvement in upper level is quite small. So it is uncertain whether it is worthwhile to trade the benefit of small accuracy

improvement with the associated cost. Thus in some situations, sampling may be a simple solution for the scaling problem.

If sampling is a solution, then how much data must be sampled to generate a classifier with comparative performance with the one generated from the whole data set ? Perhaps no one can answer this question completely, as the result is data set dependent. It depends on the distribution and redundancy of the original set.

Generally, sampling alone is not the best solution for scaling problem as compared with the approaches in the other two categories. But sampling can be utilized by other approaches to achieve accuracy and efficiency.

2.4 Processing Subsets Sequentially

The general solution for the scaling problem is to learn from multiple subsets and combine the results. We will first consider those approaches where subsets are processed sequentially. When multiple subsets are being processed sequentially, it is possible for approaches to take advantage of knowledge learned in one iteration to guide learning in the next iteration. Figure 3 and Figure 4 show two approaches to this sequential multi-subset learning. In model-guided instance selection, shown in Figure 3, class description C_i is used in the selection of instances in next subset P_{i+1} . In incremental batch learning, shown in Figure 4, class description C_i is taken as input to the learner and used in building C_{i+1} .

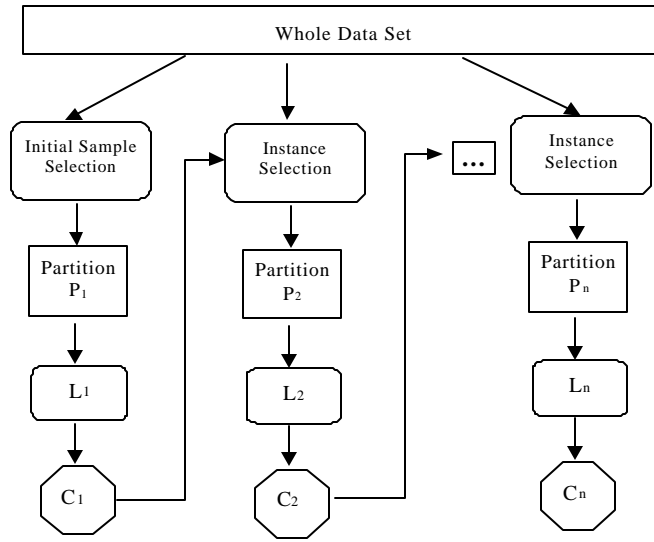


Figure 3 Model-guided Instance Selection

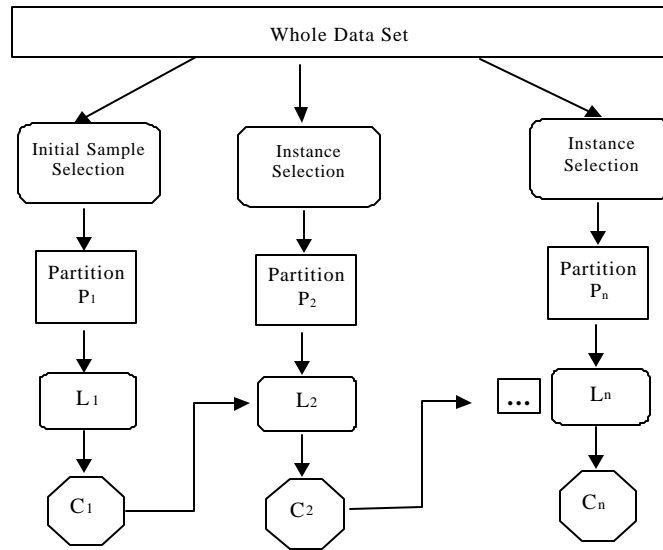


Figure 4 Incremental Batch Learning

2.4.1 Windowing

Windowing (Quinlan,1983) is the typical model-guided instance selection method. It is a sub-sampling technique proposed by Quinlan in the ID3 decision tree learning algorithm, whose goal is to reduce the complexity of a learning problem by identifying an appropriate subset of the original data, from which a classifier of sufficient quality can be learned. For this purpose, it maintains a subset of the available data, the so-called window, which is used as the training set for the learning algorithm. The window is initialized with a small random sample of the available data, and the learning algorithm induces a classifier from this sample. This classifier is then tested on the remaining examples. If the quality of the learned classifier is not sufficient, the window is adjusted, usually by adding more examples from the training data, and a new classifier is learned. This process is repeated until a classifier of sufficient quality has been found.

As shown in *procedure Windowing*, in order to keep the size of the training set small, the parameter, *MaxIncSize*, controls the maximum number of examples that can be added to the training set in one iteration. If this number is reached, no further examples are tested and the next theory is learned from the new training set. To make sure that all examples are tested in the first few iterations, the examples that have already been tested (*OldTest*) are appended to the examples that have not yet been used, so that testing will start with new examples in the next iteration.

procedure Windowing(Examples,InitSize,MaxIncSize)
Window = RandomSample(Examples,InitSize)

```

Test = Examples \ Window
repeat
    classifier = Induce(Window)
    NewWindow = F;
    OldTest = F;
    for Example Î Test
        Test = Test \ Example
        if Classify(Theory,Example)  $\neq$  Class(Example)
            NewWindow = NewWindow  $\dot{\cup}$  Example
        else
            OldTest = OldTest  $\dot{\cup}$  Example
        if |NewWindow| = MaxIncSize
            exit for
        Test = Append(Test,OldTest)
        Window = Window  $\dot{\cup}$  NewWindow
    until NewWindow = F;
return(Theory)

```

2.4.2 Integrative Windowing

To avoid discovering good rules again and again when using windowing with a rule learning algorithm, Fürnkranz developed a new version of windowing called integrative windowing (Fürnkranz, J.,1998), which tries to exploit the fact that regions of the example space that are already covered by good rules need not be further considered in subsequent iterations.

The algorithm shown below starts just like basic windowing: it selects a random subset of the examples, learns a classifier from these examples, and tests it on the remaining

examples. However, contrary to basic windowing, it does not merely add incorrectly classified examples to the window for the next iteration, but also removes examples from the window if they are covered by consistent rules. A rule is considered consistent when it does not cover a negative example during the testing phase. Note that this does not necessarily mean that the rule is consistent with all examples in the training set because it may contradict an example that has not yet been tested at the point where $MaxIncSize$ misclassified examples have been found. Thus, apparently consistent rules have to be remembered and tested again in the next iteration. However, testing is much cheaper than learning, so removing the examples that are covered by these rules from the window should keep the window size small and thus decrease the learning time.

procedure IntegrativeWindowing (Examples, InitSize, MaxIncSize)

Window = RandomSample (Examples, InitSize)

Test = Examples \ Window

OldRules = F ;

repeat

NewRules = Induce (Window)

Theory = NewRules \hat{E} OldRules

NewWindow = F ;

OldTest = F ;

for Example \hat{I} Test

Test = Test \ Example

if Classify (Theory, Example) \neq Class (Example)

NewWindow = NewWindow \hat{E} Example

else

OldTest = OldTest \hat{E} Example

```

    If /NewWindow/ = MaxIncSize
        exit for
    Test = Append (Test, OldTest)
    Window = Window  $\hat{\cup}$  NewWindow  $\hat{\cup}$  Cover (OldRules)
    OldRules = F;
    for Rule  $\hat{\in}$  Theory
        if Consistent (Rule, NewWindow)
            OldRules = OldRules  $\hat{\cup}$  Rule
            Window = Window  $\setminus$  Cover (Rule)
    until NewWindow = F;
    return(Theory)

```

2.4.3 Multiple Layer Induction

Multiple Layer Induction (Wu and Lo, 1998) falls into the category: incremental batch learning where C_i will be the input to induce C_{i+1} . It consists of three operations: data partitioning, generalization and reduction. Data partitioning breaks the initial data set into a number of data subsets of approximately equal size in a random shuffled way. Generalization can either learn a set of rules from a subset of examples, or refine a previous set of rules. The latter is achieved through a redescription operation called reduction: from a set of examples and a set of rules, a new set of examples is derived to describe the behavior of the rule set. New rules are extracted from these behavioral examples, and these rules can be seen as meta-rules, as they control previous rules in order to improve their predictive accuracy. *Procedure MLI* shows the algorithm for Multiple Layer Induction.

```

Procedure MLI ( NumOfPar)
DataPartition(NumOfPar) ;
TempRuleSet = f ;
TempDataSet = f ;
For (j=1; j < NumOfPar; j++)
    RuleSetj = Induce( Partitionj  $\hat{E}$  TempDataSet) ;
    TempDataSet = Reduce(RuleSetj) ;
EndFor
Return Induce(PartitionNumOfPar  $\hat{E}$  TempDataSet) ;

```

2.5 Process Subsets Concurrently

To further increase efficiency, learning can be performed on each subset concurrently. But this approach posts a new question: how to combine the results from different independent learning. Voting and Meta-learning are the two typical solutions for the combination question. We introduce some representative methods in each category.

2.5.1 Voting

Voting is the simplest method to combine multiple evidences into a singular prediction. The first scheme we examine is Voting. That is, based on the predictions of different base classifiers, a final prediction is chosen as the classification with a plurality of votes. A variation of Voting is Weighted Voting. Each classifier is associated with a weight, which is determined by how accurate the classifier performs on a validation set. A validation set is a set of examples randomly selected from all available data. Since each classifier is trained on only one subset, examples in the other subsets that contribute to

the validation set provide a measure of predictiveness. Each prediction is weighted by the classifier's assigned weight. The weights of each classification are summed and the final prediction is the classification with the most weight.

Though simple, Voting shows surprisingly good performance in some cases. Voting can also easily be implemented in the distributed environment, as each voting member can be treated as a black box. Based on these observations, some of our schemes use the Voting technique.

Littlestone and Warmuth (1989) propose several Weighted Majority algorithms for combining different classifiers. (In their work the classifiers are different prediction algorithms, which are not necessarily learned. The training data are only used for calculating the weights.) These combining algorithms are similar to the Weighted Voting method described above; the main difference is how the weights are obtained. The basic algorithm, called WM, associates each learned classifier with an initial weight. Each example in the training set is then processed by the classifiers. The final prediction for each example is generated as in Weighted Voting. If the final prediction is wrong, the weights of the classifiers whose predictions are incorrect are multiplied by a fixed discount β , where $0 \leq \beta < 1$, that decreases their contribution to final predictions.

A variation of the basic WM algorithm, called WML, does not allow the weights to be discounted beyond a predefined limit. A discount can only occur if the weight is larger than

γ /number of classifiers

times the total weight of all classifiers, where $0 \leq \gamma < 0.5$. Another variation, called WMR, produces randomized responses. The probability of a classification selected as the final prediction is the total weight of that classification divided by the total weight of all classifications; i.e.,

$$P(\text{class}_x) = \text{total weight}(\text{class}_x) / \sum_i \text{total weight}(\text{class}_i)$$

The weights are trained as in the WM algorithm.

2.5.2 Meta-Learning

Chan and Stolfo (Chan 1996) propose the Combiner and Arbiter, which uses a meta-learning approach to combine the results from different learning processes. Meta-learning means learning from the outputs of concept learning systems. Specifically, it is a process to learn from the predictions of base classifiers on common training data. Here base classifiers are generated through applying learning algorithm on subsets of raw training data directly. These base classifiers are programs, which can predict labels given a test datum. In either Combiner or Arbiter, there are 4 stages in the learning procedure. The Combiner or Arbiter is also a classifier, which is used to generate the final prediction.

1. The classifiers (base classifiers) are trained from the initial (base-level) training subsets.

2. Predictions are generated by the learned base classifiers on the validation set (subset of the original training set).
3. A meta-level training set is composed from the predictions generated by the classifiers.
4. The final classifier meta-classifier is trained from the meta-level training set.

To explain the strategy, the following notations are used. Let x be an instance whose classification we seek, $C_1(x)$, $C_2(x)$, ... $C_k(x)$ are the predicted classifications of x from k base classifiers, C_1 , C_2 , ... C_k . Examples randomly drawn from the entire original training set constitute the validation set, E , which is used to generate the meta-level training set according to the different strategies. $\text{Class}(x)$ and $\text{attribute_vector}(x)$ denote the correct classification and attribute vector of example x as specified in the validation set, E .

Combiner

Combiner strategy refers to the use of knowledge about how classifiers behave with respect to each other. Thus, if, for example, two classifiers predict the same class and they are always correct (relative to some test set), this simple fact may lead to a powerful predictive tool.

In the Combiner strategy, the predictions of the learned base classifiers on the training set form the basis of the meta-learner's training set. A composition rule, which varies in different schemes, determines the content of training examples for the meta-learner. From these examples, the meta-learner generates a meta-classifier, a combiner. In classifying an instance, the base classifiers first generate their predictions. Based on the same

composition rule, a new instance is generated from the predictions, which is then classified by the combiner. The aim of this strategy is to “coalesce” the predictions from the base classifiers by learning the relationship between these predictions and the correct prediction. A combiner may compute a prediction that is entirely different from any proposed by a base classifier, whereas an arbiter chooses one of the predictions from the base classifiers and the arbiter itself.

Three composition rules are experimented in Chan’s work:

Class-combiner Return meta-level training instances with the correct classification and the predictions; i.e., $T = \{(\text{class}(x), C_1(x), C_2(x), \dots, C_k(x)) \mid x \in E\}$

Class-attribute-combiner Return meta-level training instances as in class-combiner with the addition of the attribute vectors; i.e., $T = \{(\text{class}(x), C_1(x), C_2(x), \dots, C_k(x), \text{attribute_vector}(x)) \mid x \in E\}$

Binary-class-combiner Return meta-level training instances similar to those in the class-combiner scheme except that each prediction, $C_i(x)$, has m binary predictions, $C_{i1}(x), C_{i2}(x), \dots, C_{im}(x)$, where m is the number of classes. Each prediction $C_{ij}(x)$, is produced from a binary classifier, which is trained on examples that are labeled with classes j and $\neg j$

Arbiter

In the Arbiter strategy, the training set for the meta-learner is a subset of the training set for the base learners; i.e. the meta-level training instances are a particular distribution of the raw training set. The predictions of the learned base classifiers for the training set and a selection rule, which varies in different schemes, determine which subset will constitute the meta-learner's training set. This contrasts with the combiner strategy, which has the same number of examples for the base classifier as for the combiner. Also, the meta-level training instances for a combiner incorporate additional information than just the raw training data. Based on this training set, the meta-learner generates a meta-classifier, in this case called an arbiter. In classifying an instance, the base classifiers first generate their predictions. These predictions, together with the arbiter's prediction and a corresponding arbitration rule, generate the final prediction. In this strategy, one learns to arbitrate among the potentially different predictions from the base classifiers, instead of learning to coalesce the predictions as in the Combiner strategy.

Three selection rules are implemented in Chan's work. In essence, the schemes select examples that are confusing to the base classifiers.

Different-arbiter Select instances from E if none of the classes in the k base predictions gathers a majority classification ($> k/2$ votes); i.e., $T = T_d = \{x \in E \mid \text{no majority } (C_1(x), C_2(x), \dots, C_k(x))\}$. The purpose of this rule is to choose data that are in some sense "confusing"; i.e., the majority of classifiers do not agree on how the data should be classified (different opinions).

Different-incorrect-arbiter Select instances with predictions that do not gather a majority, T_d , as in the first case, but also instances with predictions that have a majority but are incorrect; i.e, $T = D \cup I$, where $I = T_i = \{x \in E \mid \text{majority}(C_1(x), C_2(x), \dots, C_k(x)) \neq \text{class}(x)\}$. Note that both cases of data are lumped together.

Different-incorrect-correct-arbiter Return a set of three training sets: T_d and T_i , as defined above, and T_c with examples that have the same correct predictions; i.e., $T = \{T_d, T_i, T_c\}$, where $T_c = \{x \in E \mid \text{majority}(C_1(x), C_2(x), \dots, C_k(x)) = \text{class}(x)\}$. Here the attempt is to separate the data into three cases and distinguish each case by learning a separate “subarbiter” T_d , T_i , and T_c generate A_d , A_i , and A_c , respectively. The first arbiter is like the one computed in the first case to arbitrate disagreements. The second and third arbiters attempt to distinguish the cases when the predictions have a majority but are either incorrect or correct.

Chapter 3 INDUCTIVE LEARNING ON PARTITIONED DATA

3.1 Introduction

Based on a thorough study of the existing strategies, we try to design our scalable schemes, which can process large or distributed data sets accurately and efficiently. Some are inspired from the techniques introduced in Chapter 2 (Voting, Multiple Layer Induction) and some are brand new.

Throughout this chapter, we use the following partitioned data set (Table 2) to illustrate our schemes. This data set consists of two subsets of Disease A symptom data. Fever and cough are the features. Disease A is the class label.

	Fever	Cough	Disease A
Partition 1	0	0	0
	0	1	0
	1	0	1
	1	1	1
Partition 2	0	0	0
	0	1	1
	1	1	1
	1	1	1

Table 2 A Partitioned Data Set on Disease A

3.2 Rule-Example Conversion

The Rule-Example Conversion scheme borrows some techniques from Multiple-Layer Incremental Induction Learning (Wu and Lo, 1998), which can be used to handle the non -

static data repositories. Rule generation (inductive learning) and rule conversion are the two major steps in this scheme. Rule generation induces a set of rules from one subset of the training examples through some inductive learning algorithm (e.g., HCV and C4.5Rules). Rule conversion converts rules into examples. Example set P_c which is converted from the rule set R would be much smaller than the base set P (R is induced from P) as the number of rules in R is usually much smaller than the number of examples in P . Such a converted example set represents some essence information from one subset and can be easily utilized by other subsets to improve their learning. The Rule-Example Conversion scheme would be particularly helpful in the distributed environment, as only small data sets need to be forwarded between different sites. Furthermore, different learning algorithm can be used in each rule induction step. Generally, The Rule-Example Conversion scheme uses rule generation and rule conversion alternatively and processes subsets of the training examples sequentially. The detailed algorithm is provided in

Procedure RuleExampleConversion:

Procedure RuleExampleConversion (NumOfPar)

DataPartition(NumOfPar) ;

TempRuleSet = \emptyset ;

TempDataSet = \emptyset ;

For (j=1; j < NumOfPar; j++)

RuleSet_j = Induce(Partition_j \hat{E} TempDataSet) ;

TempDataSet = Convert(RuleSet_j) ;

EndFor

FinalRuleSet = Induce(Partition_{NumOfPar} \cup TempDataSet) ;

Return FinalRuleSet;

Converting a rule into an example can be done as follows. Suppose that Rule R_k is defined as follows

$$A_{k1} = a_{k1}, A_{k2} = a_{k2}, \dots, A_{km} = a_{km} \Rightarrow \text{Class} = C_{jk},$$

The corresponding converted example I_k will be defined in attribute A_{k1} (where $1 \in [1, m]$) with the value a_{k1} . For those undefined attributes, use #(Don't Care Value) to fill in.

Correspondingly the class label for this instance would be C_{jk} .

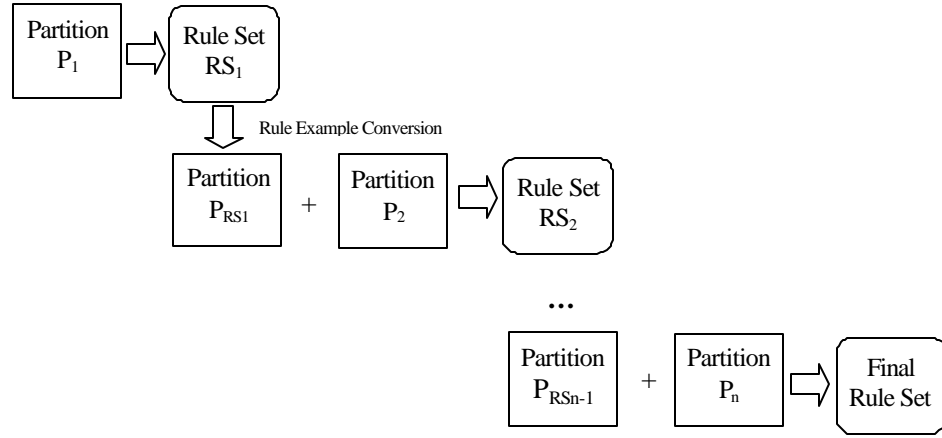


Figure 5 Rule-Example Conversion

To illustrate this approach through the sample set, suppose we get the rule:

$$\text{Fever}=1 \Rightarrow \text{Disease A}$$

after inducing on P_1' (P_1), P_2' will have a new example as shown in Table 3. The final rule set will be induced from P_2'

	Fever	Cough	Disease A
Partition 2'	0	0	0
	0	1	0
	1	1	1
	1	1	1
	1	#	1

Table 3 Expanded Data Set Through Rule-Example Conversion

3.3 Rule Weighting

Compared with the Rule-Example Conversion scheme, which learns and refines the classifier (rule set) sequentially, Rule Weighting can process each subset concurrently. This scheme is based on the assumption that the more the subsets agree with a particular rule, the more reliable this rule will be. Therefore we give each rule a weight proportional to its number of occurrences in the set of rule sets RS_j ($j \in [1, n]$). Such weights can be utilized during the deduction. Rules with high weights will have a stronger influence on the final prediction than those with low weights when a multiple-match case (An example is fired by multiple rules and these rules have different class predictions) occurs. Generally, the Rule Weighting scheme consists of two steps: 1) inducing rule set RS_j from each subset P_j ($j \in [1, n]$) and 2) integrating rule sets RS_j ($j \in [1, n]$) through rule weighting. The detailed algorithm is provided in *Procedure RuleWeighting*:

Procedure RuleWeighting (NumOfPar)
DataPartition(NumOfPar) ;
For ($j=1$; $j \leq$ NumOfPar; $j++$)

```

    RuleSetj = Induce( Partitionj ) ;
EndFor
FinalRuleSet = RuleWeighting(NumOfPar, RuleSet1,..., RuleSetNumOfPar ) ;
Return FinalRuleSet ;

```

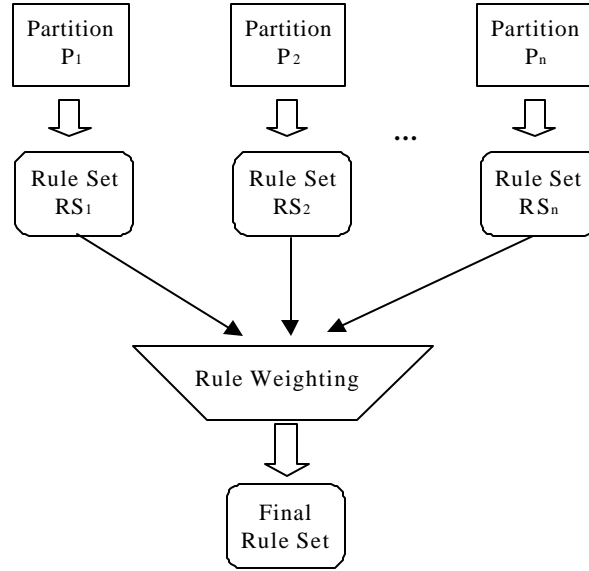


Figure 6 Rule Weighting

The following illustrates the Rule Weighting scheme. After inducing on each subset of the Disease A data set, the following rule sets RS₁ and RS₂ are generated .

RS₁: (1) fever=1 => disease A

RS₂: (1) fever=1 => disease A, (2) cough=1 =>disease A.

After weighting, Rules (1), (2) will get weight 2,1 respectively, which means that Rule(1) will be more authoritative than Rule(2) during the deduction.

3.4 Iteration

The Iteration strategy is inspired by the windowing technique (Quinlan, 1983). Suppose a classifier C (rule set) is induced using a subset of the training examples. This classifier can be used to classify examples in another subset or the same subset. Those misclassified examples represent the information unknown or vague to C . Incorporating such examples in the next iteration of learning might be helpful in generating a more powerful classifier. It is expected that 1) the size of the misclassified set will be much smaller than the size of a single subset and 2) inducing on the expanded set, which includes the misclassified examples, will not add computational cost significantly.

Theoretically, Iteration strategy is not appropriate for the noisy data sets. For example, in very noisy environments, most of the noisy data will be forwarded to the training data set in the final iteration, as noisy data will always deny rules that are generally consistent with the training set. As a result the final data set will have a higher percentage of noise than that in the whole training set.

3.4.1 Iteration

Based on Iteration strategy, the Iteration scheme works through repeatedly replenishing the initial training subset with the misclassified examples from other subsets. First, a classifier C_1 is induced from first subset P_1 . C_1 is then used to classify examples in the P_2 . Misclassified examples from P_2 are forwarded to P_1 . Such steps are repeated until all subsets are consumed. The detailed algorithm is provided in *Procedure Iteration*.

```

Procedure Iteration ( NumOfPar)
DataPartition(NumOfPar) ;
Partition1' = Partition1;
TempDataSet = f;
For (j=1; j < NumOfPar; j++)
    Classifierj = Induce( Partition1' );
    TempDataSet = Misclassified(Classifierj, Partitionj+1) ;
    Partition1' = Partition1'  $\hat{E}$  TempDataSet ;
EndFor
FinalClassifier = Induce( Partition1 ' ) ;
Return FinalClassifier

```

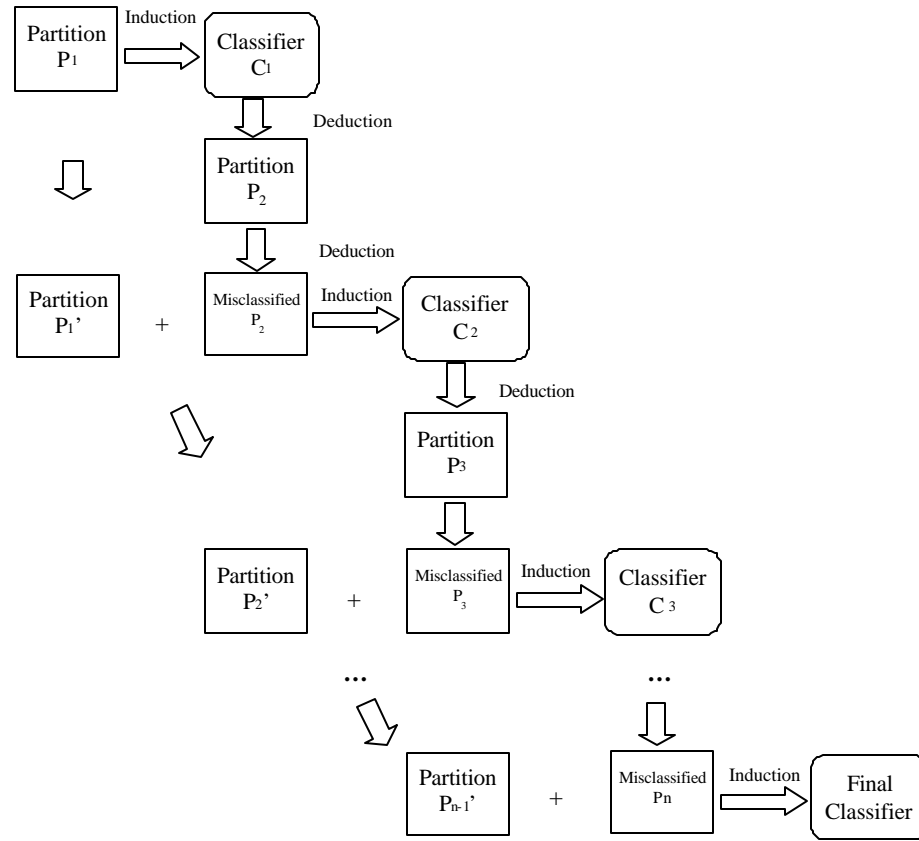


Figure 7 Iteration

To illustrate this scheme through the sample set (Table 2), suppose the following example in P_2 is misclassified by C_1 :

Fever	Cough	Disease A
0	1	1

In the next iteration P_1' will have a new example as shown in Table 4. The final rule set will be induced from P_1'

	Fever	Cough	Disease A
Partition 1'	0	0	0
	0	1	0
	1	0	1
	1	1	1
	0	1	1

Table 4 Expanded P_1' with Misclassified Examples from P_2

3.4.2 Iteration & Voting

In the Iteration scheme, the first partition is used to generate the initial classifier. Only the misclassified examples from other subsets are incorporated to generate the final classifier, which is thus biased towards the first subset. The quality of the final classifier relies mostly on the data quality of the first subset. Furthermore, Iteration scheme uses the classifier generated in the final iteration to represent the final learning result while knowledge learned in the intermediate stages is wasted. To compensate for the two shortfalls of the Iteration scheme, we designed the Iteration & Voting scheme. Contrary to the Iteration scheme, Iteration & Voting performs induction and deduction on the same subset in each iteration. Misclassified examples are forwarded to the subsequent subset. The final classifier is the combination of the classifiers generated in each iteration through Voting. The detailed algorithm is provided in *Procedure IterationVoting*:

```

Procedure IterationVoting ( NumOfPar)
  DataPartition(NumOfPar) ;
  Partition1' = Partition1;
  For (j=1; j <= NumOfPar; j++)
    Classifierj = Induce( Partitionj ' ) ;

```

$TempDataSet = Misclassified(Classifier_j, Partition_j) ;$

$Partition_{j+1}' = Partition_j \hat{\cup} TempDataSet$

EndFor

$FinalClassifier = CombineByVoting(Classifier_1, .., Classifier_{NumOfPar}) ;$

Return FinalClassifier

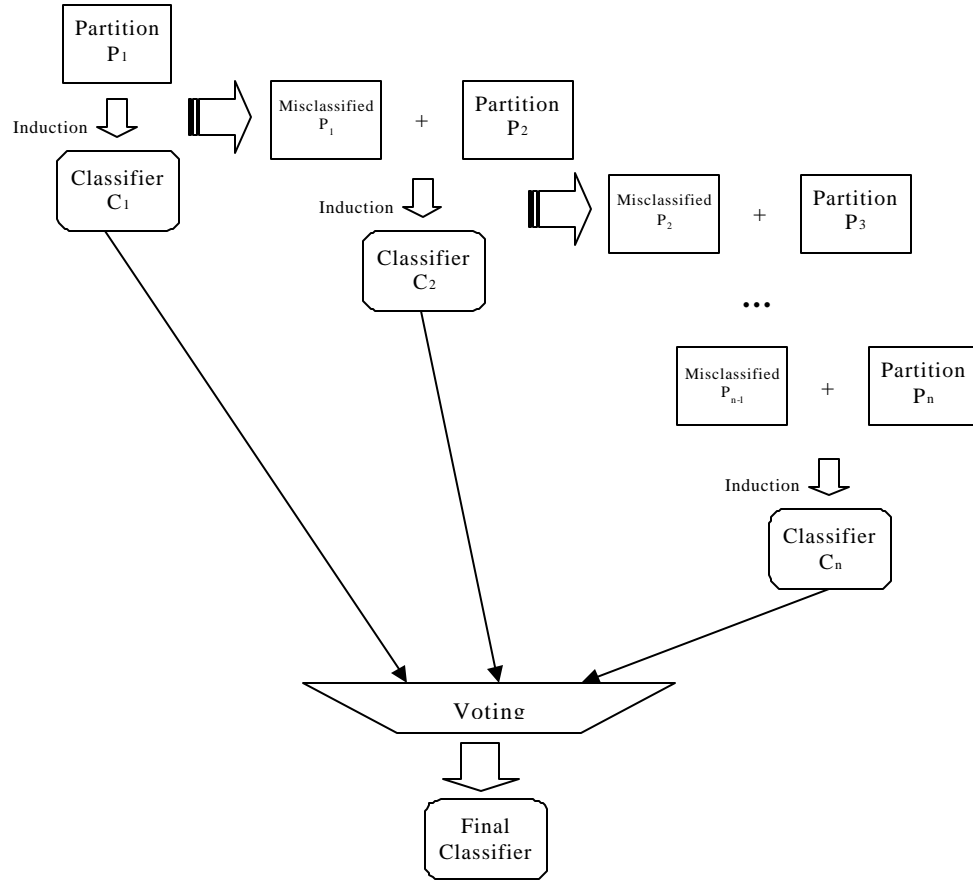


Figure 8 Iteration & Voting

To illustrate the Iteration & Voting scheme through the sample set, suppose the following example from P_1 is misclassified by C_1 :

Fever	Cough	Disease A
0	0	0

In the next iteration P_2' will have a new example as shown in Table 5. The final classifier is obtained by combining C_1 and C_2 through Voting.

	Fever	Cough	Disease A
Partition 1'	0	0	0
	0	1	0
	1	0	1
	1	1	1
Partition 2'	0	0	0
	0	1	1
	1	1	1
	1	1	1
	0	0	0

Table 5 Expanded Subets after Forwarding Misclassified Examples

3.5 Good Rule Selection

The two schemes in this category are based on the technique of selecting 'good rules'. What are 'good rules'? Generally, 'good rules' refer to those that are consistent with the training set. A rule is considered consistent if it does not cover a negative example during the evaluation. However, in some cases such as the noisy data set, this assumption may

not be true. Thus we define an accuracy threshold as “the minimum accuracy requirement for the rule to be considered ‘good’.”

3.5.1 Good Rule Selection

‘Good rules’ are not only useful in providing an accurate prediction, but also in helping “shrink” the learning scope as shown in integrative windowing. Suppose we have a rule set RS_{syn} , which is synthesized from a set of rule sets induced from the subsets of the training data. Some of the rules will be labeled as ‘good’ if their prediction accuracies over the evaluation set meet or exceed the predefined accuracy threshold. Based on our assumption that these rules are ‘good’, most of the examples in the training set would be correctly classified by the ‘good’ rules. Thus, we remove all the examples covered by these ‘good’ rules from the original training set and form a new data set with the remaining examples. We call this set an ‘uncovered’ set P_u . P_u represents some knowledge, which is lost through partitioning and can only be recovered by coalescing the ‘minority’ examples in each subset. A strong classifier can be generated through synthesizing rules induced from P_u and good rules in RS_{syn} . Based on this idea, the Good Rule Selection scheme works through combining three operations: induction, evaluation, synthesis. The detailed algorithm is provided in *Procedure GoodRuleSelection*:

Procedure GoodRuleSelection (NumOfPar)

DataPartition(NumOfPar) ;

For (j=1; j <= NumOfPar; j++)

RuleSet_j = Induce(Partition_j) ;

Rules_{whole} = Append(Rules_{whole}, RuleSet_j) ;

```

EndFor
RuleSetsyn = Synthesize( Ruleswhole, Partitionsample ) ;
RuleSetsyn' = LabelGoodRules( RuleSetsyn, Partitioneval ) ;
Partitionu = FindUncoverd( TrainingSet, RuleSetsyn' );
RuleSetu = Induce( Partitionu ) ;
Ruleswhole = Append( Ruleswhole, RuleSetu ) ;
RuleSetsyn = Synthesize( Ruleswhole, Partitionsample ) ;
Return FinalRuleSet;

```

The “rule sets synthesis operation” is a mechanism used by C4.5Rules to select and order rules (As C4.5Rules uses first hit strategy during the deduction, rules must be ordered). Given a data set P and a set of rules, the synthesizer will first partition the rules into k categories(k is the number of classes in the training set) according to the class label each rule deduce. For each category of rules, a synthesizer will choose a “best” rule subset R_p so that R_p causes the minimum ratio of false firing on data set P. All these rule subsets are then ordered by the false firing ratio on data set P. In each subset, the rules are ordered by their error rate. Figure 9 shows an ordered rule set generated from monk1 data set by C4.5Rules.

<p>Rule 1:</p> <p>attribute#5 = 1</p> <p>-> class 1 [95.3%]</p> <p>Rule 20:</p> <p>attribute#1 = 3</p> <p>attribute#2 = 3</p> <p>-> class 1 [92.2%]</p> <p>Rule 17:</p> <p>attribute#1 = 2</p> <p>attribute#2 = 2</p> <p>-> class 1 [91.2%]</p>
--

Rule 7:
attribute#1 = 1
attribute#2 = 1
-> class 1 [85.7%]

Rule 14:
attribute#1 = 1
attribute#5 = 4
-> class 0 [82.2%]

Rule 16:
attribute#1 = 2
attribute#2 = 1
attribute#5 = 4
-> class 0 [79.4%]

Rule 4:
attribute#4 = 2
attribute#5 = 2
-> class 0 [66.2%]

Rule 2:
attribute#1 = 1
attribute#5 = 2
-> class 0 [63.8%]

Rule 11:
attribute#5 = 3
attribute#6 = 1
-> class 0 [63.8%]

Rule 6:
attribute#3 = 2
attribute#5 = 2
-> class 0 [63.5%]

Rule 19:
attribute#1 = 3
attribute#2 = 2
attribute#5 = 4
-> class 0 [63.0%]

Rule 13:
attribute#3 = 2
attribute#5 = 3
-> class 0 [58.7%]

Default class: 0

Figure 9 Rules Induced by C4.5Rules on Monk1 Data Set

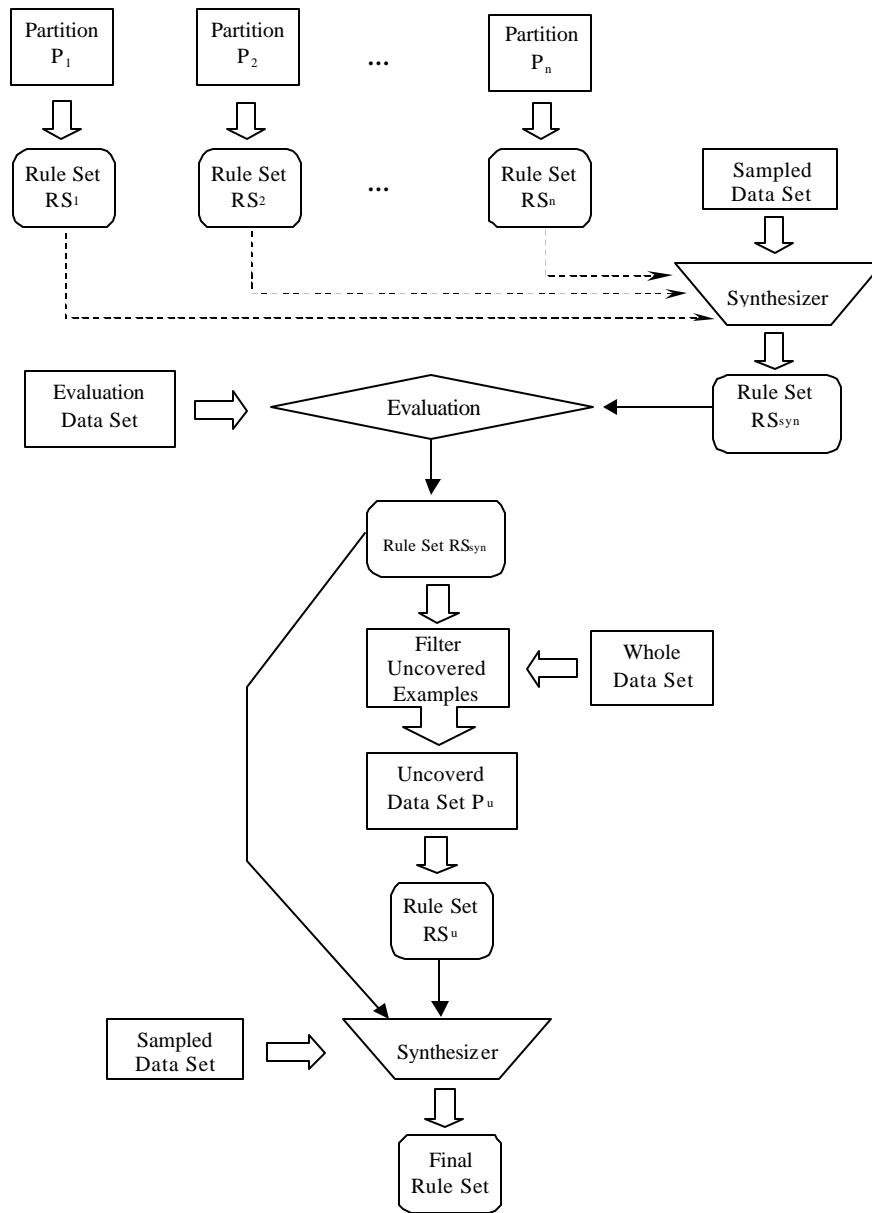


Figure 10 Good Rule Selection

3.5.2 Good Rule Selection & Voting

Good Rule Selection & Voting integrates both Voting and the Good Rule Selection strategies. Voting is the simplest and often the most effective method to integrate the predictions. In the Weighted Voting scheme, each classifier is given a weight determined by how accurate the classifier performs on the evaluation set. Inspired by this idea, we designed the Good Rule Selection & Voting scheme. We refine the granularity of weighted elements. Here rules are weighted rather than the classifiers. Furthermore, a special voting member RS_{syn} , which is synthesized from rules collected from rule sets RS_j ($j \in [1, n]$), is incorporated. Similar to the Good Rule Selection scheme, some of the rules in RS_{syn} and RS_j ($j \in [1, n]$) will be labeled as ‘good’ after evaluation. ‘Good’ rules are given higher weights when they are used to deduce a class label for the testing examples. ‘Good’ rules in RS_{syn} have higher weights than those in RS_j ($j \in [1, n]$) as RS_{syn} is constructed from a global view. The detailed algorithm is provided in *Procedure GoodRuleVoting*:

```
Procedure GoodRuleVoting ( NumOfPar)
  DataPartition(NumOfPar) ;
  Ruleswhole =  $\mathbf{F}$ ;
  For ( $j=1$ ;  $j \leq NumOfPar$ ;  $j++$ )
    RuleSetj = Induce( Partitionj ) ;
    Ruleswhole = Append(Ruleswhole, RuleSetj) ;
  EndFor
  RuleSetsyn = Synthesize(Ruleswhole, Partitionsample) ;
  RuleSetsyn' = LabelGoodRules(RuleSetsyn, Partitioneval) ;
```

For ($j=1; j \leq \text{NumOfPar}; j++$)

$\text{RuleSet}_j' = \text{LabelGoodRules}(\text{RuleSet}_j, \text{Partition}_{eval}) ;$

EndFor

$\text{FinalRuleSet} = \text{CombineByWeightedVoting}(\text{RuleSet}_{syn}', \text{RuleSet}_1', \dots, \text{RuleSet}_{\text{NumOfPar}}')$

Return FinalRuleSet

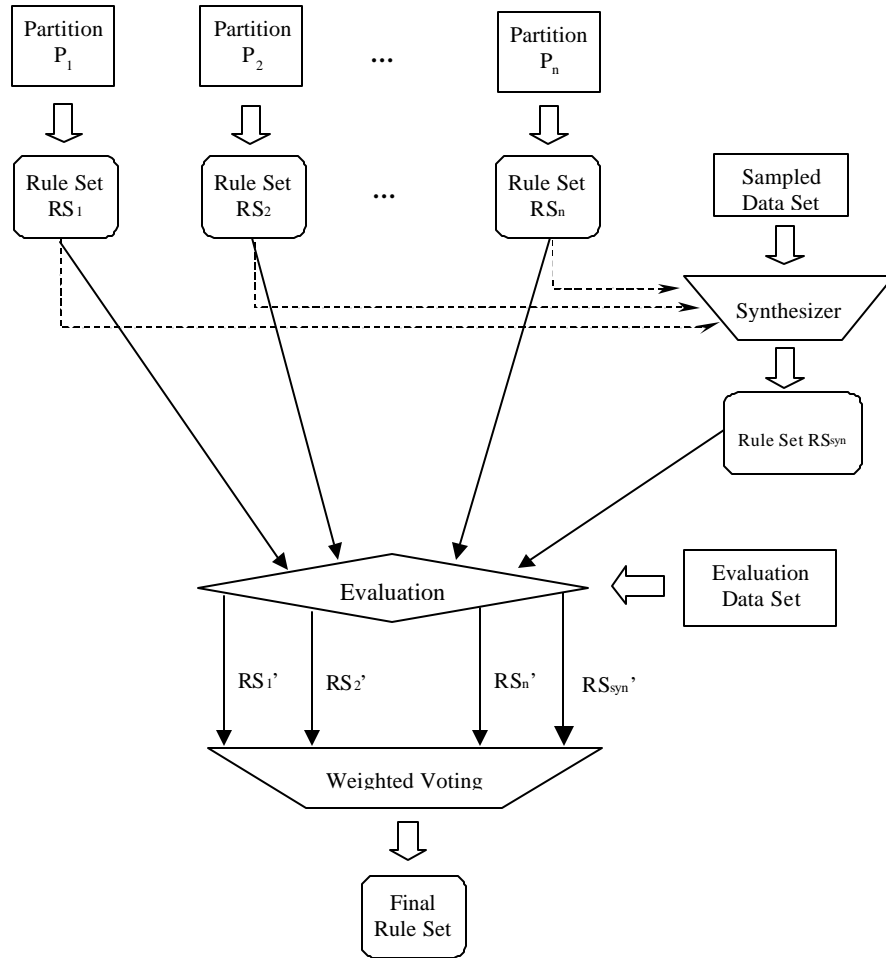


Figure 11 Good Rule Selection & Voting

3.6 Data Dependent Rule Selection

As introduced in the Good Rule Selection scheme, given a set of rules RS_{whole} and a data set P , a synthesizer can generate a sifted and ordered rule set RS_{syn} from RS_{whole} . This sifted rule set RS_{syn} has the highest prediction power over the data set P . For the scalable algorithm problem we are trying to solve, suppose in a distributed environment, each local site can obtain the rules generated from other sites. Can such rules help generate a stronger local classifier? To solve this interesting question, we designed the Data Dependent Rule Selection scheme. Similar to the Good Rule Selection scheme, first a rule set RS_j ($j \in [1, n]$) is induced from each subset of the training examples. Following that all the rules in RS_j ($j \in [1, n]$) are gathered together to form a set of rules RS_{whole} . We use RS_{whole} and each data subset P_j to generate a new local rule set RS'_j ($j \in [1, n]$) through the synthesizer. Finally, rule sets RS'_j ($j \in [1, n]$) are combined through Voting. Assuming that the size of the rule file would be much smaller than the size of data file, thus rule information can be easily exchanged between different sites. One advantage of the Data Dependent Rule Selection scheme is that it avoids revealing confidential information caused by exchanging data set directly. The detailed algorithm is provided in *Procedure DataDependentRuleSelection*:

Procedure DataDependentRuleSelection (NumOfPar)

DataPartition(NumOfPar) ;

Rules_{whole} = \mathbf{F} ;

For (j=1; j <= NumOfPar; j++)

RuleSet_j = Induce(Partition_j) ;

```

    Ruleswhole = Append( Ruleswhole, RuleSetj )
  EndFor
  For (j=1; j <= NumOfPar; j++)
    RuleSet'j = Synthesize( Ruleswhole, Partitionj ) ;
  EndFor
  FinalRuleSet = CombineByVoting( RuleSet'1, .. , RuleSetNumOfPar' ) ;
  Return FinalRuleSet ;

```

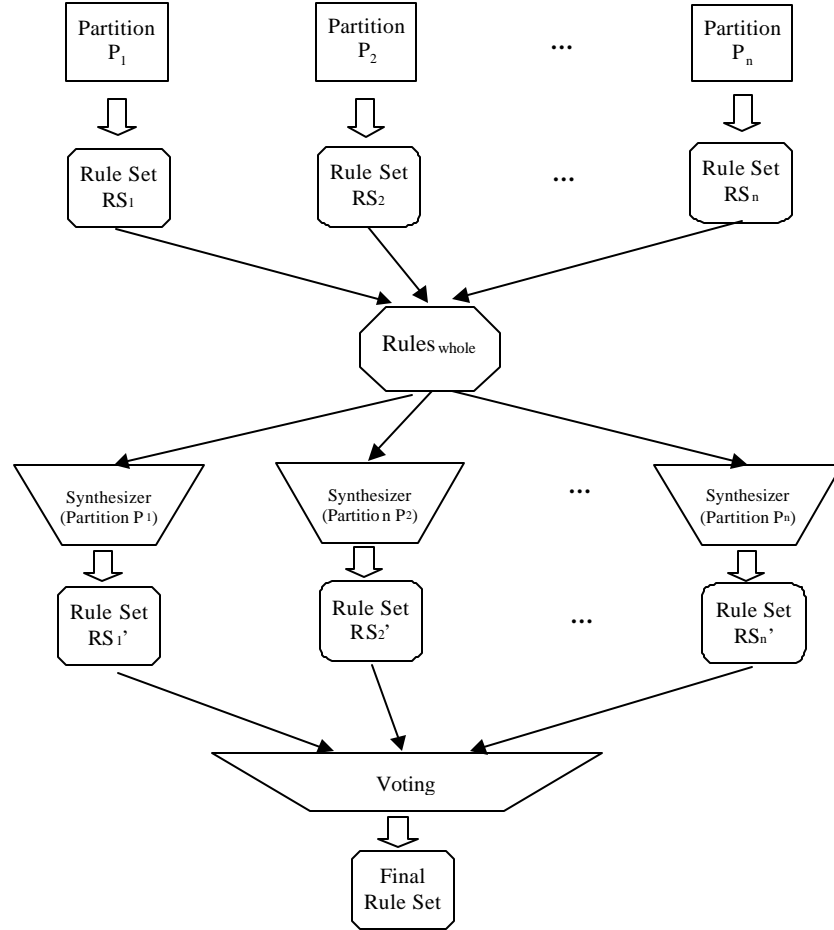


Figure 12 Data Dependent Rule Selection

3.7 Summary

In this chapter, we identified five strategies to achieve scalability. Seven corresponding scalable schemes are designed and developed. Table 6 summarizes their common characteristics and differences as shown below:

	Rule- Example Conversion	Rule Weighting	Iteration	Iteration & Voting	Good Rule Selection	Good Rule Selection & Voting	Data Dependent Rule Selection
Can Handle Continuous Attributes	N	N	Y	Y	Y	Y	Y
Can Handle Exclusive Conditions in Rule	N	N	Y	Y	Y	Y	Y
Can Process Subsets Concurrently	N	Y	N	N	Y	Y	Y
Applicable to Different Learning Algorithms	Y	N	Y	Y	N	N	N
Theoretically Noise Sensitive	N	N	Y	Y	N	N	N
Need to Expand the Original Training Subset	Y	N	Y	Y	Y	N	Y

Table 6 Characteristics of 7 Scalable Schemes

Chapter 4 EXPERIMENTS

This chapter lists the experimental results for the seven schemes discussed in Chapter 3. Major findings for each scheme are provided after each result. These experiments are conducted at different time periods while exploring the scalable learning schemes. Not all the data sets were used in different schemes due to the adjustments made during the experiments and the limitations of some schemes. Also different underlying inductive learning algorithms were chosen for different schemes.

4.1 Test Methodologies

The data sets used in the experiments are mainly from UCI 's (University of California at Irvine) machine learning repository. One artificial data set named "ibm", generated from IBM data set generator, is used. These represent a mixture of nominal and continuous data, small and large sets. The characteristics of these data sets are provided in Table 7 .

Database Name	Source	Training Set Size	Testing Set Size	Contain Noise	Number of Symbolic Attributes	Number of Continuous Attributes	Number of Classes
h-r	UCI	132	28	No	4	0	3
monk1	UCI	124	432	No	6	0	2
monk2	UCI	169	432	YES	6	0	2
monk3	UCI	122	432	YES	6	0	2
vote	UCI	300	135	No	16	0	2
car	UCI	1503	288	No	6	0	4
ibm	IBM Data Generator	2000	400	No	3	6	4
adult	UCI	11693	2824	No	8	6	2
connect	UCI	14568	3590	No	42	0	2
splice	UCI	2687	503	No	60	0	3
connectn	UCI	14568	3590	Yes	42	0	2

Table 7 Data Set Properties

In presenting our results in this chapter, we use the following terminologies to indicate the different experiment conditions:

- HCV, C4.5, and C4.5Rules are the underlying inductive learning algorithms used to develop the scalable schemes. We use HCV, C4.5 and C4.5Rules to represent the prediction accuracy of the global classifier learned from the whole data set by applying HCV, C4.5 and C4.5Rules respectively. The result of a specific algorithm is provided if we use it as the underlying learning algorithm.
- We call the classifier generated from a subset of training set a base classifier. We use avg-base and max-base to represent the average accuracy and the highest accuracy of the base classifiers respectively.

- P_n means the whole data set is partitioned into n equal-sized subsets before applying different schemes. In all experiments, we used stratified partitioning (The class distribution in the subset is the same as the whole data set). We also ensure each subset is disjointed with each other.
- T_n is used in the experiments of the Good Rule Selection and Good Rule Selection & Voting schemes, which means a rule is labeled as ‘good’ only if it has a prediction accuracy higher than n percent over the evaluation set.
- E_n is only used in the experiments of the Good Rule Selection and Good Rule Selection & Voting schemes, which means the evaluation set is n times of the size of the subset.
- The default parameters of HCV, C4.5, C4.5Rules are used to induce the base classifiers and the global classifier.
- We use the following acronyms to represent different schemes (Table 8).

Acronym	Description
REC	Rule Example Conversion
RW	Rule Weighting
IT	Iteration
VT	Voting
IV	Iteration & Voting
GRS	Good Rule Selection
GRSV	Good Rule Selection & Voting
DDRS	Data Dependent Rule Selection
Avg-Base	Average Prediction Accuracy of the Base Classifiers
Max-Base	Maximum Prediction Accuracy of the Base Classifiers

Table 8 Scheme Acronyms

4.2 Rule-Example Conversion

Table 9 shows the experimental results of the Rule-Example Conversion scheme and the major findings are listed below:

- Generally Rule-Example Conversion scheme does not maintain the accuracy of the global classifier. Such an accuracy degradation can be caused by two things: 1) losing information as a result of partitioning and 2) overgeneralization in Rule - Example Conversion as one rule that covers many examples in one subset is treated only as one example in another subset.
- It is expected that Rule Example Conversion dilutes the effects of noise as examples converted from the rules are not supposed to be noisy. Further investigation must be done to verify the accuracy improvement in the data set monk2 where the results of P7 and P9 beat the global classifier is the effect of noise dilution.
- Limitation of Rule-Example Conversion scheme is found through implementation: inability to process the continuous attributes and exclusive conditions of a rule in the rule-example conversion. For example, suppose we have one rule:

$$A1 < 1 \text{ and } A2 \neq b \Rightarrow \text{Class 1,}$$

How should such a rule be processed? How many examples should be generated from this rule?

- Through the above analysis, we conclude that Rule-Example Conversion is not the most effective method to generate knowledge, which can further be used by different iterations of learning.

	C4.5	P3	P5	P7	P9	P11
car	89.6	85.8	85.1	77.8	79.2	77.8
h-r	92.9	78.6	46.4	75	71.4	67.9
monk1	75.7	70.8	70.8	75	75	75
monk2	65	63.9	55.1	67.1	67.1	62.5
monk3	97.2	97.2	80.6	80.6	80.6	80.6
vote	97	97	97	95.6	97	97

Table 9 Results (Rule-Example Conversion)

4.3 Rule Weighting

Table 10 lists the results for the Rule Weighting scheme and the major findings are listed below:

- Experimental result demonstrates that Rule Weighting scheme does not maintain the accuracy of the global classifier consistently. A better result is achieved sometimes; nevertheless, accuracy boostings do not occur regularly.
- Through observing the rules generated from each subset, we find that it is quite rare for one particular rule to appear repeatedly in different rule sets. This can be caused by the complexity of the rule space or rule pruning in different stages. Therefore effective rule weighting cannot be achieved.

- Similar to Rule-Example Conversion scheme, Rule Weighting is incapable in dealing with the continuous values. Suppose we have two rules

1) $a < 1 \Rightarrow \text{class 1}$

2) $a \leq 1 \Rightarrow \text{class 1}$

Should these two rules be treated as the same or not ?

	HCV	P3	P5	P6	P9
car	91.319	90.278	88.542	88.194	84.722
h-r	78.571	67.857	82.143	57.143	78.571
monk1	100	73.38	72.685	76.157	65.278
monk2	65.509	60.185	63.194	59.259	60.185
monk3	97.222	98.148	96.991	97.222	97.222
vote	97.037	96.296	97.037	97.037	96.296

Table 10 Results (Rule Weighting, HCV)

4.4 Iteration

Table 11 to Table 13 list the results for the Iteration scheme and the major findings are listed below:

- Similar to the Rule Weighting scheme, accuracy boosting does occur sometimes. But it is unknown when such improvements will happen. This is concluded from two groups of experimental results based on the two learning algorithms (HCV and C4.5) as shown in Table 11 and Table 12.

	HCV	P3	P5	P7	P9	P11
car	91.319	90.972	88.889	85.764	87.847	87.153
h-r	78.571	75	75	85.714	64.286	82.143
monk1	100	76.157	91.204	80.324	78.009	77.546
monk2	65.509	64.583	59.722	64.815	62.731	67.361
monk3	97.222	97.222	88.889	97.222	87.963	94.676
vote	97.037	95.556	94.815	97.037	97.037	94.074

Table 11 Results (Iteration, HCV)

	C4.5Rules	P3	P5	P7	P9	P11	P13	P15
car	93.8%	91.00%	92.40%	92.70%	96.50%	94.40%	87.20%	91.70%
ibm	80.8%	77.20%	74.80%	70.80%	73.20%	66.20%	71.20%	69.00%

Table 12 Results (Iteration, C4.5Rules)

- Table 13 shows the performance of the Iteration scheme on the data set connectn, which contains 10% of manually injected noise (X Zhu, X Wu, Q Chen, 2003). Severe accuracy droppings can be seen at some spots, which demonstrates the poor performance of the Iteration scheme in the noisy environment. Furthermore, Iteration can not maintain the performance of Voting in dealing with noisy data .

connectn	C4.5Rules		P4	P8	P12	P16	P20
	75.2%	IT	62.40%	49.60%	62.70%	72.30%	51.20%
		VT	78.40%	76.00%	74.20%	74.10%	72.90%

Table 13 Results (Iteration, Noisy Data Set)

4.5 Iteration&Voting

Table 14 to Table 16 list the results for the Iteration & Voting scheme and the major findings are listed below:

- As shown in Table 14 and Table 15, Iteration & Voting does not maintain the accuracy of the global classifier consistently.
- Iteration&Voting can outperform Voting consistently. Such an accuracy boosting becomes more significant when the number of subsets increases. Therefore Iteration strategy can recover some of the lost knowledge caused by partitioning. This conclusion is further confirmed from experimental result on large set, as shown in Table 15.
- Iteration & Voting outperforms both Iteration and Voting. Therefore the integration of the two techniques, Voting and Iteration, can boost the accuracy of the final classifier.

	C4.5Rules		P3	P5	P7	P9	P11	P13	P15
car	93.80%	IV	91.70%	91.70%	89.90%	89.60%	87.80%	88.50%	86.50%
		VT	92.40%	86.50%	88.50%	87.80%	87.80%	87.50%	84.70%
ibm	80.80%	IV	81.00%	77.00%	79.00%	76.00%	75.50%	71.80%	75.00%
		VT	77.80%	76.00%	72.80%	70.20%	73.50%	70.50%	69.00%

Table 14 Results (Iteration &Voting)

	C4.5Rules		P4	P8	P12	P16	P20
connect	85.9%	IV	85.50%	83.50%	82.90%	82.80%	81.50%
		VT	82.30%	81.70%	79.90%	81.00%	79.60%

Table 15 Results (Iteration &Voting, Large Set)

- To find out whether Iteration & Voting is more capable than Iteration in the noisy environments, we tried Iteration & Voting on connectn. The result (Table 16) demonstrates that Iteration&Voting outperforms both Iteration and Voting in

dealing with noisy data. Unexpectedly, Iteration&Voting can even outdo the global classifier sometimes in the noisy environments.

	C4.5Rules		P4	P8	P12	P16	P20
connectn	75.20%	IV	75.20%	77.90%	77.60%	78.00%	78.80%
		IT	62.40%	49.60%	62.70%	72.30%	51.20%
		VT	78.40%	76.00%	74.20%	74.10%	72.90%

Table 16 Results (Iteration & Voting, Noisy Data Set)

4.6 Good Rule Selection

Table 17 shows the experimental results of the Good Rule Selection scheme and the major findings are listed below:

- Good Rule Selection can achieve a higher accuracy than the global classifier sometimes. However, regularities of accuracy improvement can not be found.
- Generally, the higher the specified ‘good’ rule accuracy threshold, the higher the accuracy of the final rule set will be, since less rules will be labeled as ‘good’ rules, the uncovered data set will have more examples, and consequently, the higher the quality of the rules generated from the uncovered set.
- One drawback of this scheme is found through implementation: the size of the uncovered set is uncontrollable, as it relies on the accuracy threshold for ‘good’ rule and the initial data quality. So this scheme can not deal with a situation where the uncovered data set is too large to be processed with one iteration of learning.

car	C4.5Rules		P3	P5	P7	P9	P11
	93.80%	T95	91.70%	93.40%	93.40%	91.00%	94.10%
		T90	91.70%	93.40%	93.40%	89.90%	94.10%
		T85	89.20%	91.00%	89.90%	89.90%	88.50%
		T80	89.20%	91.70%	88.50%	89.90%	86.10%
		T70	89.90%	92.70%	88.50%	87.50%	86.10%
ibm	C4.5Rules		P3	P5	P7	P9	P11
	80.80%	T95	78.50%	79.20%	80.20%	81.80%	78.00%
		T90	78.50%	78.00%	80.20%	78.50%	77.20%
		T85	78.00%	78.50%	77.00%	78.80%	77.50%
		T80	76.00%	83.50%	77.00%	78.80%	77.50%
		T70	74.00%	77.80%	76.20%	76.50%	76.00%

Table 17 Results (Good Rule Selection)

4.7 Good Rule Selection & Voting

Table 18 to Table 20 show the results of Good Rule Selection & Voting scheme (evaluation data set size = 2* size of one subset). The major findings are listed below:

- Good Rule Selection & Voting does not maintain the accuracy of the global classifier as the number of subset increases.
- Overall, Good Rule Selection & Voting outperforms Voting. Therefore the increased granularity of the weighted element and good rule labeling is useful in generating a more accurate classifier. But a consistent accuracy improvement can be achieved only when the accuracy threshold $\geq 90\%$, which means the rules to be considered as ‘good’ should be truly good.

car	C4.5Rules		P3	P5	P7	P9	P11
	93.80%	T95	94.10%	91.00%	89.90%	87.80%	88.20%
		T90	93.40%	91.00%	89.60%	87.80%	88.20%
		T85	92.70%	90.60%	89.60%	87.80%	87.50%
		T80	92.00%	90.60%	89.60%	87.80%	87.50%
		T70	92.40%	91.00%	89.60%	88.20%	87.80%
		Voting	92.40%	86.50%	88.50%	87.80%	87.80%
ibm	C4.5Rules		P3	P5	P7	P9	P11
	80.80%	T95	78.80%	77.50%	74.50%	71.50%	74.20%
		T90	77.80%	75.50%	74.50%	71.20%	74.50%
		T85	77.50%	75.50%	74.20%	72.80%	73.80%
		T80	77.50%	75.50%	75.80%	73.80%	74.00%
		T70	76.80%	76.00%	75.50%	74.80%	73.80%
		Voting	77.80%	76.00%	72.80%	70.20%	73.50%

Table 18 Results (Good Rule Selection & Voting)

- To verify whether Good Rule Selection & Voting can outperform Voting in dealing with the large data sets, we tried Good Rule Selection-Voting on adult and connect (evaluation data set size = 2* size of one subset). The result(Table 19) further strengthens that Good Rule Selection & Voting can indeed outperform Voting.

	C4.5Rules		P4	P8	P12	P16	P20
adult	85.1%	T5	85.30%	85.00%	85.00%	85.10%	85.10%
		VT	85.00%	84.70%	84.90%	85.10%	85.10%
connect	85.9%	T5	83.20%	82.50%	81.40%	81.70%	80.90%
		VT	82.70%	82.30%	81.30%	81.70%	80.60%

Table 19 Results (Good Rule Selection & Voting, Large Sets)

- To explore the relationship between the size of the evaluation set and the accuracy of the final classifier, we performed another experiment, where the only changing

condition is the size of evaluation set (the other two parameters are T5, P9). As shown in Table 20, the size of the evaluation set is loosely related the accuracy of the final classifier when the size of the evaluation set $> 1 * \text{size of a subset}$.

	E1	E2	E3	E4	E5	E6	E7	E8	E9
car	86.80%	87.80%	87.80%	87.80%	87.80%	87.80%	87.80%	87.80%	87.80%
ibm	71.20%	71.50%	71.20%	71.20%	71.20%	71.20%	71.20%	71.20%	71.20%

Table 20 Results (Good Rule Selection & Voting, with Different Sizes of Evaluation Set)

4.8 Data Dependent Rule Selection

Table 21 shows the results of the Data Dependent Rule Selection scheme as compared with the Voting scheme. The major findings are listed below:

- Data Dependent Rule Selection does not maintain the accuracy of Voting when the number of subsets is small. But significant accuracy improvement can be found as the number of subsets increases. Therefore incorporating rules generated from other learning processes can help local learning and compensate the information loss caused by partitioning.
- Contrary to Data Dependent Rule Selection, Good Rule Selection & Voting performs well when the number of subsets is small. Therefore it is expected that combining these two strategies can help generate a more accurate classifier. The experimental result (Table 21) confirmed our expectation.

car	C4.5Rules		P3	P5	P7	P9	P11	P13	P15
	0.938	DDRS	92.70%	89.60%	88.20%	87.80%	89.60%	91.30%	86.80%
		SV	92.40%	86.50%	88.50%	87.80%	87.80%	87.50%	84.70%
		GRSV-DDRS	93.40%	89.90%	88.90%	88.20%	89.60%	91.30%	87.20%
ibm	C4.5Rules		P3	P5	P7	P9	P11	P13	P15
	0.808	DDRS	77.00%	75.20%	73.50%	75.20%	76.20%	76.50%	76.00%
		SV	77.80%	76.00%	72.80%	70.20%	73.50%	70.50%	69.00%
		GRSV-DDRS	77.80%	76.00%	73.80%	75.20%	76.50%	76.20%	75.80%

Table 21 Results (Data Dependent Rule Selection)

4.9 Time Complexity Analysis

Table 22 lists the training time of the different schemes on connect as compared with that of C4.5Rules. Due to the limitations of the Rule Example Conversion and the Rule Weighting scheme (incapability in dealing with continuous attributes and exclusive conditions in rules), no experimental results of these two schemes are provided. The following conclusion can be drawn from the result:

- All the schemes evaluated here require significantly less training time than that of running C4.5Rules over the whole data set. This observation conforms to our expectation, that is, if a learning algorithm's time complexity is more than linear in the number of examples, processing small, fixed size samples sequentially can reduce the complexity to linear
- In the Good Rule Selection & Voting and the Iteration & Voting schemes, training time is inversely proportional to the number of partitions. This result further demonstrates the timesaving property of the data reduction technique.

C4.5Rules		P8	P16	P24	P32	P40
4661	GRSV	38	22	19	18	13
	DDRS	401	476	561	667	585
	IV	240	121	84	58	49

Table 22 Training Time of GRSV, DDRS, IV and C4.5Rules (unit seconds)

4.10 Experiment Summary

Table 23, Table 24 and Table 25 summarize the experimental results(car, splice, ibm) of all the schemes we investigated. To compare the quality of the classifier generated by integration techniques with the quality of the base classifier, which is induced from a subset of the training set, we list the avg-base and max-base. We can see that most of our integration schemes can outperform avg-base and max-base. Therefore sampling of the training data is definitely not sufficient to generate accurate classifier and integration is necessary and helpful. However, none of our scheme can main tain the accuracy of the global classifier as the number of subsets increases. Iteration and Data Dependent Rule Selection are the two most promising strategies. Combined with Voting, they can generate schemes, which outperform Voting consistently. Furthermore Data Dependent Rule Selection is the best scheme to compensate the information loss caused by partitioning as the number of subsets increases.

	Global Classifier		P3	P5	P7	P9	P11
C4.5	89.60%	REC	85.80%	85.10%	77.80%	79.20%	77.80%
		Max-Base	87.50%	84.40%	83.70%	81.90%	79.50%
		Avg-Base	84.70%	81.32%	79.88%	79.08%	76.92%
C4.5Rules	93.80%	IT	91.00%	92.40%	92.70%	96.50%	94.40%
		IV	91.70%	91.70%	89.90%	89.60%	87.80%
		GRS	91.70%	93.40%	93.40%	91.00%	94.10%
		DDRS	92.70%	89.60%	88.20%	87.80%	89.60%
		GRSV	94.10%	91.00%	89.90%	87.80%	88.20%
		GRSV-DDRS	93.40%	89.90%	88.90%	88.20%	89.60%
		VT	92.40%	86.50%	88.50%	87.80%	87.80%
		Max-Base	89.60%	89.60%	86.80%	86.50%	84.70%
		Avg-Base	87.60%	85.90%	84.30%	81.00%	79.00%
HCV	91.32%	RW	90.28%	88.54%	88.19%	84.72%	83.55%
		IT	90.97%	88.89%	85.76%	87.85%	87.15%
		Max-Base	88.19%	82.63%	86.80%	84.37%	79.86%
		Avg-Base	84.72%	79.23%	79.61%	75.46%	75.03%

Table 23 Results (7 Schemes Comparison on Car Data Set)

C4.5Rules		P6	P9	P12	P15	P18	P21
93.60%	IT	89.90%	92.00%	90.30%	86.70%	91.70%	88.90%
	IV	93.60%	94.20%	93.80%	93.00%	92.40%	90.10%
	GRS	93.60%	94.20%	93.80%	93.00%	92.40%	90.10%
	GRSV	94.00%	93.80%	94.20%	93.20%	93.00%	90.50%
	DDRS	94.40%	94.00%	93.80%	94.00%	94.00%	94.00%
	GRS-DDRS	93.80%	94.00%	93.80%	94.00%	94.00%	94.20%
	VT	93.40%	93.60%	93.80%	93.00%	90.30%	87.70%
	Avg-Base	86.77%	86.07%	84.89%	82.06%	79.37%	76.71%
	Max-Base	90.50%	91.80%	92.40%	90.50%	87.70%	84.30%

Table 24 Results (5 Schemes Comparison on Splice Data Set)

C4.5Rules		P3	P5	P7	P9	P11	P13	P15
80.80%	IT	77.20%	74.80%	70.80%	73.20%	66.20%	71.20%	69.00%
	VT	81.00%	77.00%	79.00%	76.00%	75.50%	71.80%	75.00%
	GRS	78.50%	79.20%	80.20%	81.80%	78.00%	75.50%	75.00%
	GRSV	78.80%	77.50%	74.50%	71.50%	74.20%	70.50%	70.00%
	DDRS	77.00%	75.20%	73.50%	75.20%	76.20%	76.50%	76.00%
	GRS-DDRS	77.80%	76.00%	73.80%	75.20%	76.50%	76.20%	75.80%
	VT	77.80%	76.00%	72.80%	70.20%	73.50%	70.50%	69.00%
	Max-Base	74.80%	71.20%	73.20%	70.50%	69.00%	70.20%	70.00%
	Avg-Base	73.10%	68.24%	66.47%	65.39%	64.07%	63.32%	60.68%

Table 25 Results (5 Schemes Comparison on IBM Data Set)

Chapter 5 SUMMARY AND FUTURE WORK

Summary

With the rapid development of computer technology, machine learning systems are challenged to extract knowledge from huge data repositories. Although processor speed and memory capacity increase at an amazing pace, data generated and accumulated grow at an even faster pace. In this thesis, we attempt to address the problem of conducting inductive learning on large or distributed data repositories. Our goal is to design some scalable schemes through which accurate classifiers can be generated efficiently. Data Reduction technique is used as an underlying principle to achieve scalability.

Five strategies (Rule-Example Conversion, Rule Weighting, Iteration, Good Rule Selection, Data Dependent Rule Selection) have been investigated. Rule-Example Conversion treats examples converted from rules as the sharing knowledge between different iterations of learning. Rule Weighting weights rules according to their occurrences in sets of rule sets. Weights will be used in deduction. Iteration tries to forward misclassified examples between different learning processes as misclassified examples represent the knowledge that the original classifier is ignorant about. Good Rule Selection uses ‘good rule’ to shrink the learning scope. Data Dependent Rule

Selection uses rules from other learning iterations to improve local learning. A substantial number of experiments have been performed to evaluate the schemes in each strategy with different constituent learning algorithms.

Our study shows that classifiers trained individually from random subsets of a large data set are not as accurate as integrating a collection of separately learned classifiers. Therefore integration is necessary. The integration technique yields significantly higher accuracy than a classifier trained from a random set (average 10% improvement). Furthermore, our integration schemes require significantly less training time than that of generating a global classifier over the global data set (average 90% improvement).

Among all the schemes we designed and evaluated, Iteration&Voting and Good Rule Selection – Data Dependent Rule Selection are the two most promising integration schemes, which can outperform Voting consistently. Consequently, combining knowledge from other learning processes can improve local learning.

Future Work

- Some of our strategies such as Iteration&Voting can be applied to different inductive learning algorithms. Further experiments that use other learning algorithms can be performed to strengthen our conclusion.

- Some of our strategies such Good Rule Selection can only be used with some particular inductive learning algorithms. Strategy improvement needs to be done to make it independent of underlying algorithms.
- Further experiments can be done to verify whether the schemes we developed can outperform the global classifier consistently in situations where the quality of the global classifier is poor (predication accuracy $\leq 60\%$).
- Further investigation can be done to verify the accuracy improvement (compared with the global classifier) in the noisy data set in the Rule-Example Conversion scheme is the effect of noise dilution.
- Further experiments can be done to find out the reason why Iteration&Voting outperforms Voting consistently in the noisy environment.
- Further experiments can be done to find out the reason why accuracy improvement of Data Dependent Rule Selection over Voting becomes increasingly significant when the number of subsets increases.
- To further strengthen our conclusion, K-Folder Cross Validation (dividing the original data set (training + test if available) into k subsets and performs the same function k times. Each time, one of the k subset is used as the test set and the other k-1 subsets are put together to form a training set. The average accuracy across all k trials is treated as the final prediction accuracy) can be done.

REFERENCES

- Breiman, L. (1994). Bagging Predictors. (Technical Report 421), Berkeley, CA: Dept. of Statistics, Univ. of California.
- Catlett, J. (1991) Megainduction: Machine learning on very large databases. Ph.D. thesis, School of Computer Science, University of Technology, Sydney, Australia.
- Carbonell, J. (1989). Introduction: Paradigms for machine learning. *Artificial Intelligence*, 40, 1-9.
- Chan, Philip Kin-Wah (1996), An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning. Ph.D. Thesis Columbia University.
- Cheseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., & Freeman, D. (1988). Autoclass: A bayesian classification system. *Proc. Fifth Intl. Conf. Machine Learning* (pp. 54-64).
- Clark, P. & Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3, (pp. 261-285).
- Dietterich, T. & Michalski, R. (1983). A comparative review of selected methods for learning from examples. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, (pp. 331-363). Morgan Kaufmann.

Fayyad, U. M., G. Piatetsky-Shapiro, and P. Smyth (1996). From data mining to knowledge discovery: An overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthursamy (Eds.), *Advances in Knowledge Discovery and Data Mining*, pp. 1–34. Menlo Park, CA: AAAI Press.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2, (pp.139-172).

Fürnkranz, J. (1998). Integrative windowing. *Journal of Artificial Intelligence Research* 8, (pp. 129–164).

Harris-Jones, C. and T.L. Haines (1997) Sample size and misclassification: Is more always better ? Working Paper AMSCAT-WP-97-118, AMS Center for Advanced Technologies.

Huber, P. (1997). From large to huge: A statistician's reaction to KDD and DM. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, (pp. 304–308), AAAI Press.

Littlestone, N. & Warmuth, M.(1989) The weighted majority algorithm (Technical Report UCSC-CRL-89-16), Santa Cruz, CA: Computer Research Lab., Univ. of California.

Hunt, E.B., Marin, J. and Stone, P. (1966) "*Experiments in Induction*" New York: Academic Press.

Michalski, R. (1983). A theory and methodology of inductive learning. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, (pp. 83-134). Morgan Kaufmann.

Michalski, R. & Stepp, R. (1983). Learning from observation: Conceptual clustering. In R. Michalski, J. Carbonell, & T. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, (pp. 331-363). Morgan Kaufmann.

Michalski, R. S., Mozetic, I., Hong, J., & Lavrac, N. (1986). The multipurpose incremental learning system AQ51 and its testing application to three medical domains. *Proc. AAAI-86* (pp. 1041-1045).

Provost, F. and V. Kolluri (1997). Scaling Up inductive algorithms: An overview. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, Menlo Park, CA, (pp. 239-242). AAAI Press.

Provost, F. and V. Kolluri (1999). A Survey of Methods for Scaling Up Inductive Algorithms. *Data Mining and Knowledge Discovery* 3

Quinlan, J. R. (1979). *Induction Over Large Databases*. (Technical Report STAN-CS-79-739): Comp. Sci. Dept., Stanford Univ.

Quinlan, J. (1983). Learning Efficient Classification Procedures and Their Application to Chess Endgames. In R. Michalski, C. J., and T. Mitchell (Eds.), *Machine Learning: An AI approach*, (pp. 463-482). Los Altos, CA: Morgan Kaufmann.

Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1, (pp. 81-106).

Quinlan, J. R. (1993). C4.5: programs for machine learning. San Mateo, CA: Morgan Kaufmann.

Wu, X, (1995) Knowledge Acquisition from Databases,. Ablex Publishing Corp.

Wu, X. and W. H. Lo (1998). Multi-layer incremental induction. In Proceedings of the Fifth Pacific Rim International Conference on Artificial Intelligence, (pp. 24-32), Springer-Verlag

X Zhu, X Wu and Q Chen (2003) Eliminating Class Noise in Large Datasets. Proceedings of the 20th International Conference on Machine Learning (ICML-2003), Washington D.C., USA, August 21-24, 2003, (pp. 920-927).