Learning and Generalization
with
Bounded Order Rule Sets

A Thesis

Presented to the

Department of Computer Science

Brigham Young University

In Partial Fulfillment

of the Requirement for the Degree

Master of Science

by

Timothy L. Andersen

April 1995

Acceptance Page

This thesis, by Tim Andersen, is accepted in its present form by the Department of Computer Science of Brigham Young University as satisfying the thesis requirement for the degree of Master of Science.

Tony R. Martinez, Committee Chairman

Theodore A. Norman, Committee Member

William A. Barrett, Committee Member

Date

David W. Embley, Graduate Coordinator

Table of Contents

List of Figures

1 Introduction

This thesis deals with the problem of inducing useful rules, or extracting critical, higher-order conjunctions of attributes, from a set of preclassified examples, where little or nothing is known about the underlying functional form of the distribution from which the examples were taken. This problem has been extensively studied in the literature, and belongs to the general area of constructive induction (Elio 1991; Pagallo 1990; Wnek 1994; Raedt 1992; Donoho 1994).

All of the rule-based methods found in the literature use some form of heuristic in order to limit the size of the search space, and the number and types of rule sets considered. The approach taken in this thesis differs in that it does not limit the size of the rule search space. Rather, every possible conjunction of input attributes is considered by the learning algorithm as a potential rule component. In so doing, this thesis is attempting to determine (mainly from an empirical standpoint) how generalization performance is affected when certain areas of the search space are ignored (as is the case with most rule-based learning methods), as compared to when the entire search space is considered. If it can be shown that higher-order rules are not required to approxi mate many real world learning problems, or in other words that many real world classification problems can be solved using only relatively low-order, simple rules, then this would allow for a vast reduction in the size of the search space, and would justify a search bias which favors lower-order, more general rules.

In dealing with the above question, this thesis develops and studies several methods for inducing rules and using them for classification of novel examples. This thesis also uses results obtained with the C4.5 rule-induction method for comparison purposes, and to support the main points of the thesis. In addition, the difficulty of generating optimal rule sets is discussed, where the measure of optimality is the complexity or size of the rule set and/or the degree of predictive accuracy, and the problem of NP-completeness is discussed in relation to these two optimality measures.

2 Machine Learning History and Related Work

2.1 Inductive Learning

Inductive learning is defined to be the process of learning from examples. In general, the method of learning from examples proceeds in two phases. The first phase is called the adaptive phase or training phase. During this phase the learning algorithm is presented preclassified examples from the problem domain. The algorithm attempts to extract information from these examples that will hopefully lead to a general functional mapping of input vectors to output classification (predicted) which minimizes classification error for all of the data points in the problem domain.

The second phase is the non-adaptive phase or execution phase. During this phase the algorithm is presented novel examples which it then attempts to classify correctly using the functional mapping derived during the first phase of the algorithm. In general, the algorithm will not attempt to adjust the functional mapping during this phase (however, there are algorithms which can do this). In the case where the underlying functional form of the true distribution is unknown, the problem of determining how well the functional mapping derived by the learning algorithm agrees with the actual underlying functional mapping is usually approximated in the following manner. A set of preclassified examples, usually called a test set, is withheld during the training phase. This set of examples is then presented after training to see how well the learning algorithm is able to predict the output classifications for the examples in this set.

2.2 The Importance of Inductive Learning

There are two main reasons for the existence of inductive learning systems. The first reason is that there are many potentially profitable classification problems, such as in the field of sociology and economics, where our degree of knowledge concerning the problem consists mostly (or entirely) of examples obtained by observing the environment. In general, the set of observed examples is very small compared to the set of all possible examples, and outside of this relatively small set of observed examples there may be little or no domain knowledge, or the existing domain knowledge may be incomplete. That is, the existing domain knowledge will not be sufficient to provide a satisfactory solution to the problem. In this case, we would like to be able to present these examples to an inductive learning system which could with high probability attempt to discover a satisfactory approximation to the true function which governs the underlying distribution. The solution which the inductive learner derives should then allow us to correctly predict the classification of previously unseen examples from the problem domain.

The second reason for the existence of inductive learning systems is that although we may have the domain knowledge necessary to solve a particular problem, it may be prohibitively expensive or difficult to encode that knowledge in the form of a computer program. For example, Quinlan points out that the standard rate of knowledge acquisition from a human expert is only a few rules per day (Quinlan 1986). Since a typical expert system can require on the order of several hundred to several thousand rules to implement, it is costly to use the interview approach to knowledge acquisition in most cases. However, while extracting domain knowledge from human experts can be relatively expensive and time-consuming, the process of obtaining a set of preclassified examples from the problem domain is generally much easier and less expensive. Thus, the cost of knowledge acquisition and the number of required man-hours can be reduced by using an automated learning system to extract much of the required knowledge directly from a set of preclassified examples from the problem domain.

2.3 History of Inductive Learning

Because of the potential value of such systems, several neural, statistical, and machine learning methods have been proposed as solutions to the problems of inductive learning and pattern recognition. The statistical methods used for pattern recognition and classification include nearest neighbor techniques, bayesian estimation techniques, and parametric and nonparametric clustering techniques (Afifi and Clark 1990; Dasarathy 1991; Fukun 1990; McLachlan 1992; Schalkoff 1992).

Nearest neighbor techniques (Dasarathy 1991) use distance metrics to determine similarity of new patterns to a set of stored exemplars, and classifies new patterns according to the classification of the nearest stored exemplar(s). This makes sense only if nearness in pattern space is a strong indicator of pattern classification, but can suffer when this is not the case, for example when a single critical variable is responsible for determining pattern classification. The nearest neighbor method can also suffer in the presence of noise.

Bayesian techniques are based upon the principle of minimizing the total expected loss. From this standpoint, they theoretically represent the statistically optimum classification technique. However, this optimum can be difficult to achieve in practice, in part due to the problem of having to know or estimate the required *a priori* probabilities and the underlying problem distribution.

Machine learning methods include incremental version space merging (Mitchell 1990; Hirsh 1990), genetic algorithms (Greene and Smith 1993; De Jong, Spears, and Gordon 1993; Booker, Goldberg and Holland 1989), decision trees (Quinlan 1986) and other rule-induction methods (Weiss and Indurkhya 1993; Ozden and Wallace 1993; Chidanand, Damerau and Weiss 1994; Quinlan 1993), and other methods that incorporate meta-knowledge in the learning algorithm (Brazdil 1990).

The version space method (Mitchell 1990; Hirsh 1990) works by storing the set of all possible hypotheses which are consistent with the set of observed examples. It stores this set by keeping track of its boundaries, which can be represented by the single most general hypothesis G consistent with the set of observed examples, and the single most specific hypothesis S consistent with the set of observed examples. The algorithm is said to converge completely when S and G are the same hypothesis ($S = G$). The main weakness of the algorithm is that it requires that there not be any inconsistencies in the training set. Also, the set of training examples is often not sufficient to produce complete convergence. That is, after observing all training instances it is still often the case that *S* is not equal to *G*.

Genetic algorithms (Greene and Smith 1993; De Jong, Spears, and Gordon 1993; Booker, Goldberg and Holland 1989) utilize a heuristic search method based upon population genetics that works relatively well on large search spaces. Initially, the search begins with a few randomly selected individuals, where each individual represents a single point in the search space. The system is then allowed to run for several generations, where at every generation each individual is evaluated according to some fitness function, and based upon that evaluation individuals are allowed to pair and produce offspring through the genetic process of recombination. Individuals with better fitness scores have a greater chance of participating in this phase. After each generation, individuals with the lowest fitness scores are dropped from the population. While genetic algorithms have proven successful at searching extremely large search spaces and finding good solutions, the basic paradigm does not always include generalization capability (predicting classification of novel inputs), and so this has to be added to the basic model by the user.

There are a number of different tree-induction methods, such as ID3, C4.5, CART, and other variants (Quinlan 1986; Quinlan 1987; Quinlan 1993; Brieman et al 1983). One of these methods, the C4.5 tree induction algorithm, is used on the data sets in this thesis for comparison purposes. These methods use a divide and conquer strategy, which works by partitioning the training set into successively smaller chunks until each partition contains training examples which are all, or mostly all, from the same output class. The basic decision tree induction method begins by creating a single node, or root of the tree. A test is introduced at this node which splits the training set into two, or possibly more, parts (Buntine and Niblett 1992). Generally, the algorithm attempts to find a test on a single attribute which fulfills some kind of optimality measure. For each partition created by this split a new node is created, and the examples of each node's corresponding partition are passed to it. A test is then introduced at each newly created node which optimally splits that node's example set or partition. The formula used for partitioning the training set, called the splitting criterion, plays a large role in determining the size and shape of the resultant decision tree. Once the formation of the decision tree is complete it is possible to utilize the tree directly in the generation of production rules (Quinlan 1987; Ozden and Wallace 1993). One of the main problems with decision tree formation is directly related to the divide and conquer strategy which they employ. That is, as the training set is split into progressively smaller parts at each successive node in the tree, the number of training examples remaining in a particular partition may be too small from a statistical standpoint to provide for reliable decision making. Several strategies have been employed to address this issue, including rules for prepruning and post pruning the tree which utilize statistical measures of reliability (Breiman et al 1983, Quinlan 1986, Ozden and Wallace 1993).

Other rule induction systems include SWAP-1, (Weiss and Indurkhya 1991; Chidanand, Damerau, and Weiss 1994) AQ inductive algorithms (Michalski 1983, 1986; Wnek 1994), RULES (Pham and Aksoy 1995), BBG (Van Horn and Martinez 1993) and CN2 (Clark and Niblett 1990).

Neural Networks are designed to use simple algorithms which are applied over and over again in parallel until the network has successfully learned the problem. In general, neural networks do not require a priori knowledge about problems which they are intended to solve, but rather learn about the problem through the presentation of preclassified examples from the problem domain. Processing in the net has inherent parallelism, and most nets are capable of varying degrees of self organization (Beale 1990; Fahlman and Lebiere 1991; McCLelland and Rumelhart 1986; Redding, Kowalczyk and Downs 1993; Romaniuk and Hall 1993; Schalkoff 1992).

The ASOCS (Martinez 1986; Martinez and Vidal 1988; Martinez 1990; Martinez and Campbell 1991; Martinez 1991) learning systems are incremental learners. The different ASOCS architectures (AA1, AA2, AA3) are connectionist implementations of boolean logic. As such, they have a number of advantageous features. The various ASOCS architectures are fast, learn incrementally, do not require that variables be predefined, and guarantee learning of arbitrary boolean mappings. Unfortunately, some of the ASOCS architectures have been shown to have poor generalization capabilities. The PASOCS (Priority ASOCS) model has been developed to address some of the shortcomings of the earlier models.

3 Rule Induction

3.1 The Focus of this Thesis

The focus of this thesis is on the problem of inducing rules which can be used for classification or decision making through direct analysis of preclassified examples from the problem domain. The goals of this thesis are two-fold in this area.

- First, this thesis discusses the computational complexity of solving some of the main problems with which rule induction algorithms are confronted, such as generating rules, minimizing rule set complexity, and maximizing rule set accuracy. In conjunction with this discussion, this thesis develops a formal proof of NP-Completeness concerning the problem of generating minimum size (simple) rule sets, and it also discusses the complexity of maximizing rule set accuracy.
- The second and main goal of this thesis is to demonstrate the benefit (or lack thereof) of utilizing higher-order rules for classification and decision making tasks.

3.2 Justification for Rule Induction

Since the focus of this thesis is on rule-induction, one might ask why we should attempt to develop such systems when there are other learning systems, such as artificial neural networks, which appear to perform well on a variety of complex learning problems (Mooney et al 1989; Weiss and Kapouleas 1989). Three of the main reasons for the development and use of rule induction systems are:

- Explanatory capability.
- Similarity to human reasoning.
- We can always improve on current models.

One of the main reason for generating/using rules is that they provide us with an easy to understand explanation (for a human) as to why the decision was made. There are many areas in which we not only want to know which decision to make, but we would also like to know why we should make that decision. For example, if a computer rejects a loan application, the persons making the application may wish to know why their loan was rejected. This information would help them to rectify the problem with their financial situation which led to the rejection.

Another reason for using rules is that it appears that Humans often use rules in their reasoning processes for certain types of situations, such as certain pattern classification tasks. For example, a human may classify an animal that has not been seen before as a predator if the animal has large, sharp teeth. In this case, the rule for classification is "if an animal has large, sharp teeth, it is a predator". In addition, a human might associate some form of probability with the rule, such as, "if an animal has large, sharp teeth, it is usually a predator". It is therefore desirable to build learning systems which generate these types of easily understood, and "human usable" rules. By examining the rules generated by such systems it is often possible for humans to obtain a better understanding of various classification and decision problems. Also, it is simpler to augment such a system by directly adding the rules which humans have learned via experience than it is to augment a system where the reasons for making certain decisions are less apparent, such as with a back-propagation neural network.

A final reason for the development of a variety of learning systems, including rule induction systems, is that it would seem unlikely that any one learning system or method of learning will outperform all others on all learning tasks of interest (Schaffer). While there are many inductive learning systems which perform well for the most part, there is always room for improvement. Also, there has been much work done on the use of multiple styles of generalization which has shown that a great deal of benefit (in the form of increased classification accuracy) can be obtained by combining different types of learning systems into a single, unified learning system that uses the results of each individual classifier in order to make a final decision (Wolpert 1993; Battiti and Colla 1994). It has even been suggested that for this type of system one should choose classifiers which are as different as possible in their learning method and in the types of errors which they produce (Wolpert 1993). Thus, the development of many different types of learning algorithms is a useful and necessary element of the field of inductive learning.

3.3 Issues with Rule Induction Systems

With any rule induction system we must deal with the following problems:

- Generation of individual candidate rules.
- Selection of the final set of rules which will be used in classification.
- Organization of the rules for classification.

The creation of individual rules is essentially a search through the space of all possible rules. Since the set of all possible rules is exponential in size, the search space generally must be limited in some fashion. During the rule generation process, some learning algorithms seek to discover the single most optimal rule at each successive step. That is, given a set of rules F_t, the learning algorithm seeks to add a rule f to F_t, producing a set of rules F_{t+1} which maximizes the classification performance of the set of rules. Since the size of the set of all possible rules from which f must be selected is exponential in the number of possible attributes, it would appear that the problem of selecting the single most optimal rule could be very difficult.

Another difficulty which the learning algorithm must deal with is the selection of the final set of rules, and their organization for classification. The method used to organize the rules often affects which rules will be selected during this phase. Two possible methods for organizing the rules are ordering the rules into a list and using the first matching rule in the list to classify each incoming example, or allowing each matching rule to vote for a particular classification for each incoming example. Obviously there are many different ways in which rules could be used for classification. But whatever the method used for organizing the rules and using them for classification, it is preferable that the learning algorithm select a set of rules which is optimal in at least two ways. First, we would like the learning algorithm to select the set of rules which maximizes classification performance. Second, we would like the learning algorithm to select the simplest set of rules, all other things being equal.

The first measure of optimality (selecting the set of rules which maximizes classification performance) at first glance seems like it ought to be simple, and it can be if the maximum classification performance for a given set of rules is 100 percent. However, if the maximum possible classification performance for a given set of rules is less than 100 percent, then the problem of determining the set of rules which maximizes classification performance becomes much more difficult. This is discussed further in section 5.3.

Achieving the second measure of optimality (selecting the simplest set of rules) is proven to be NP-Complete in section 5.2 of this thesis, and is likely to be even more difficult to approximate than the problem of selecting the set of rules which maximizes classification performance. This thesis uses a straightforward transformation from minimum set cover to show that this problem is NP-Complete. This transformation is similar to that used by (Van Horn 1994) for the minimum feature set problem. Due to the nature of this transformation, it can also be shown that the minimum rule set problem is at least as hard to approximate as minimum set cover, for which no known approximation algorithm comes within a constant factor of optimal.

While it is unlikely that any rule induction algorithm can guarantee optimum solutions to the problems of rule set simplicity and rule set accuracy (unless P = NP), it does not preclude a rule induction algorithm from generating near-optimal, or satisfactory solutions to these problems.

3.4 The Approach Used in this Thesis

This thesis utilizes a learning method which extracts conjunctions of attribute values from an existing data set through direct analysis of the examples contained in that data set. Rules are then created by associating each selected attribute conjunction with an appropriate output classification. The rules are then used to classify novel examples from the problem domain through a number of different methods which are explained in chapter 6. Since the number of potential rules for a given data set is exponential in the number of inputs it is generally not feasible to search the entire attribute space for critical attribute conjunctions. The method proposed in this paper to prune the search space is to limit the order of considered attribute conjunctions or rules, where the order is said to be the number of terms in the conjunction. In other words, the approach used in this thesis is to bound the search through the attribute space so that it encompass only lower-order rules.

For example, consider a mushroom classification problem, where the inputs to the problem are stalk length (SL), stalk color (SC), and cap color (CC). Figure 3.1 lists a possible set of rules for this problem. These rules have an equivalent DNF representation, which is shown in figure 3.2. Rule f_1 is a 3rd order rule, rule f_2 is a 2nd order rule, and the remaining rules are all 1st order rules.

	SL	SC	CC		Outcome
f_1	SL: long	SC: yellow	CC: orange	-->	edible
f_2	SL: long	SC: *	CC: orange	-->	death
f_3	SL: long	SC: *	CC: *	-->	edible
f_4	SL:*	SC: red	CC: *	-->	edible
f_5	SL:*	SC: *	CC: orange	-->	death

Figure 3.1. Rule set for mushroom classification.

f_1	SL: long $\wedge$ SC: yellow $\wedge$ CC: orange	-->	edible
f_2	SL: long $\wedge$ CC: orange	-->	death
f_3	SL: long	-->	edible
f_4	SC: red	-->	edible
f_5	CC: orange	-->	death

Figure 3.2. DNF representation.

There are several reasons why it is likely that the restriction of the search space to lower-order rules will not prove to be an unreasonable approach in the general case. The first is that it is not feasible to look at the whole search space for many problems of interest (since it is exponential in size), so some form of heuristic must be implemented to restrict the size of the search space. Since higher-order rules appear to be particularly hard to generate due to the greater number of terms which their attribute conjunctions contain, they offer themselves as ready candidates for exclusion. Also, if the search space is limited to rules of order j or less, then the size of the search space is polynomially bounded by j, a much more manageable size.

Another reason for favoring lower-order DNF rules is that they are by nature more general. That is, in general lower order rules cover more of the input space (they match more of the possible examples in the problem domain) than higher order rules. One of the main goals in any learning problem is to be able to accurately predict classification for examples which have not been seen before. Since lower order rules have a higher probability of matching a novel example due to the fact that they cover more of the potential input space, it would appear that in general they will be matched more often, and will therefore be more useful in predicting classification for novel examples.

The second advantage of lower-order rules is that they tend to occur more often in the training set than higher order rules. Since lower order rules will, in general, occur more often, it is easier to determine if the correlation's they exhibit with particular pattern classes are valid. If a higher order rule only occurs once or twice in a relatively large training set, it is impossible to tell if the occurrence of that rule is merely a statistical aberration, or a valid predictor of pattern classification. In summary, high-order rules tend to occur less often in the training set (and will therefore tend to be less useful for generalization purposes), are more susceptible to noise, and the amount of search required to discover them may be very large. Indeed, it is expected that the use of higher-order rules will often be detrimental to the generalization performance.

The major goal of this paper is to show whether or not satisfactory learning can be accomplished on real world problem domains if the order of stored rules is bounded. If this can be shown, then it would allow for a large reduction in the size of the search space required to find critical rules. In order to do this, this paper first proposes methods which utilize bounded order rules to perform learning and generalization, then applies these methods to several data sets and analyzes the results. This paper attempts to demonstrate how limiting the order of stored rules affects the ability of the proposed methods to learn and generalize correctly. In so doing, an assessment of the viability of learning using only low-order rules is made.

4 Terms and Definitions

The following terms and definitions are used throughout the remainder of this thesis. First, the idea of the problem domain is defined. Then the definition of an example from the problem domain is given. Next, the definition of a rule is given, and finally, definitions which relate rules to examples are given.

Definition 4.1 D is defined to be the problem domain (the set of all possible data points).

Definition 4.2 O is defined to be the finite set of states into which the individual elements of D may be classified.

Definition 4.3 For $1 \leq i \leq |O|$, the ith class of O is written as o_i.

Definition 4.4 Each data point $d \in D$ is composed of two parts (in the following a dot notation is used, for example, $d.\boldsymbol{a}$ is written to indicate d's input vector $\boldsymbol{a}$):

a) $d.\boldsymbol{a} = a_1 a_2 \ldots a_m$, an input vector of m nominally valued attributes.

b) $d.\boldsymbol{c} = d.c_1 c_2 \ldots c_{|O|}$, a probability vector of real values on the range [0,1]. The length of the vector is equal to the number of possible output classifications for any pattern in D. Note that while this information is said to exist, it is not necessary that it be known (in general, this information is not known).

Definition 4.5 Let $d \in D$. Then the probability that the classification of d is o_i is written as $d.c_i$. For each data point $d \in D$ we define the sum

$$\sum_{i=1}^{|O|} d.c_i = 1.$$

Definition 4.6 Given a problem domain D, m is defined to be the number of nominally valued attributes for any input vector $d.\boldsymbol{a}$, where $d \in D$. m is defined to be constant for any given problem domain.

Definition 4.7 Given a data point $d \in D$, the individual a_i's of d are commonly called attributes or features. For the purposes of this thesis, the number of discrete values which any given attribute a_i of d may take is constant and finite. The finite set of states which a_i may take is written as S_i. Furthermore, each S_i always contains the * state (the asterisk is a special "don't care" state, the use of which is explained in definition 4.18). That is, $* \in S_i$ for all i, $1 \leq i \leq m$.

Definition 4.8 Let d, q be any two data points from D ($d, q \in \boldsymbol{D}$). If $d.\boldsymbol{a} = q.\boldsymbol{a}$ then this implies that $d = q$ (thus if $d.\boldsymbol{a} = q.\boldsymbol{a}$ then $d.\boldsymbol{a} = q.\boldsymbol{a}$).

Definition 4.9 A Data Set E is defined to be a finite set of preclassified data points (examples or instances) from a particular problem domain D.

Definition 4.10 Let $e \in E$. Then e is an instance or example from the data set E. For $1 \leq j \leq |E|$ we write e_j to indicate the jth example in E (where the elements of E have been ordered arbitrarily from 1 to $|E|$).

Definition 4.11 Each example $e \in E$ is composed of three parts:

a) $e.\boldsymbol{a} = a_1 a_2 \ldots a_m$, an input vector of m nominally valued attributes.

b) $e.o = o_i$, an output classification which is equal to some $o_i \in O$.

c) $e.id$ = an integer valued identification label. It is required that for all e, $y \in E$ that $e.id = y.id$ if and only if $e = y$. Further, for all $e \in E$, $1 \le e.id \le |E|$.

Definition 4.12 Given E, a finite set of preclassified data points from a problem domain D, then the following is true:

$$(\forall e)(e \in E \rightarrow (\exists o_i)(o_i \in O \wedge e.o = o_i) \wedge (\exists d)(d \in D \wedge d.\mathbf{a} = e.\mathbf{a})).$$

Definition 4.13 A rule f of a data set is composed of 3 basic parts:

a) $f.\boldsymbol{a} = a_1 a_2 \ldots a_m$, an input vector of m nominally valued attributes.

b) $f.o = o_i$, a nominally valued output classification with which the rule is associated.

c) $f.p$, a relative strength metric which indicates the degree of correlation that the rule has with the output $f.o$ in relation to other rules.

Definition 4.14 We write F to indicate the current set of all rules for a data set.

Definition 4.15 Let f_j be the jth rule of F (where the rules in F have been arbitrarily ordered from 1 to $|F|$).

Definition 4.16 A set of rules F is said to be ***defined*** on a problem domain D iff the following is true:

$$(\forall f)(f \in F \rightarrow (f.o \in O) \wedge (\forall j)((1 \le i \le m) \rightarrow f.a_i \in S_i))$$

Definition 4.17 Let $f \in F$. Then the ***order*** of rule f is defined to be the number of attributes of $f.\boldsymbol{a}$ which are not equal to '*' (don't care).

Definition 4.18 We define a matching function for attributes of rules and examples called *a_match*(*a, b*) (attribute match) as follows:

define *a_match*(*a, b*) such that

$a\_match(a, \mathrm{b}) = 1$ if $((a = b) \vee (a = *) \vee (b = *))$,

else $a\_match(a, b) = 0$.

Definition 4.19 A rule $f \in F$ is said to match an example $e \in E$ iff

$$\sum_{i=1}^{m} a\_match(f.a_i, e.a_i) = m.$$

We thus define a matching predicate for rules and examples r_*match*(*f*, *e*) (rule match) as follows:

$r\_match(f, e) = \text{True if } \sum_{i=1}^{m} a\_match(f.a_i, e.a_i) = m,$

otherwise $r\_match(F, E) = \text{False}$.

Definition 4.20 A rule f is said to be a proper subset of a rule y iff

$$(\exists i)((1 \le i \le m) \wedge (f.a_i \ne y.a_i)) \wedge (\forall i)(((1 \le i \le m) \wedge (f.a_i \ne y.a_i)) \rightarrow (f.a_i = *)).$$

In this case we write $f \subset y$.

Definition 4.21 Let $f, z \in F$, where F is a set of rules defined on a problem domain D. Then f is said to be ***distinct*** from z iff

$(\exists i)((1 \le i \le m) \wedge (f.a_i \ne z.a_i)) \vee (f.o \ne z.o)$.

This implies that the value of $f.p$ (where $f.p$ is the relative strength metric of the rule) does not affect the process of determining whether or not two rules are distinct.

Definition 4.22 It is required that all rules in the set of rules F be pairwise distinct.

Definition 4.23 Given a set of preclassified examples E from a problem domain D, then we say that a set of rules F is ***derived*** or ***derivable*** from E iff the elements of F are all pairwise distinct and

$$(\forall f)(f \in F \rightarrow (f \text{ is } \textit{defined} \text{ on } D) \wedge (\exists e)(e \in E \wedge \text{r\_}\textit{match}(f,e)).$$

Definition 4.24 A rule f is said to be ***positive*** in relation to an example e iff $f.o = e.o$. Otherwise the rule is said to be ***negative*** in relation to the example.

Definition 4.25 Let E be a data set of preclassified examples from a problem domain D. Then let T and V be two non-empty sets which form a partition of E. In other words let T and V be two sets such that $T \subset E \wedge V \subset E \wedge T \cup V = E \wedge T \cap V = \varnothing$. Then T is called the **training set** and V is called the **test set** (verification set).

5 On The Complexity of Learning DNF Rules

The usual approach to learning from examples is as follows: given a data set *D* containing preclassified examples from a particular problem domain, randomly partition the data set into two (not necessarily equal) parts, a training set *T* and a test set *V*. Run the learning algorithm in adaptive mode using only the examples in the training set *T*. Next, run the learning algorithm in nonadaptive mode on the test set *V* to determine generalization accuracy.

During the adaptive phase, the learning algorithm attempts to extract critical rules from the examples in the training set. The hope is that the training set contains the information necessary to extract these rules, and does not contain invalid information which could lead to the extraction of misleading rules. In spite of the fact that this ideal situation is usually not realized, it is generally the case that the data set will contain enough valid information to come up with a reasonably good hypothesis.

Obviously, since the number of possible attribute combinations is exponential in the number of inputs, we will not be able to look at all of the possible combinations for every learning problem of interest. This means that in the search for potentially interesting rules a realistic learning algorithm will have to ignore certain, potentially vast areas of the search space, in particular with problems that have large numbers of input attributes.

The first question we may ask is can we guarantee that ignoring certain areas of the search space will not adversely affect the solution generated by our learning algorithm? If the answer to this questions is no, then another question we may wish to ask is that if areas of the search space are ignored, how harmful will this be to the solution? This thesis attempts to address these issues in this chapter.

There are two main measures of optimality which are used by most rule induction systems to constrain the set of acceptable hypotheses. The first of these is classification accuracy. That is, we would like the learning algorithm to generate the rule set which maximizes classification accuracy. The second measure of optimality is rule set simplicity. In other words, if there are two (or more) distinct rule sets that exhibit the same degree of classification accuracy, we would like the learning algorithm to pick the simplest such rule set.

This section discusses the problem of rule induction, and the difficulty of generating optimal rule sets by first discussing the size of the hypothesis space in section 5.1, which turns out to be very large indeed (a super exponential). The difficulty of finding the simplest (or smallest) rule set which solves a particular learning problem is discussed in section 5.2, and a proof of the NP-Completeness of this problem is given. Finally, the problem of optimizing rule set accuracy is discussed in section 5.3.

5.1 The Vastness of the Hypothesis Space

The functional mapping of input vectors to output classification derived by the learning algorithm is called the hypothesis. Different learning algorithms may have vastly different ways of representing hypotheses. For example, many rule induction algorithms represent hypotheses with sets of rules that are in disjunctive normal form, while neural networks represent hypotheses with networks of nodes that are connected via weighted links. In general, the set of all possible hypotheses on a particular problem domain which are representable by a given learning algorithm is extremely large. This set is usually so large, that it simply is not feasible to check every possible hypothesis in order to determine which is the best.

For example, let us consider the method used for representing hypotheses in this paper. Let R be the set of all possible rules, then the size of the set of all possible rules is exponential in the number of input attributes, and can be calculated from the following formula:

$$|R| = \prod_{i=1}^{m} |S_i| .$$

Furthermore, each and every rule in the set of all possible rules is available for selection by the learning algorithm as a member of the final set of rules which will be used for decision making. The number of different possible rule sets which the learning algorithm can generate is thus the power set of the set of all possible rules. The size of this set is then $2^{|R|}$. Notice that this is already a very large number (super-exponential). Furthermore, let us assume that the learning algorithm organizes the selected rules into an ordered list (in this case the first matching rule is used to classify novel examples). For any given set of rules F, the number of different ways to order the rules is $|F|!$. So, the actual number of different hypotheses theoretically expressible by one of the simpler learning algorithms used in this thesis turns out to be (note that the summation has a super-exponential number of terms, each term being a factorial):

$$\sum_{i=1}^{2^{|R|}} |F_i|!,$$

where F_i represents the ith possible rule set (the set of all possible rule sets having been ordered from 1 to $2^{|R|}$). This is a number that grows very quickly with the number of input attributes, and with even a relatively small number of input attributes the number of different hypotheses theoretically expressible by the learning algorithm can quickly outpace the number of atoms in the galaxy.

Simply demonstrating the size of the hypothesis space for this approach does not prove how difficult it might be to find the "best" hypothesis. Indeed, it might be hoped that in spite of the extremely large hypothesis space, there might exist a polynomial time algorithm which can deterministically find the optimum hypothesis, perhaps by implementing some type of greedy search mechanism.

5.2 Minimum Rule Set

In the problem of learning from a set of training examples T, the learning algorithm can be thought of as selecting a potential hypothesis from the set of all possible hypotheses which would correctly classify (total error < ε) the elements of T. In general, when given a choice between two or more hypotheses which exhibit the same degree of classification accuracy, the most preferable hypotheses is the one which is the simplest, and so we would like the learning algorithm to select the simplest such hypothesis.

This requires the definition of some reasonable measure of simplicity, which will of course depend upon the form that the explanations may take. If the hypothesis takes the form of a set of rules, where the set of rules is selected from all possible sets of rules, then one measure of simplicity which could be used would be the size of the set of rules selected. In other words, assuming that F_1, F_2 are sets of rules, then we say that F_1 is simpler than F_2 if $|F_1| < |F_2|$. This is obviously true if F_1 is a proper subset of F_2. The weakness of this definition is that F_1 might not really be simpler than $F2$ in the case where $F_1 \not\subset F_2$, even though $|F_1| < |F_2|$. This is true since the simplicity measure is not taking into account the complexity of the individual rules in F_1 and F_2. However, the strength of this measure of simplicity is that it is a relatively straightforward and intuitive notion that is easily measured. For the proof of NP-completeness in this thesis we consider F_1 to be simpler than F_2 if $|F_1| < |F_2|$, even if $F_1 \not\subset F_2$. However, we informally extend the proof to the problem of determining the simplest set of rules when individual rules have an associated complexity, which can also be seen to be NP-Complete in the general case.

More formally, suppose that we have a set of preclassified examples T from a problem domain D, and that we have a set of rules F which we may use to classify these examples. Suppose further that we have a generalization algorithm *gen* which takes as input examples from T and a set of rules F, and classifies elements of T based upon the rules in F. Given an example e, the output of the function $gen(e,F)$ is said to be defined if there is an $f \in F$ such that f matches e, otherwise the output of $gen(e,F)$ is undefined (since in this case it would not have any matching rules with which to classify e). Other than this requirement, there need not be any other restrictions on how *gen* may use the rules in F to classify the examples in T.

Given this information, one might ask the question, is there some subset $F' \subset F$ such that the output of $gen(e,F')$ is correct (the error of classification is minimized) for all $e \in T$? If so, what is the smallest such subset of F? Alternately, one might ask the (weaker) question, is there some subset $F' \subset F$ such that the output of $gen(e,F')$ is *defined* for all $e \in T$ (in this case we don't care if the predicted classification is correct, only that <u>some</u> classification is predicted)? And if so, what is the smallest such subset of F? It is shown in this thesis that both of these questions are NP-complete.

5.2.1 Generalization and Learning Algorithms

There are an infinite number of ways that rules could be used to predict classification. For the purposes of the proof given in this thesis, it is not necessary that any specific generalization method for resolving conflicts between matched rules be defined. It is only required (for the proof) that for each example there exists at least one rule which matches the example and correctly predicts the classification of the example.

Although no specific generalization method is required, for the sake of simplicity (and the purposes of illustration), we define a generalization scheme which uses a voting method that sums the metrics of all matched rules for the respective possible output classifications, then chooses the output classification with the greatest sum. Again, the proof of NP-completeness for the problem of minimizing the size of the rule set does not depend upon this form of generalization. It is merely used for convenience.

The formal generalization algorithm is given below in figure 5.1.

Given

a set E of examples from a problem domain D,
a set F' of rules defined on D,

then

for each $e \in E$, choose the predicted output classification o_i which maximizes the summation

$$\sum_{f \in F' \wedge f\_match(f,e) \wedge f.o=o_i} f.p \quad .$$

If there exists an $e \in E$ such that there is no $f \in F'$ which matches e, then the output of the generalization algorithm is said to be undefined. Otherwise the output is said to be defined (ties are resolved arbitrarily).

Figure 5.1. Generalization algorithm

The proof is then defined for any learning algorithm which satisfies the following:

Given

a set T of preclassified training examples,

a set F of rules derived from T,

return a set of rules $F' \subseteq F$ such that

$$(\forall e)(e \in E \rightarrow gen(e, F') = e.o).$$

(Alternately, in the case where there are inconsistencies in the set of examples, we would choose the set of rules F' which minimizes the error of misclassification).

This implies that the learning algorithm returns a set of rules F' such that the output of the generalization algorithm is always defined for each example $e \in T$. Further, for each example $e \in T$, there will be a rule $f \in F'$ which matches e and correctly predicts e's output classification. In other words, the following is true of the set of rules F' returned by the learning algorithm:

$$(\forall e)(\exists f)(e \in T \rightarrow f \in F' \wedge r\_match(f,e) \wedge e.o = f.o).$$

5.2.2 Proof of NP-Completeness

We first begin by defining the problem of minimum set cover, called MIN SC.

MINIMUM SET COVER (MIN SC)

INSTANCE: = (X, C, k), a collection C of subsets of a set X, positive integer k.

THE PROBLEM: find some $C' \subseteq C$ such that $|C'| \leq k$ and

$$\bigcup_{C_i \in C'} C_i = X.$$

This problem is known to be NP-complete.

We now formally define the problem of minimum rule set, called MIN RS.

MINIMUM RULE SET (MIN RS)

INSTANCE: = (T,F,k), a set of training examples T, a set of rules F, and a positive integer k.

THE PROBLEM: Find some $F' \subseteq F$ such that $|F'| \leq$ k and

$$(\forall e)(e \in T \rightarrow gen(e, F') = e.o).$$

Note that, as stated above, this means the following condition (as a minimum requirement) must hold:

$$(\forall e)(\exists f)(e \in T \rightarrow f \in F' \wedge r\_match(f,e) \wedge e.o = f.o).$$

Lemma 1 *MIN RS* $\in$ *NP*:

Proof. Given an instance (T,F,k) of MIN RS, a non-deterministic algorithm need only guess a subset F' of F of size $\leq k$, and then run the generalization algorithm on each example in T to determine if the condition $(\forall e)(e \in T \rightarrow gen(e, F') = e.o)$ holds true. Assuming that the function *gen* is bounded by a polynomial, then the time bound for this process will also be a polynomial. Thus MIN RS $\in$ NP.

Lemma 2 *There is a function* μ *which maps instances of MIN SC to instances of MIN RS which has the following properties:*

1. *The function* μ *is computable in polynomial time.*

2. *If there is cover* $C' \subseteq C$ *for X such that* $|C'| \leq k$, *then there is a rule set* $F' \subseteq F$ *such that* $|F'| \leq k+1$ *and* $(\forall e)(e \in T \rightarrow gen(e, F') = e.o)$

3. *If there is a rule set* $F' \subseteq F$ *such that* $|F'| \leq k+1$ *and* $(\forall e)(e \in T \rightarrow gen(e, F') = e.o)$, *then there is a cover* $C' \subseteq C$ *for X such that* $|C'| \leq k$.

Proof. (In the following proof we assume that C contains a cover for X). Given an instance (X,C,k) of MIN SC we define the following mapping function μ to an instance (T,F,k) of MIN RS:

Let $n = |X|+1$, $m = |C|+1$. Enumerate the instances of X and C. Then $X = \{x_1, x_2, . . . ,x_{n-1}\}$, and $C = \{C_1, C_2, . . . ,C_{m-1}\}$.

Let $T = \{e_1, e_2, . . ., e_n\}$, where for $1 \le j < n$, e_j in T corresponds to element x_j in X. For each $e_j \in T$, where $1 \le j < n$, let

$e_j.\boldsymbol{a} = a_1a_2 . . . a_m$, where

$a_l = 1$ if $x_j \in C_l$

else $a_l = 0$.

$e_j.o = 1$.

For $j = n$ let

$e_j.\boldsymbol{a} = a_1a_2 . . . a_m$, where $a_l = 0$ for all a_l in $e_j.\boldsymbol{a}$

$e_j.o = 0$.

Let $F = \{f_1, f_2, . . ., f_m\}$, where for $1 \le j < m$, f_j in F corresponds to set C_j in C (in other words, there is exactly one f_j in F for each C_j in C). For each $f_j \in F$, where $1 \le j < m$, let

$f_j.\boldsymbol{a} = a_1a_2 . . . a_m$, where

$a_l = 1$ if $j=l$

else $a_l = *$.

$f_j.o = 1$

For $j = m$ let

$f_j.\boldsymbol{a} = a_1a_2 \ldots a_m$, where $a_l = *$ for all a_l in $f_j.\boldsymbol{a}$

$f_j.o = 0$.

We first show that the mapping function fulfills the first condition. The time required to create the set of training examples T is bounded above by $O(mn^2)$. There are n examples in T. Each example has a vector of length m. For each element of the vector, the time required to determine if $x_j \in C_k$ is at most n-1, where n-1 is the maximum size of any subset of X (since $n = |X|+1$). Thus an upper bound for the creation of T is $O(mn^2)$.

The upper bound for the creation of F is $O(m^2)$. There are m rules in F. Each rule in F has a vector of length m. The operation $j=k$ takes time O(1). Therefore we have the upper bound of $O(m^2)$ to create F.

From above we have $O(m^2 + mn^2)$ as the upper time-bound for the mapping function. This upper bound thus fulfills the first condition that the mapping can be computed in polynomial time.

We will now show that the mapping fulfills the second requirement. That is, it has the property that if there is cover $C' \subseteq C$ such that $|C'| \leq k$, then there is a rule set $F' \subseteq F$ such that $|F'| \leq k+1$ and $(\forall e)(e \in T \rightarrow gen(e, F') = e.o)$. Assume that there is a cover C' such that $C' \subseteq C$ and $|C'| \leq k$. Select $F' \subseteq F$ where $F' = \{f_j \mid C_j \in C'\} \cup \{f_m\}$, then the size of $F' \leq k+1$. For all f_j, where $j < m$, let $f_j.p = 2$. Let $f_m.p = 1$.

From the construction we know that the only rule which matches example e_n is rule f_m (in other words, $\left\{f \mid f \in F \wedge f\_match(f_j, e_n)\right\} = \{f_m\}$). Since $f_m.o = 0$ and $f_m.p = 1$ we have

$$\sum^{f \in F' \wedge r\_match(f, e_n) \wedge f.o=0} f.p = 1 \text{ and}$$

$$\sum^{f \in F' \wedge r\_match(f, e_n) \wedge f.o=1} f.p = 0,$$

which implies that $gen(e_n, F') = e_n.o = 0$.

We also know from the construction that for all $1 \leq j < m$, $\left\{x_i \middle| e_i \in T \wedge r\_match(f_j, e_i)\right\} = C_j$. Therefore, since C' is a cover for X, and for each $C_j \in C'$ we have $f_j \in F'$, we know that

$$\bigcup_{f_j \in F', j \neq m} \{x_i \mid e_i \in T \wedge r\_match(f_j, e_i)\} = X.$$

Therefore,

$$\bigcup_{F_j \in F', j \neq m} \{e_i \mid e_i \in T \wedge r\_match(f_j, e_i)\} = T - e_n.$$

This means that for each example $e \in T$ (with the exception of example e_n) there is at least one rule $f \in F'$ with $f.o = 1$ and $f.p = 2$ which matches e. In addition, we know that rule f_m matches all $e \in T$. Therefore the following equations hold true for all $e \in T$, where $e \neq e_n$.

$$\sum^{f \in F' \wedge r\_match(f,e) \wedge f.o=0} f.p \quad = 1$$

$$\sum^{f \in F' \wedge r\_match(f,e) \wedge f.o=1} f.p \quad \geq 2,$$

which implies that for all $e \in T$ where $e \neq e_n$, $gen(e, F') = 1$, thus fulfilling the second property.

Lastly, we show that the mapping has the property that if there is a rule set $F' \subseteq F$ such that $|F'| \leq k+1$ and

$$(\forall e)(e \in T \rightarrow gen(e, F') = e.o),$$

then there is a cover $C' \subseteq C$ such that $|C'| \leq k$.

Assume that there is a set of rules $F' \subseteq F$ such that $|F'| \leq k+1$ and $(\forall e)(e \in T \rightarrow gen(e, F') = e.o)$. Since rule f_m is the only rule which matches example e_n, this rule must be assigned a positive metric, and be included in the set F' in order for example e_n to be classified correctly. Let $f_m \in F'$ and $f_m.p = x$, where x is a positive real number. Since we have assumed that all examples from T are classified correctly, we know that for all $e \in T$, where $e \neq e_n$, there must be some $f_j \in F'$ which matches e, and such that $f_j.o = 1$ and $f_j.p > x$. We also know that in this case $j \neq m$ (since $f_m.o = 0$).

This implies that for each such f_j there is a corresponding $C_j \in C$ (since rule f_j in F, $j < m$, corresponds to set C_j in C from the construction). Let $C' = \left\{C_j | f_j \in F' \wedge j \neq m\right\}$. From the above paragraph we know that

$$(\forall i)(\exists j)(e_i \in T \wedge i \neq n \rightarrow f_j \in F' \wedge j \neq m \wedge r\_match(f_j, e_i)).$$

Then this implies that

$$\left\{e_i \,|\, e_i \in T \wedge f_j \in F' \wedge j \neq m \wedge r\_match(f_j, e_i)\right\} = T - \{e_n\},$$

and thus

$$\left\{x_i \,|\, e_i \in E \wedge f_j \in F' \wedge j \neq m \wedge r\_match(f_j, e_i)\right\} = X.$$

Furthermore, from the construction we have that for any j such that $1 \geq j \geq m$,

$$C_j = \left\{x_i | e_i \in T \wedge r\_match(f_j, e_i)\right\}.$$

Therefore,

$$\bigcup_{C_j \in C'} C_j = \left\{ x_i \,|\, e_i \in T \wedge f_j \in F' \wedge j \neq m \wedge r\_match(f_j, e_i) \right\} = X .$$

Therefore C' is a cover for X of size $\leq k$.

Theorem 1 MIN RS is NP-complete.

Proof. By Lemma 1, we know that MIN RS is in NP. By lemma 2, we know that there is a mapping function which maps instances from a known NP-complete problem (MIN SC) to instances of MIN RS which can be computed in polynomial time. The mapping has the property that if a yes answer is obtained for MIN SC, then a yes answer can be obtained for MIN RS. Conversely, if a yes answer is obtained for MIN RS, then a yes answer can be obtained for MIN SC. Therefore, MIN RS is NP-complete.

5.2.3 Discussion on the NP-completeness of MIN RS

The proof above can also be used to show that the question of determining the smallest set of rules $F' \subseteq F$ such that

$$(\forall e)(\exists f)(e \in T \rightarrow f \in F' \wedge r\_match(f, e) \wedge e.o = f.o)$$

is NP-complete. The motivation for this question is that we assume that *gen* does not have a chance of correctly predicting the classification for an example *e* if there isn't at least one rule in F' with the appropriate output which matches *e*, and so we are asking, "can we find the smallest set of rules which has even a remote chance of correctly classifying examples from *T*?" Intuitively, this question would seem easier to answer than that of MIN RS, since we no longer have the requirement to check the output of the generalization algorithm. However, this question is also NP-complete.

With some modification, the proof can also be extended to show that the even weaker question of determining the smallest set of rules $F' \subseteq F$ such that

$$(\forall e)(\exists f)(e \in T \rightarrow f \in F' \wedge r\_match(f,e))$$

is NP-complete. We know that the output of the generalization algorithm is undefined for an example *e* if it does not have any rules which match *e*, so as a minimum requirement we know that for each example *e* in *T* we must have at least one rule in *F'* which matches *e*. Otherwise the output of *gen* is guaranteed to be incorrect since it will be undefined. Thus, the NP-completeness of this question implies the NP-completeness of MIN RS, since MIN RS contains this problem as a sub-problem (in other words, there are problems where we will have to be able to answer this question in order to be able to answer MIN RS).

The actual output of the generalization algorithm *gen* is not referenced in either of the two NP-complete questions stated above. This implies that no matter how the generalization algorithm utilizes rules to predict classification, the general question of MIN RS is NP-complete regardless of the exact form of the generalization algorithm. This holds as long as we require that $(\forall e)(\exists f)(e \in T \rightarrow f \in F' \wedge r\_match(f,e))$ in order for the output of *gen* to have a chance of being correct.

The NP-completeness of MIN RS is not dependent upon the format used for rules or examples given in this paper. For any rule format, the set of examples that a given rule matches is always a subset of the total set of examples. We can define this set as follows for any given rule *f*, where *r_match* is a deterministic matching function:

$$\{e \mid e \in T \wedge r\_match(f,e)\} \subseteq T.$$

Thus, for this paper, we could have defined the following mapping from instances of MIN SC to MIN RS for the rules in MIN RS:

For each $f_j \in F$, where $1 \leq j < m$, let

$r\_match(f_j,e_i) = 1$ if $x_i \in C_j$

else $r\_match(f_j,e_i) = 0$

$f_j.O = 1$

For $j = m$ let

$r\_match(f_j,e_i) = 1$ for all $e_i \in T$

$f_j.O = 0.$

Note that with this mapping the set of examples which each rule matches is the same as in the mapping given in the proof to theorem 1. However, the mapping does not depend upon any specific formulation for the rules or for the examples, nor does it specify the exact inner workings of the function *f_match*. Thus, the results of this paper can be generalized to any rule-based system where examples are classified by first matching them with rules via some type of deterministic matching function, and then using those rules to classify the examples.

For example, consider a system which uses an ordered list of rules to classify examples. The system classifies novel examples by selecting the first rule in the list which matches each example and classifying it accordingly - a winner take all strategy. The question we would ask is, what is the shortest such list which classifies all examples correctly? With slight modification, the proof of theorem 1 can be used to show that this problem is NP-complete. In this case the set F' does not change, and the modification simply relies on the fact that rule f_m must be the last rule in the list, and the other elements of F' may then be arbitrarily ordered ahead of f_m.

It can also be seen that the problem of determining the simplest set of rules which accurately describes a given data set is (in general) still NP-Complete, even when the complexity of individual rules is included in the calculation of the over-all complexity of a given set of rules. For example, in the proof of theorem 1 the rules f_1 through f_{m-1} all have exactly one input value specified, with all other input variables set to a "don't care" state, while rule f_m has ALL of its variables set to the "don't care" state. Thus, a reasonable rule complexity measure might assign each of the rules f_1 through f_{m-1} the same complexity value (since each of these rules requires a test on exactly one of the input variables - all input variables being 3-state variables), and would assign a lower complexity value to rule f_m (since it does not require a test on any variables to determine a match - it always matches any example). In this case the proof of theorem 1 remains unchanged, since the set F' will remain unchanged. We have already shown that the set F' must contain rule f_m, since it is the only rule which covers example e_n. The remaining rules are then of equal complexity, and we are thus free to choose rules from the remaining set of rules without the need to worry about the complexity of individual rules. The set of rules F' which we obtain in this case is then equal to the set of rules F' which we obtain in theorem 1 (in other words, if F' is an answer for MIN RS, then F' will also be an answer in this case, and visa versa).

5.3 Maximizing Classification Accuracy

The maximization of classification accuracy is an important optimality measure which all rule-induction algorithms attempt to approximate. Generally speaking, it is desirable that the learning algorithm select the set of rules which optimizes classification accuracy over the examples in the training set. There are, of course, other important factors which also constrain which rules are available for selection, such as the complexity of the set of rules (discussed above), the statistical significance of the individual rules, and avoiding learning noise or overfitting the data in the training set. Even though these are important issues, this section mostly ignores them and is only concerned with the single optimality measure of classification accuracy. However, the problem is stated in terms of maximizing the classification accuracy over any given set of rules, where the set of rules could be constrained in a variety of ways. So, even with a constraint on what types of rule sets are available, the problem can still be computationally intractable.

The general problem can be stated as follows.

MAX RULE SET ACCURACY (MAX RSA)

INSTANCE: = (*E*,*F*), a set of examples *E* and a set of rules *F*.

THE PROBLEM: Find a metric assignment for *F* which maximizes the set of elements of *E* which are correctly classified.

At first glance, the MAX RSA problem seems simple enough. However, it is analogous to the NP-complete problem of MAX 2SAT (Garey and Johnson 1979), which asks, "Given a set U of variables, a collection C of clauses over U such that each $c \in C$ has $|c| = 2$, find a truth assignment for U which maximizes the number of satisfied clauses in C." Both of these problems are relatively simple if the maximum percent accuracy (maximum number of clauses which can be satisfied) is 100 percent, and are in fact solvable in polynomial time if this is the case.

However, if the maximum percent possible is less than 100 percent, then the problem of maximizing the accuracy over a set of rules becomes much more difficult. For example, if we think of each rule as being an input to a single perceptron, where the input is high if the rule is matched and low otherwise, then the problem of finding a metric assignment to *F* which maximizes the number of correctly classified examples is equivalent to finding the hyperplane in the extended input space (by training the perceptron node via the Delta Rule or something similar) which optimally separates the elements of the training set.

The problem of finding the optimal ordering for a set of rules in a rule list can also be computationally intractable. This problem is essentially a subset of the problem of allowing all rules to vote according to a strength metric. For example, a scheme which allows all rules to vote could implement a rule list by assigning the bottom rule in the list a metric of 1. The next rule would be assigned a metric of 2, the next 4, and so on, doubling the metric at each step. This assignment will guarantee that the top rule in the list will win, no matter how the rules below it vote, since each rule in the list has been assigned a metric value which is one greater than the summed metrics of all other rules below it.

The difficulty of finding an optimal ordering for a particular set of rules is further illustrated by the following example. Suppose that we have the set of examples $E = \{e_1, e_2, \ldots e_{10}\}$, where for $1 \leq i \leq 5$ e_i.o = 1, otherwise e_i.o = 0. Suppose also that we have the set of rules $F = \{f_1, f_2, f_3\}$, where

$$f_1.o = 1 \text{ and } \{e | e \in E \wedge r\_match(f_1, e)\} = \{e_1, e_2, e_3, e_4, e_6\}$$
$$f_2.o = 0 \text{ and } \{e | e \in E \wedge r\_match(f_2, e)\} = \{e_4, e_5, e_7, e_8, e_9, e_{10}\}$$
$$f_3.o = 0 \text{ and } \{e | e \in E \wedge r\_match(f_3, e)\} = \{e_5, e_6\}$$

Suppose further that the learning algorithm is a greedy, top down list building algorithm which selects the single most optimum rule at each step, and the measure of optimality is the percentage accuracy of the rule over the remaining set of unmatched examples. Initially, the accuracies for the rules are f_1 = 80%, ,f_2 = 67%, and,f_3 = 50%. Thus the first rule is selected to be at the top of the list. Matched examples are removed from consideration, and the remaining rules now match

$$\{e | e \in E \wedge r\_match(f_2, e)\} = \{e_5, e_7, e_8, e_9, e_{10}\}$$
$$\{e | e \in E \wedge r\_match(f_3, e)\} = \{e_5\}$$

The accuracy for rule f_2 is now 80%, and the accuracy for rule f_3 is 0%. Thus, rule f_2 is selected and added below rule f_1 in the list, covering the remaining examples. The overall accuracy for the set of rules is 80%. However, a more accurate rule list can be created if the learning algorithm first selects rule f_3, the rule which initially has the worst accuracy score, then f_1 and finally f_2. In this case the overall accuracy is 90%, with only one example being misclassified. In general, given any greedy approximation algorithm for find MAX Rule Set Accuracy, it should be possible to construct a problem which leads the algorithm astray.

5.4 NP-Completeness Conclusion

The problem of determining the smallest set of rules which covers a set of examples and correctly predicts their classification has been shown to be NP-Complete for a specific type of rule and generalization algorithm. It has also been shown how the more general problem, where the specific form of the rules and generalization algorithm is not exactly defined, is NP-Complete given a few reasonable assumptions about how rules match examples, and how the generalization algorithm uses those rules to classify examples. The assumptions are that the matching of rules with examples must be deterministic, and the output of the generalization algorithm is undefined in the case where an example has no matching rules. In addition, the probable NP-Completeness of the problem of maximizing the classification accuracy of a given set of rules has been discussed.

Assuming that MAX RSA is NP-Complete (a proof of this is currently in preparation), and having proven the NP-Completeness of MIN RS, this implies that we will have to use an algorithm which returns an approximate solution for these problems, since an algorithm which returns the optimal solution will take time exponential in the size of the problem. In addition, the proof of the NP-Completeness of MIN RS given in this thesis uses a transformation from MIN SC to MIN RS which is similar to that given by (Van Horn and Martinez 1994) for the minimum feature set problem. Due to the nature of this transformation it can also be shown that in addition to being NP-Complete, the problem of approximating MIN RS is at least as difficult as the problem of approximating MIN SC (for which it has been shown that no arbitrarily close approximation exists). In addition to this, the current best approximating algorithm for MIN SC is only guaranteed to approximate MIN SC to within a factor $ln|X| + 1$ of the size of the optimal solution (not a constant factor). This implies that it will be very difficult to find any approximating algorithm for MIN RS which returns a result better than a log factor of optimal.

6 Methods

The main goal of this thesis is to demonstrate the benefit (or lack thereof) of higher-order rules. Thus, the approach used in this thesis does not limit the size of the search space in the sense that it generates the set of all possible rules for a particular problem, where the rules are in DNF (disjunctive normal form), and then selects from this set the final set of rules which will be used for classification. Rather, the algorithm generates all interesting attribute conjunctions, where we say that a attribute conjunction is interesting if it appears as part of at least one example from the training set, and then tests each conjunction to determine if it will be included as part of a rule in the final set of rules.

In order to demonstrate the benefit of higher-order rules, the learning algorithm is first run using the set of all possible rules. After assessing the generalization accuracy of the learning algorithm using unbounded rule sets, an upper bound is placed on the order of rules, and the learning algorithm is ran using this bounded-order set. This continues until all possible upper bounds on rule order have been tested. The results are then analyzed and compared with results obtained with a well established machine learning method, the C4.5 decision tree induction algorithm.

It is recognized that there are a number of interdependent factors which will influence the outcome of this phase of the research. For example, the choice of method for metric assignment could have an effect on which of the voting schemes appears to work the best and different data sets could also bias the results. However, the goal of this thesis is not to find the optimum method for metric assignment and conflict resolution. Rather, the goal of this thesis is to find reasonable methods for assigning metrics and resolving conflicts between matched critical rules, and then use these methods to show that many, if not most real world learning problems are not benefited by the addition of higher-order rules.

While there are obviously many other ways in which rules could be used for pattern classification, it was felt that the methods listed here are sufficient to demonstrate the main points of this thesis. It is true that some conflict resolution methods could show some favoritism for higher-order rules, while other methods might favor the use of lower-order rules. However, some of the conflict resolution methods used in this thesis have been widely used in the field of rule induction, such as methods which organize the rules into lists. Since these widely used methods appear to favor lower-order rules from an empirical standpoint, this is an indication that search techniques involving these methods can be justifiably limited (or biased) to lower-order rules. In addition, some of the methods used in this thesis are novel, such as the Greedy training method given in section 6.2.5.1. If these methods also favor lower-order rules, then it could indicate from an empirical standpoint that in general the restriction of the rule search space to lower-order rules is a valid search bias.

The methods which are used in this thesis to resolve conflicts between matched rules, or the generalization schemes, are defined in section 6.1. The various methods tested in this thesis for assigning metrics to rules are defined in section 6.2.

6.1 Resolving Conflicts Between Matched Rules

The voting methods used in this thesis involve assigning a metric to each rule, and then allowing all matching rules to compete or vote for an output classification according to their assigned metric. Several different approaches to rule voting are implemented. These are given in sections 6.1.1 through 6.1.3.

6.1.1 A True Democracy

The most obvious voting scheme is that of a true democracy in which all individuals are allowed to vote. The democratic voting method used in this thesis differs slightly from a normal democracy (where each individual casts a vote of equal value) in that each individual rule votes according to the strength of its metric. Given an example, the metrics of all matching rules are summed, and the output class with the greatest corresponding sum is chosen as that example's classification. The formal algorithm is given in figure 6.1.

Given the following:

a problem domain D
a set E of examples taken from D
a set F of features defined on D

For each $e \in E$, choose the predicted output classification the output class o_i which maximizes the summation (ties being resolved arbitrarily):

$$\sum_{f \in F \wedge r\_match(f,e) \wedge f.o = o_i} f.p \quad .$$

Figure 6.1. Formal democratic voting algorithm.

Problem domain D = the set of all mushrooms.

Each mushroom is measured on 3 attributes, stalk length (SL), stalk color (SC), and cap color (CC). The problem is to determine from these three attributes whether a given mushroom will cause nausea, death, or delight upon being eaten.

Figure 6.2. Example Learning Problem.

For example, suppose that we have the learning problem defined in figure 6.2. Furthermore, suppose that the learning algorithm has selected the set of rules listed in figure 6.3. Given the instance with input values shown in figure 6.4, the set of rules which matches this example is then shown in figure 6.5. The generalization algorithm then sums the metrics of these rules for their corresponding output classes, and obtains a sum of 1.75 for the edible class, and a sum of 1.70 for the death output class. The generalization algorithm would therefore choose the edible class as the predicted output classification for the given example, since it had the greatest corresponding sum.

Rule set F =						
	SL	SC	CC		Outcome	Metric
f_1	SL: long	SC: yellow	CC: orange	-->	edible	1.00
f_2	SL: long	SC: *	CC: orange	-->	death	.95
f_3	SL: long	SC: *	CC: *	-->	edible	.91
f_4	SL:*	SC: red	CC: *	-->	edible	.82
f_5	SL:*	SC: *	CC: orange	-->	death	.75

Figure 6.3. Set of rules for classifying mushrooms.

Mushroom = SL: long	SC: red	CC: orange

Figure 6.4. Example mushroom.

f_2	SL: long	SC: *	CC: orange	-->	death	.95
f_3	SL: long	SC: *	CC: *	-->	edible	.90
f_4	SL:*	SC: red	CC: *	-->	edible	.85
f_5	SL:*	SC: *	CC: orange	-->	death	.75

Figure 6.5. Set of matching rules.

The advantage of this method is that it allows greater flexibility in the way rules are used to classify novel examples than, say, organizing the rules into a list and using the first matching rule to classify novel examples. This means that for any given set of rules and examples, this method will be theoretically capable of doing at least as well at classifying novel examples as any method which organizes the rules into a list, assuming that appropriate metric values are assigned to the rules. The problem, of course, is to find the appropriate metric values, which may in fact be very difficult. Another disadvantage of allowing all matching rules to vote is that the set of all matching rules may be quite large. Thus, the reason for making a decision will be more difficult for a human to comprehend, with potentially tens or hundreds of rules combining to predict classification, than it would with a rule list implementation, where only a single rule is being used to make the decision.

6.1.2 Winner Takes All

The single greatest rule voting method chooses the matching rule which has the greatest metric, and uses that rule's corresponding output classification as the example's predicted output classification. This is equivalent to ordering the rules in a list and using the first matching rule to predict classification, a commonly used approach. The formal algorithm is given in figure 6.6. Going back to the example given above, the generalization algorithm will predict that the example will cause death upon being ingested, since this is the output classification associated with rule f_2 (this rule has the greatest metric of all matching rules). Notice that this prediction differs from that predicted by the Democratic voting scheme.

Given the following:

a problem domain D
a set E of examples taken from D
a set F of features defined on D

For each example $e \in E$, define F_e to be the set of rules from F which match e. In other words, let

$$F_e = \left\{ f \middle| f \in F \wedge r\_match(f, e) = True \right\}.$$

Then for all $e \in E$, if $F_e = \varnothing$ then the predicted output classification is undefined, else choose the predicted classification to be $f.o$, where the following is true about the rule f (ties are resolved arbitrarily):

$$(f \in F_e) \wedge (\forall y)(y \in F_e \rightarrow f.p \geq y.p).$$

Figure 6.6. Rule list generalization algorithm.

The advantage of this method is that it is relatively simple to implement, and the decision making process is more easily comprehended by a human. Also, since this method is simpler than the voting method, the number of different hypotheses which this method can represent is a proper subset of the number of different hypotheses which the democratic voting method can represent. This is both an advantage and disadvantage. By choosing this method for generalizing new instances, the size of the hypothesis space has been greatly reduced. This should mean that, assuming the hypothesis one is looking for (or a reasonable approximation to it) is in the reduced search space, it should be easier to find. However, if the reduced search space does not contain a good hypothesis, the learning algorithm is dead before it begins.

6.1.3 The Republic

The Republic voting method allows only those matched rules whose attribute conjuncts are not a subset of any other matched rule's attribute conjuncts to vote for the example's output classification. The formal algorithm is given in figure 6.7. The intuition behind this method is as follows. Suppose that there are two rules, r_1 and r_2, where the set of input attribute-value pairs which are conjuncted in r_1 is a proper superset of the set of input attributes-value pairs that are conjuncted in r_2 (for example, the rules f_1 and f_2 given in figure 6.3 fulfill these criteria). Then we know that for any given example, if r_1 matches that example then this implies that r_2 will match that example. In other words, r_1 is a more specific rule than r_2, which implies that the case where r_1 is matched relates more information than the case where only r_2 is matched.

Given the following:

a problem domain D
a set E of examples taken from D
a set F of features defined on D

For each $e \in E$, choose the predicted output classification the output class o_i which maximizes the summation (ties being resolved arbitrarily):

$$\sum_{f \in F \wedge r\_match(f,e) \wedge f.o = o_i \wedge (\forall y)(y \in F \wedge r\_match(y,e) \rightarrow f \not\subset y)} f.p \quad .$$

Figure 6.7. Republic voting method.

For example, r_1 could be the rule that an animal with lots of sharp teeth is a predator, while r_2 could be the rule that an animal with lots of teeth is a predator. It is obvious that if an animal has lots of sharp teeth (matching rule r_1) then it is also true that the animal has lots of teeth (matching rule r_2). It is also obvious that the knowledge that a particular animal has lots of sharp teeth is a more likely indication that the animal is a predator than only knowing that the animal has lots of teeth. Therefore, if r_1 is matched it seems reasonable that we should ignore the lower confidence (metric) of r_2 in calculating the output classification.

While the example given above demonstrates that the Republic method is intuitively appealing, the use of this method does have the drawback that it will decrease the likelihood that lower-order rules will participate in voting. Given a particular instance from the problem domain, the set of matching rules which will be allowed to vote for that instance's classification will then tend to be composed mostly (or possibly entirely) of higher-order rules. Since higher-order rules cover less of the input space than lower-order rules, the predictive value of such rules can be difficult to ascertain due to the decreased probability that there will be sufficient numbers of training instances which match them. This problem can be remedied somewhat by deleting rules which do not occur often enough in the training set to be statistically significant.

f_2	SL: long	SC: *	CC: orange	-->	death	.95
f_4	SL:*	SC: red	CC: *	-->	edible	.85

Figure 6.8. Set of rules selected by the Republic voting method.

Returning to the mushroom classification problem given in figure 6.2, the Republic method would select the set of rules listed in figure 6.8 to participate in voting. The sum for the edible class would then be .85, and the sum for the death classification would be .95, so the generalization algorithm would predict the example of figure 6.4's output classification to be death.

6.2 Metric Assignment

The methods for assigning metrics are largely dependent on what method is used to resolve conflicts between matched rules. This thesis implements several methods for assigning metrics to rules. These are listed along with a brief description of each method in sections 6.2.1 through 6.2.5.

6.2.1 Count

One of the simpler metric assignment method is to count the number of times a rule matches an example of the same output class from the training set, and then set the rule's metric equal to this number (see figure 6.9). While this metric assignment method is easy and straightforward to apply to a set of rules, it has the drawback of not doing any rule set minimization by itself.

For each $f \in F$,

$$f.p = \left|\left\{ e \middle| e \in T \wedge e.o = f.o \wedge r\_match(f, e) \right\}\right|$$

Figure 6.9. Count metric assignment method.

With this method, rules which occur infrequently in the training set will be assigned relatively small metric values. This is desirable, since rules which occur less often may be the result of noise, or some other aberration. Thus, this method naturally limits the effect of such rules on generalization, without the need to resort to statistical significance tests.

6.2.2 One

This is the simplest method of all, and is an implementation of a true democracy. For all $f \in F$ it assigns $f.p = 1$. This method will be useful as a baseline indicator. Assuming that each rule in the set of rules has a better than even chance of predicting the correct classification, then this method should produce reasonably good results, especially as the maximum order of stored rules is increased.

6.2.3 p($o|f$)

This method attempts to estimate the probability that a given rule is associated with a particular output by examining the examples in the training set, and then assigns the rules metric to be equal to this probability. The method for calculating this probability is given in figure 6.10 below.

For each $f \in F$,

$$f.p = \frac{\left|\{e|e \in T \wedge e.o = f.o \wedge r\_match(f,e)\}\right|}{\left|\{e|e \in T \wedge r\_match(f,e)\}\right|}$$

Figure 6.10. $p(o|f)$ metric assignment method.

The main problem with this method is that rules which occur only once will have the highest possible metric (1.00), in spite of the fact that a rule which occurs several times and exhibits the same degree of correlation with a particular output (100 percent), and thus will be assigned the same metric value, would appear to be more reliable.

6.2.4 Training the Metric Via Delta Rule

An algorithm similar to the Delta Rule training algorithm is used to assign metrics to rules. This algorithm (given in figure 6.11) has the advantage that it is guaranteed to converge to a solution if the elements in the training set are linearly separable. In the case where the training set is not linearly separable the Delta Rule training algorithm is often able to generate a relatively good, approximate solution.

Given:

E, a set of examples

F, a set of rules

c, an integer valued constant

while $(\exists e)(e \in E \wedge gen(e,F) \neq e.o)$

for each $e \in E$ do

if $gen(e,F) \neq e.o$ do

let $W = \{f | f \in F \wedge r\_match(f,e) \wedge f.o \neq e.o \wedge gen(e,F) = f.o\}$

let $V = \{f | f \in F \wedge r\_match(f,e) \wedge f.o = e.o\}$

for each $f \in W$ let $f.p = f.p - c$

for each $f \in V$ let $f.p = f.p + c$

end

end

end

Figure 6.11. Delta Rule training algorithm.

While there are many training sets which do not exhibit linear separability, the process of combining attributes into higher-order rules can generate a linearly separable input space from one that was initially non-linearly separable. In fact, a linearly separable problem can always be generated from a non-linearly separable one if we are willing (and able) to look at combinations of attributes which are of high enough order (assuming that no two examples from the training set with the same input values have different classifications, or in other words assuming that the training set is consistent). However, the hope is that most real-world problems will exhibit the characteristic of linear separability with sets of rules that are of relatively low-order, since it would appear that generating higher and higher order rules is in general a prohibitively expensive and difficult task.

The weakness of this method is that the higher order rules tend to be assigned larger metric values. This is not a problem if the high-order rules are important. However, in general this training method when used on higher-order rule sets leads to memorization of the training set, rather than the desired, general functional mapping.

6.2.5 Greedy Training Methods

This thesis implements three greedy training methods, one of which is used in conjunction with the DEMOCRACY voting scheme, and the others are used in conjunction with the MONARCHY or rule list voting scheme. The three methods are referred to as Greedy, Topdown (a top down method), and Bottomup (a bottom up method). Each algorithm, along with the intuition behind each, is explained in detail shortly. The three greedy training methods have several common elements. These are:

- Initially, the set of rules used by the generalization algorithm for classifying new instances (called *OPT*) is empty.
- There is a pool of rules which the greedy training method can draw upon.
- At each step of the algorithm, either the single most optimal rule from the pool of rules is added to *OPT*, or the metric of a rule from *OPT* is modified, whichever produces the greatest gain in classification accuracy.
- The algorithm terminates when no gain can be made in classification accuracy.
- The algorithm naturally minimizes the size of the set *OPT*.

6.2.5.1 A Greedy Democracy

In addition to the items listed in definition 4.13 of chapter 4, each rule is augmented with the following information:

score: Indicates the net gain in number of examples from the training set which will be correctly classified if the given rule's metric is set to *f.opt_met* (see below).

pos: The number positive, misclassified examples from the training set which this rule matches (an example is said to be positive in relation to a rule if the example's output matches the rule's output).

neg: The number of negative, correctly classified examples from the training set which will become misclassified if the rule's metric is set high enough to correctly classify all of the examples indicated by *pos*.

curr_neg: The number of negative, incorrectly classified examples from the training set which this rule matches.

curr_pos: The number of positive, correctly classified examples from the training set which will become misclassified if the rule's metric is set low enough to cause all examples indicated by *curr_neg* to be correctly classified.

opt_met: Set to one of either two values. Either it is set to the smallest positive integer needed for the rule to classify all *pos* examples, or it is set to zero, whichever gives the greatest increase in classification accuracy.

This extra information is required as part of the book-keeping procedures of the Greedy training algorithm.

The basic Greedy algorithm is formally defined in figures 6.12 through 6.15. Initially, the set of rules used by Greedy for classification (called *OPT*) is empty. There is a set of rules (called *POOL*) which Greedy can draw upon to add rules to *OPT* in order to increase classification performance. At each step of the algorithm, all of the rules in *POOL* are evaluated, and each rule is assigned a score which is based upon how much that rule would increase classification performance if it was added to *OPT* (function *FindBestPoolRule*, figure 6.13) . The score is simply the net gain in the number of correctly classified examples if the given rule is added to *OPT* with a metric value set to *opt_met*. In this case, *opt_met* is set to the smallest integer which will guarantee that all of the positive examples which the given rule matches will be correctly classified.

In addition to evaluating all of the rules in *POOL* at each step, the algorithm also evaluates all of the rules in *OPT* (function *FindWorstOptRule*, figure 6.12). Each rule in *OPT* is given a score based upon the increase in classification accuracy obtained if that rule's metric is increased to the point where all of the positive examples it matches are correctly classified. After all of the rules in *OPT* and *POOL* have been assigned scores, the rule with the greatest score is selected and added to *OPT* with metric set to *opt_met* (or if the rule is already in *OPT* its metric is increased to *opt_met*).

Finally, at the end of each step each rule in *OPT* is tested to see if the removal of the rule from *OPT* increases classification accuracy (procedure *DeleteBadRules*,figure6.14). If it does, or if classification accuracy is not affected by removal of the rule, then the rule is immediately removed from *OPT* and returned to *POOL*. This is repeated until no more rules can be removed from *OPT*.

For each $f \in OPT$ do

$W = \{e | e \in E \wedge gen(e, OPT) \neq e.o \wedge e.o = f.o\}$

$f.pos = |W|$

let OPT' be a set of rules which is equal in all respects to the set of rules OPT

let $f' \in OPT'$ where f' is equal to f of OPT

increase $f'.p$ to the smallest integer x such that

$(\forall e)(e \in W \rightarrow gen(e, OPT') = e.o)$

$f.neg = |\{e | e \in E \wedge gen(e, OPT) = e.o \wedge gen(e, OPT') \neq e.o\}|$

$f.opt\_metric = f'.p$

$f.score = f.pos - f.neg$

end

let y be a rule such that the following statement holds for y:

$y \in OPT \wedge (\forall z)(z \in OPT \rightarrow (z.score \leq y.score))$

return y

Figure 6.12. Function *FindWorstOptRule*.

for each $f \in POOL$ do

$W = \{e | e \in E \wedge gen(e, OPT) \neq e.o \wedge e.o = f.o\}$

$f.pos = |W|$

let $f.p$ equal the smallest integer x such that

$(\forall e)(e \in W \rightarrow gen(e, OPT \cup \{f\}) = e.o)$

$f.neg = |\{e | e \in E \wedge gen(e, OPT) = e.o \wedge gen(e, OPT \cup \{f\}) \neq e.o\}|$

$f.opt\_met = f.p$

$f.score = f.pos - f.neg$

end

let y be a rule such that the following statement holds for y:

$y \in POOL \wedge (\forall z)(z \in POOL \rightarrow (z.score \leq y.score))$

return y

Figure 6.13. Function *FindBestPoolRule*.

```
do
    For each f ∈ OPT do
        f.curr_pos = |{e | e ∈ E ∧ gen(e,OPT) = e.o ∧ gen(e,OPT − f) ≠ e.o}|
        W = {e | e ∈ E ∧ gen(e,OPT) ≠ e.o ∧ gen(e,OPT − f) = e.o}
        f.curr_neg = |W|
        f.score = f.curr_neg − f.curr_pos
    end
    let y be a rule such that the following statement holds for y:
        y ∈ OPT ∧ (∀z)(z ∈ OPT → (z.score ≤ y.score))
    if y.score ≥ 0
        OPT = OPT − y
        POOL = POOL + y
    end
while a rule is deleted
```

Figure 6.14. Procedure *DeleteBadRules*.

```
let POOL = F
let OPT = ∅
do
    P_rule = FindBestPoolRule()
    O_rule = FindWorstOptRule()
    if O_rule.score ≥ P_rule.score
        if O_rule.opt_metric = 0
            OPT = OPT − O_rule
            POOL = POOL + O_rule
        end else
            O_rule.p = O_rule.opt_metric
        end
    end else
        POOL = POOL − P_rule
        OPT = OPT + P_rule
    end
    DeleteBadRules()
while improvement takes place
```

Figure 6.15. Greedy training algorithm.

We are guaranteed that the algorithm will terminate in at most $|E|$ steps, since at each step of the algorithm there is either a net increase of at least one in the number of examples correctly classified, or there is not and the algorithm terminates. Also, when the algorithm does terminate (and at any point during the execution of the algorithm) there will be at most $|E|$ rules in the set of rules OPT, since each step of the algorithm adds at most one rule to OPT. Furthermore, the upper time bound on the FindBestPoolRule function is $O(|F|)$, the upper time bound on FindWorstOptRule is $O(|E|)$, and the upper time bound on DeleteBadRules is $O((|E|^2+|E|)/2)$. Therefore we have a worst case upper time bound of $O(|E||F| + (|E|^3+|E|^2)/2)$ on the execution of the algorithm.

6.2.5.2 Top Down Greedy Generation of Rule Lists

The top-down greedy training algorithm for generating rule lists, or Topdown, is given in figure 6.16. Initially, all examples are assumed to be incorrectly classified, and the set of rules OPT is empty. At each step of the algorithm, the single most optimal rule from $POOL$ is selected and added to the bottom of the list of rules. The algorithm terminates when no more progress can be made in correctly classifying examples from the training set.

$POOL = F$
$OPT = \emptyset,$ an ordered list of rules.
$MISCLASS = E$
do
    for each $f \in POOL$
        if $\{e | e \in MISCLASS \wedge e.o = f.o \wedge r\_match(f,e)\} = \emptyset$
            let $POOL = POOL - f$
        else $f.score = \dfrac{\left|\{e | e \in MISCLASS \wedge e.o = f.o \wedge r\_match(f,e)\}\right|}{\left|\{e | e \in MISCLASS \wedge e.o \neq f.o \wedge r\_match(f,e)\}\right|}$
    end
    if $POOL \neq \emptyset$
        let $f \in POOL$ such that $(\forall y)(y \in POOL \rightarrow y.score \leq f.score)$
        put f at the bottom of the list OPT
        $MISCLASS = MISCLASS - \{e | e \in MISCLASS \wedge r\_match(f,e)\}$
    end
while $POOL \neq \emptyset$ and $MISCLASS \neq \emptyset$

Figure 6.16. Top down greedy list algorithm.

Each step of the algorithm begins by deleting all of the rules in *POOL* which do not match any of the positive examples remaining in *MISCLASS*. The remaining rules in *POOL* are assigned a score which is the ratio of the number of positive examples which the rule matches to the number of negative examples which the rule matches. The score is said to be infinite in the case where the number of negative examples which the rule matches is zero. After assigning each remaining rule in *POOL* a score, the algorithm selects the most optimal rule, or the rule with the largest score, to add to the bottom of the list. In the case where the score between two or more rules is equal, then the rule which matches the greatest number of positive examples is preferred. The examples which the rule matches are then deleted from *MISCLASS*, and the selected rule is deleted from the pool of rules.

As with the Greedy training method, the maximum number of iterations of the main loop is bounded by the number of examples in *E*, since at each step the algorithm either adds a rule which classifies at least one example from *E* (and all matching examples are then deleted from further consideration), or the algorithm terminates. Each iteration of the main loop assigns a score to (or deletes) each rule of *POOL*, then selects the rule with the largest score. Assuming that the rule with the largest score is kept track of during the assignment process, this will require at most $|F|$ steps. Thus the upper time bound on the execution of Topdown is $O(|E||F|)$.

6.2.5.3 Bottom Up Greedy Generation of Rule Lists

In addition to the top-down training method, this thesis also tests a bottom-up approach, called Bottomup. The main difference between this approach and the top-down approach is that it adds rules to the top of the list (as it's name implies).

$POOL = F$
$OPT = \varnothing$, an ordered list of rules.
$CLASS = \varnothing$
$MISCLASS = E$
do
    for each $f \in POOL$
        $f.score = \left|\{e|e \in MISCLASS \wedge e.o = f.o \wedge r\_match(f,e)\}\right|$
            $-\left|\{e|e \in CLASS \wedge e.o \neq f.o \wedge r\_match(f,e)\}\right|$
    end
    if $(\exists f)(f \in POOL \wedge f.score > 0)$
        let $f \in POOL$ such that $(\forall y)(y \in POOL \rightarrow y.score \leq f.score)$
        put f at the top of the list OPT
        $POOL = POOL - f$
        $MISCLASS = MISCLASS + \{e|e \in CLASS \wedge e.o \neq f.o \wedge r\_match(f,e)\}$
            $-\{e|e \in MISCLASS \wedge e.o = f.o \wedge r\_match(f,e)\}$
        $CLASS = CLASS + \{e|e \in MISCLASS \wedge e.o = f.o \wedge r\_match(f,e)\}$
            $-\{e|e \in CLASS \wedge e.o \neq f.o \wedge r\_match(f,e)\}$
    end
while there is progress and $MISCLASS \neq \varnothing$

Figure 6.17. Bottom up greedy list algorithm.

The Bottomup training method keeps track of two sets of examples. The first set, called *CLASS*, keeps track of the examples which are classified correctly. Conversely, the other set, called *MISCLASS* keeps track of the examples which are not classified correctly. Initially, the set *CLASS* is empty and the set *MISCLASS* contains all of the examples in the training set. In addition, the algorithm has a set of rules, called *POOL*, to draw upon for classifying examples from the training set.

Each step of the algorithm proceeds as follows. First, each rule in *POOL* is assigned a score, the score being equal to the net gain in the number of examples which will be correctly classified if the given rule is added to the top of the list of rules. After all rules in *POOL* are assigned a score based on the current situation, the single most optimal rule, or the one with the largest score, is added to the top of the list *OPT*. The rule is then deleted from *POOL*. The sets *MISCLASS* and *CLASS* are then adjusted to reflect the addition of the new rule.

The upper time bound on the execution of Bottomup is the same as that for the top down method, $O(|E||F|)$.

7 Data Sets And Experiments

The research was done in two phases. The first phase of the research attempted to establish the utility of low-order rules versus high-order rules from an empirical standpoint. During this phase of the research, every possible upper bound on the order of extracted rules is used, and the results of the various learning algorithms are reported. A comparison is then made with the results obtained using only 1st order rules, the results obtained using 1st and 2nd order rules, the results obtained using 1st, 2nd and 3rd order rules, and so on up to the maximum possible order of rules (the number of inputs).

The main goal of this paper being to determine the learning and generalization traits of bounded order rule sets on real world problems, several data sets taken from real world problems were selected for testing. In addition to the real world data sets, 3 artificial data sets, called the Monk data sets, are also used. Each data set is randomly divided into two equal parts, a training set and a test set (the exception to this is that the training sets used for the three Monk data sets are identical to the ones supplied by Professor Thrun in the original problem). The data in the training set is used during the training phase of the algorithm to obtain critical rules and for metric calculation, while the data in the test set is used to test the generalization capabilities of the extracted rules.

During the learning phase of the algorithm a search bounded above by some integer less than (or equal to) the number of inputs for critical rules contained in the training set is conducted, and each discovered rule is assigned a metric. After the training phase the test set is used to determine how well the algorithm is able to generalize (predict classification) on novel patterns, utilizing only those rules and metrics extracted during the training phase. 15 Runs using different, randomly selected training sets and test sets were conducted for each metric and generalization method on each data set, and the average accuracy for the 15 runs was calculated.

Due to time constraints, the Delta Rule training method was limited to 200 iterations through the training set. The extremely large number of rules generated with some of the data sets made it impractical to do more, and going beyond 200 iterations did not appear to significantly improve results for the most part.

The second phase of the research compares the results obtained during the first phase with those obtained using the C4.5 tree induction algorithm. The results show that C4.5 is not able to do significantly better than many of the methods explored in this paper using relatively low-order rules (for example, using only 1st and 2nd order rules, the perceptron training method did relatively well in comparison with C4.5).

Since the learning algorithms explored in this thesis require discrete valued attributes, all of the continuous valued attributes were discretized utilizing a simple equal interval discretization technique. Because of the simplicity of this technique, it is possible that some of the discretized data sets will be more difficult to learn due to the potential loss of valuable information. On the other hand, it's possible that they may be easier to learn (although this is less likely). In any case, by comparing the results with those obtained via the C4.5 learning algorithm, it is possible to see how well the learning methods explored in this paper are doing with what they are given.

The data sets which are tested in this thesis are:

- Bupa
- CPU
- Iris
- Lenses
- Monk1
- Monk2
- Monk3
- Pima

- Tic-tac-toe

All of these data sets were obtained from the UC Irvine Machine Learning Database.

The Bupa data set contains 345 examples. Each example represents a patient who may or may not have some form of liver disorder. The data contains the results of 6 tests which have been administered to each patient. The goal is to determine from the results of these 6 tests which patients have liver disorders.

The CPU data set contains 209 examples, each of which represents a type of computer. The goal is to predict the relative performance of each computers' CPU based upon a series of measurements (such as cache memory size, cycle time, etc.).

The Iris data set contains 150 examples. The data contains 3 classes of iris plants. The goal is to determine which class of iris plant each example belongs to from measurements such as sepal length, petal length, etc. This data set has been used extensively in the machine learning literature.

The Lenses data set is an extremely small data set which contains only 24 examples. Each possible example is represented. Each example represents a patient, where the goal is to determine which type of contact lenses to fit to the patient from the patients known characteristics. Unfortunately, since the data set is so small, most learning algorithms have a difficult time generalizing on unseen test cases.

The Monk data sets are artificial data which were used in one of the first international comparison of learning methods. Each data set contains 432 examples. The target concept for Monk1 was generated from the rule ($a1 = a2 \vee a5 = 1$). The correlation between a1 and a2 would seem to imply the need for the learning system to generate second order rules in order to be able to accurately describe the target concept. The concept for Monk2 was generated with the rule that exactly two of the 6 input attributes be equal to 1. It would seem that higher-order rules might be necessary to solve this problem, since to be certain that something is in the concept we must know what all the inputs are (however, it is often possible to tell that something is NOT in the concept from a subset of the inputs, for example if the first three inputs are equal to 1). Finally, the concept for Monk3 was generated from the rules $(a5 = 3 \wedge a4 = 1) \vee (a5 \neq 4 \wedge a2 \neq 3)$. In addition, 5 percent class noise was added to the elements of the training set (which is pre specified). This is the simplest data set of the three.

The Pima data set contains 768 examples of Pima Indians. The goal is to determine which of the patients are diabetic from a series of standard tests prescribed by the World Health Organization.

The Tic-tac-toe data set contains 958 examples. The data set encodes the complete set of all legal board configurations at the end of tic-tac-toe games. The goal is to determine if the board represents a win for the player playing “x”.

8 Results

The results of the first phase of the research are shown in section 8.1. Underneath each figure is listed the conflict resolution and metric assignment method which were used to obtain the results given in the figure. For some of the metric assignment methods (Greedy, Topdown, Bottomup, and Delta Rule), only the metric assignment method is listed, since each of these methods was used with only a single method for conflict resolution.

Since we are mainly interested in the generalization capabilities of the methods, only the results on the test set are reported. The accuracy on the training set was generally close to 100 percent for each of the methods tested in this paper. This is especially true of the greedy and perceptron training methods, and the top down and bottom up list building methods.

In each of the graphs below, the numbers along the vertical axis indicate the percentage accuracy on the test set, and the numbers along the horizontal axis indicate the upper bound on stored rules. Also, the standard deviation for each of the data points on the graph is listed at the bottom of the graph, below the upper bound on rule order with which it is associated.

The results obtained with C4.5 are shown in section 8.2. The first three bars of each graph show the average results over 15 runs for the unpruned and pruned decision trees, and for the rule list generated by the C4.5rules rule list generation algorithm. The second three columns of each graph show the single best result obtained with the methods used in this paper, the average of each methods single best result, and the overall average of all the results of each method. Again, only the results on the test set are reported.

8.1 General Results from Phase 1

8.1.1 Bupa Data Set.

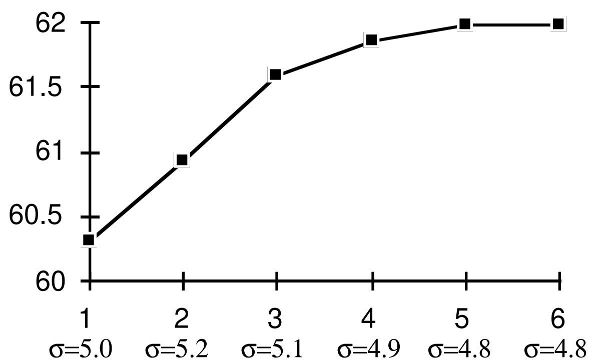


Figure 8.1a. Democracy - Counts.

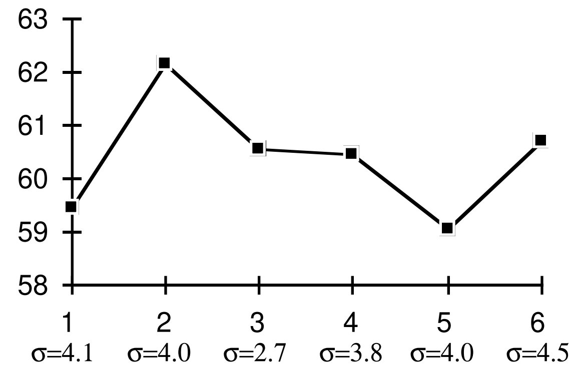


Figure 8.1b. Republic - Counts.

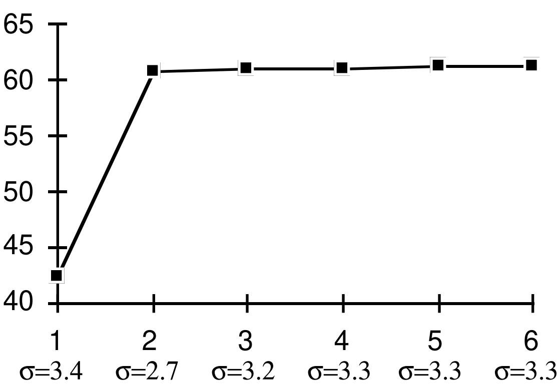


Figure 8.1c. Democracy - one.

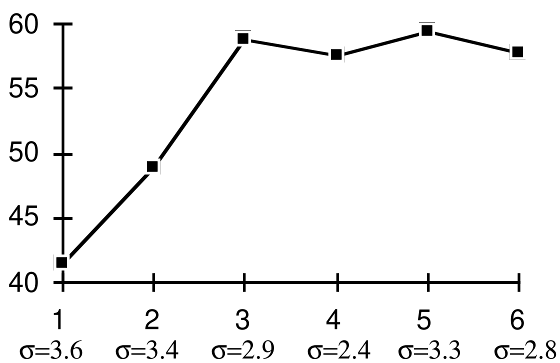


Figure 8.1d. Republic - one.

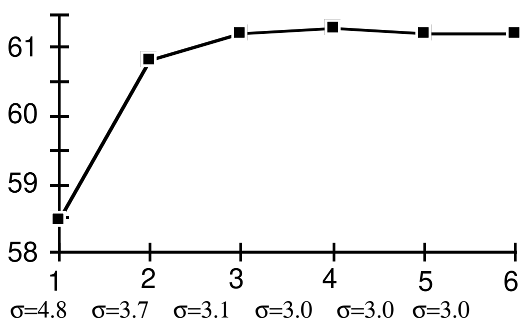


Figure 8.1e. Democracy - $p(o|f)$.

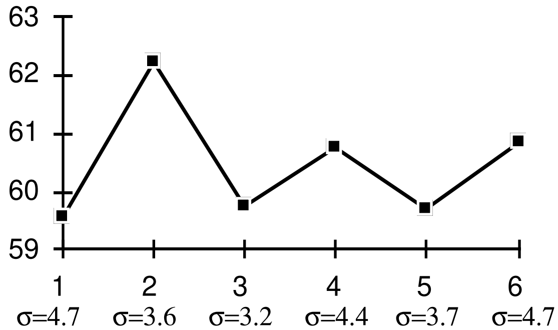


Figure 8.1f. Republic $p(o|f)$.

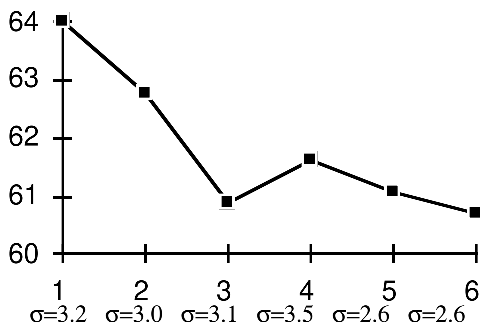


Figure 8.1g. Delta_Rule

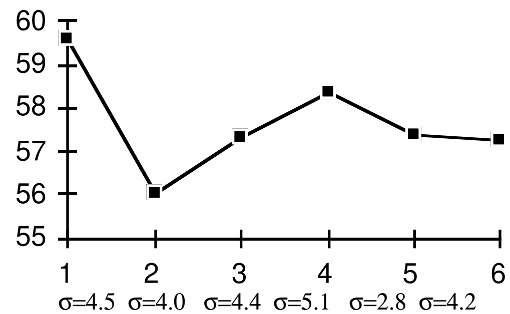


Figure 8.1h. Greedy.

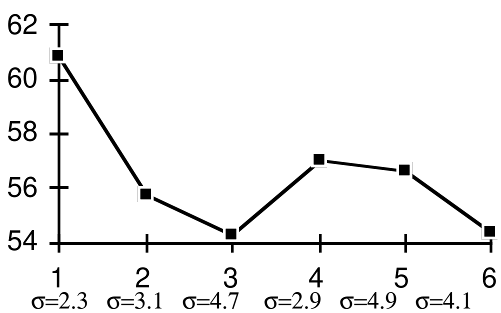


Figure 8.1i. Topdown.

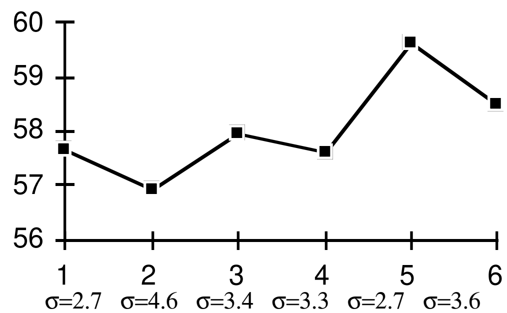


Figure 8.1j. Bottomup.

8.1.2 CPU Data Set

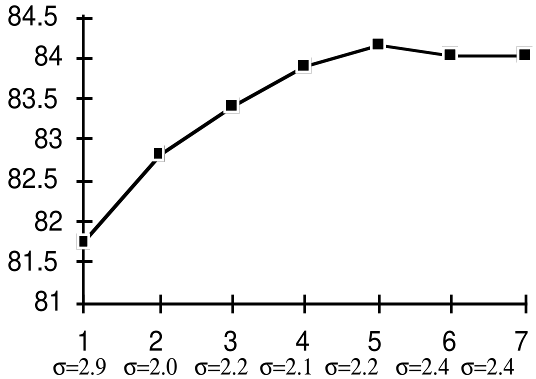


Figure 8.2a. Democracy - Counts.

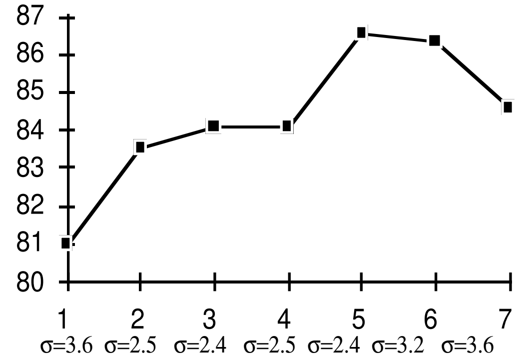


Figure 8.2b. Republic - Counts.

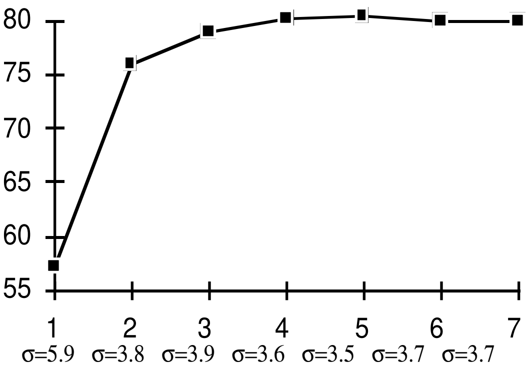


Figure 8.2c. Democracy - one.

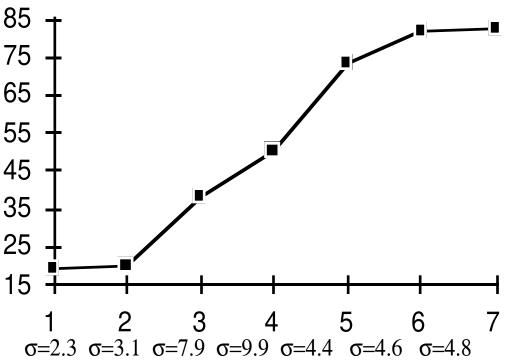


Figure 8.2d. Republic - one.

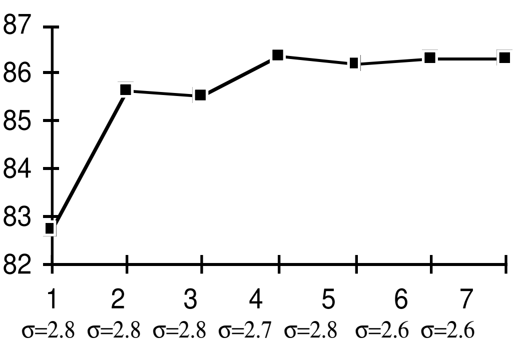


Figure 8.2e. Democracy - $p(o|f)$.

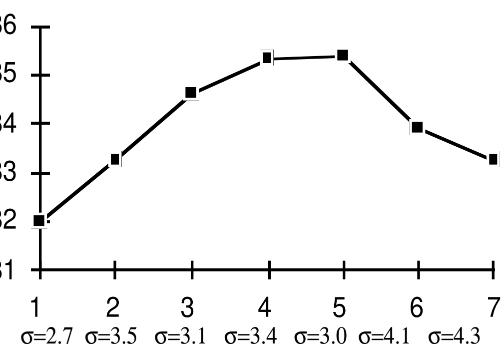


Figure 8.2f. Republic $p(o|f)$.

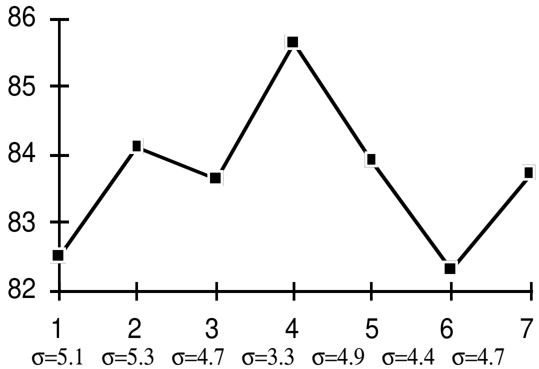


Figure 8.2g. Delta_Rule

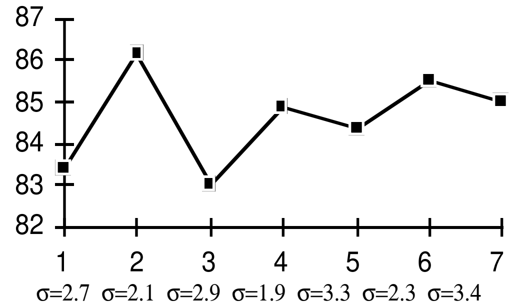


Figure 8.2h. Greedy.

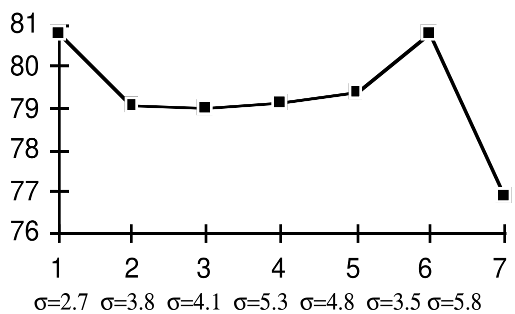


Figure 8.2i. Topdown.

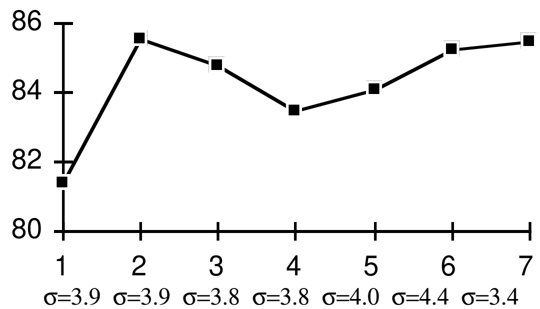


Figure 8.2j. Bottomup.

8.1.3 Iris Data Set

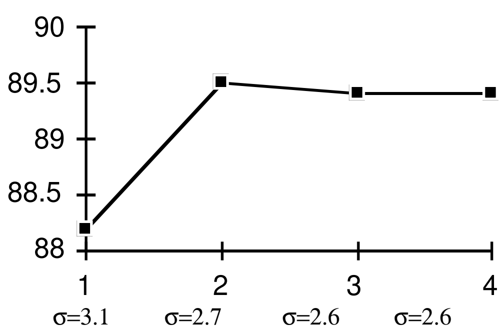


Figure 8.3a. Democracy - Counts.

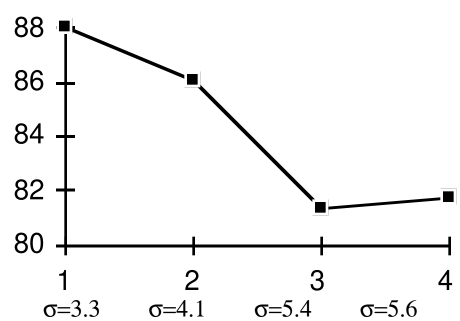


Figure 8.3b. Republic - Counts.

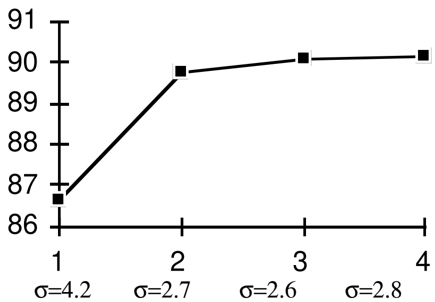


Figure 8.3c. Democracy - one.

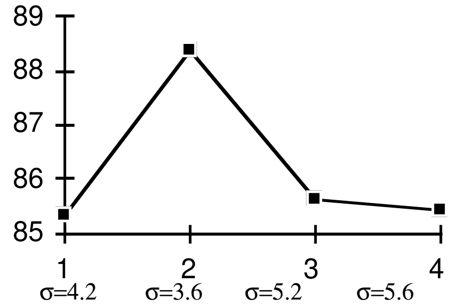


Figure 8.3d. Republic - one.

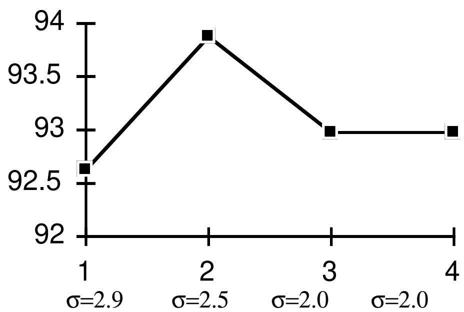


Figure 8.3e. Democracy - p(*o*|*f*).

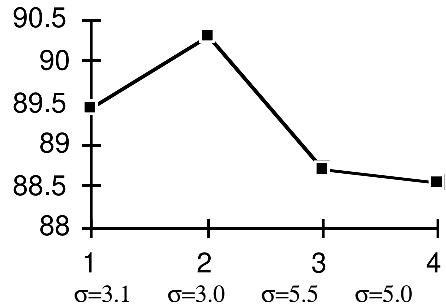


Figure 8.3f. Republic p(o|f).

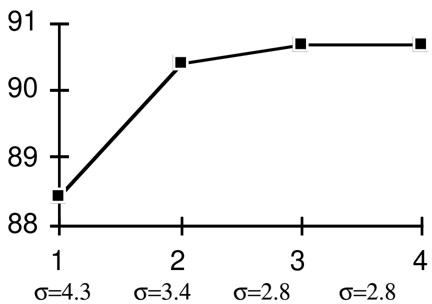


Figure 8.3g. Delta_Rule.

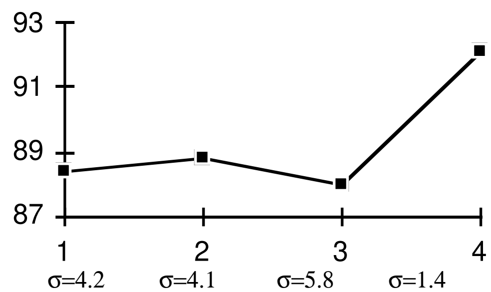


Figure 8.3h. Greedy.

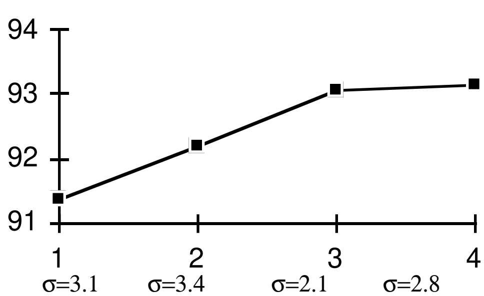


Figure 8.3i. Topdown.

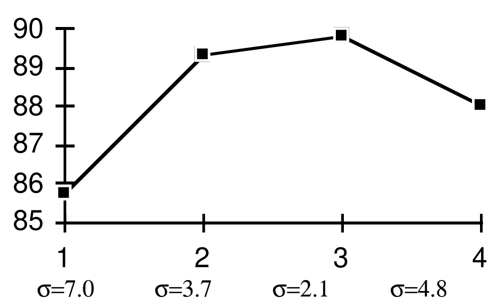


Figure 8.3j. Bottomup.

8.1.4 Lenses Data Set

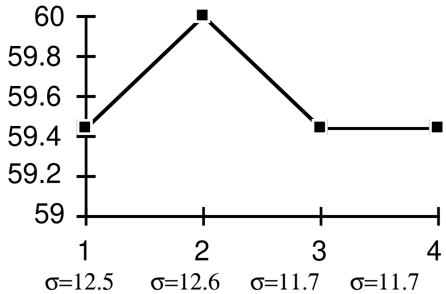


Figure 8.4a. Democracy - Counts.

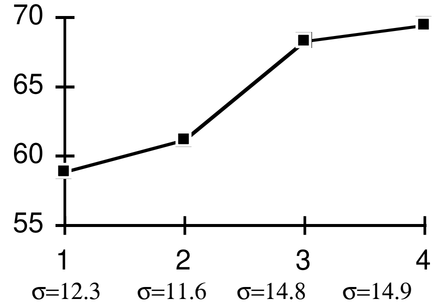


Figure 8.4b. Republic - Counts.

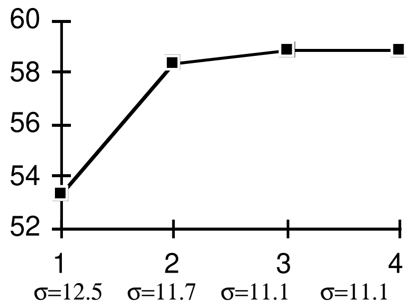


Figure8.4c. Democracy - one.

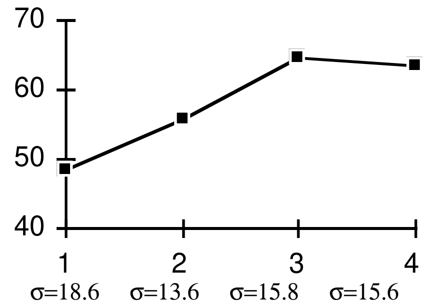


Figure 8.4d. Republic - one.

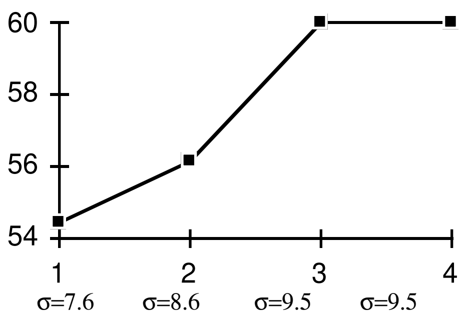


Figure 8.4e. Democracy - p(*o*|*f*).

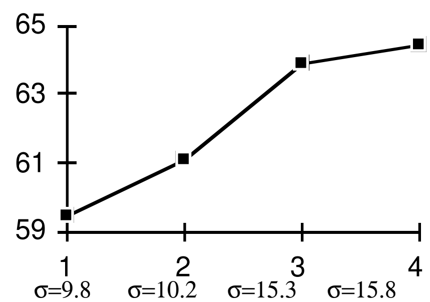


Figure 8.4f. Republic p(o|f).

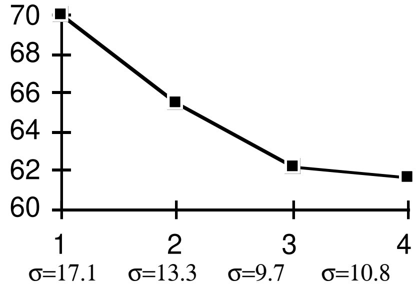


Figure 8.4g. Democracy - Delta_Rule.

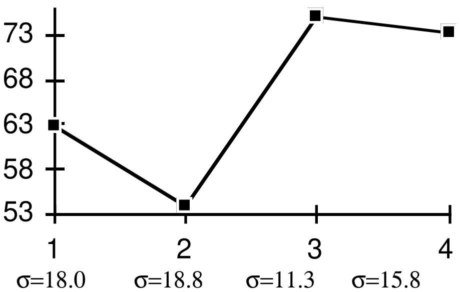


Figure 8.4h. Greedy.

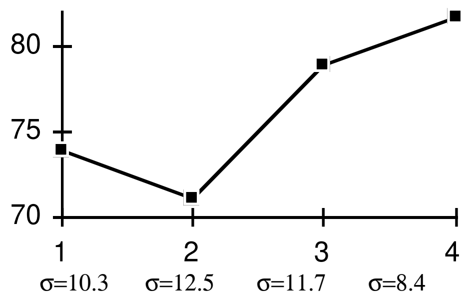


Figure 8.4i. Topdown.

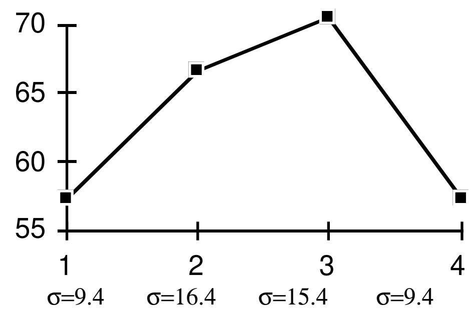


Figure 8.4j. Bottomup.

8.1.5 Monk 1 Data Set

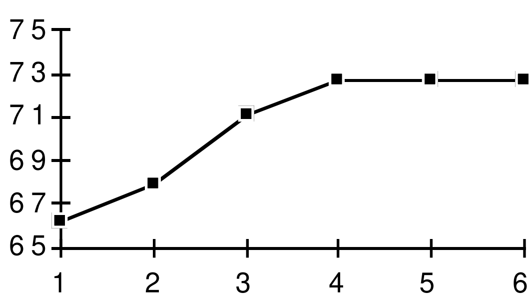


Figure 8.5a. Democracy - Counts.

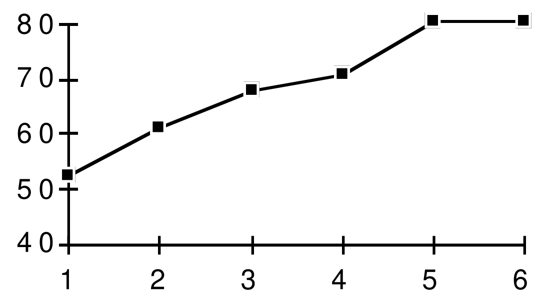


Figure 8.5b. Republic - Counts.

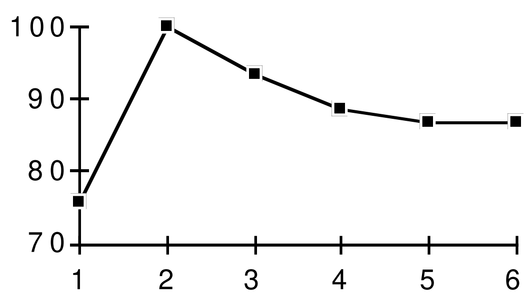


Figure 8.5c. Democracy - One.

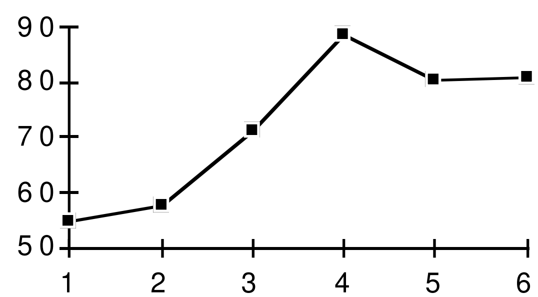


Figure 8.5d. Republic - One.

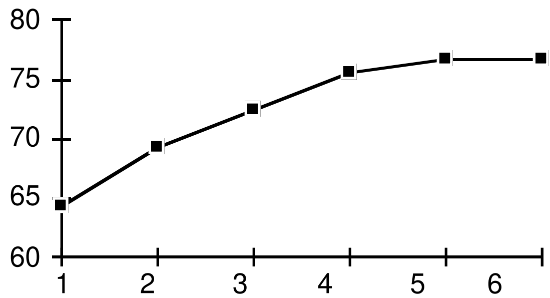


Figure 8.5e. Democracy - $p(o|f)$.

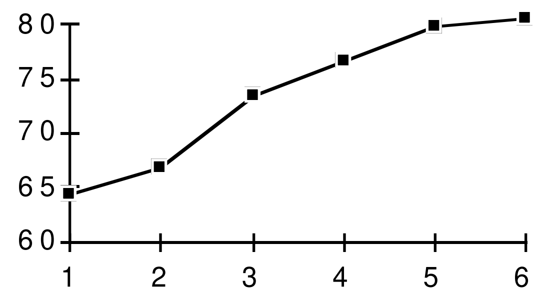


Figure 8.5f. Republic - $p(o|f)$.

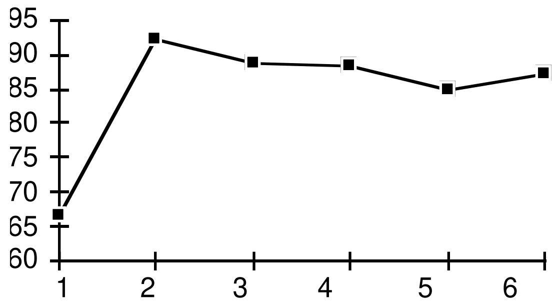


Figure 8.5g. Delta Rule.

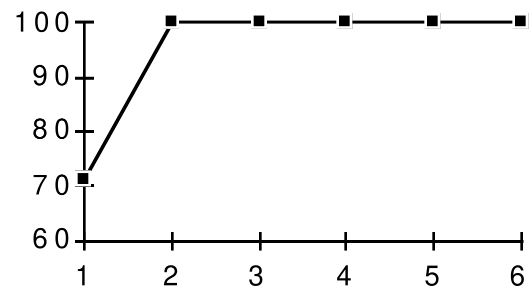


Figure 8.5h. Greedy.

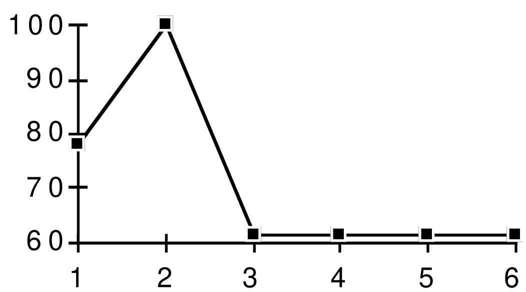


Figure 8.5i. Topdown.

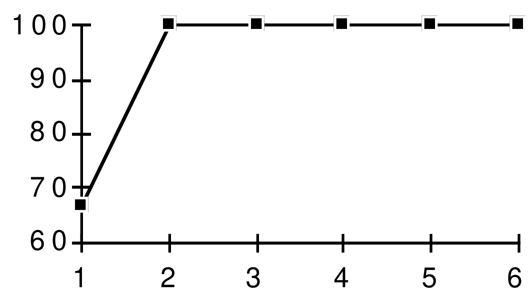


Figure 8.5j. Bottomup.

8.1.6 Monk 2 Data Set

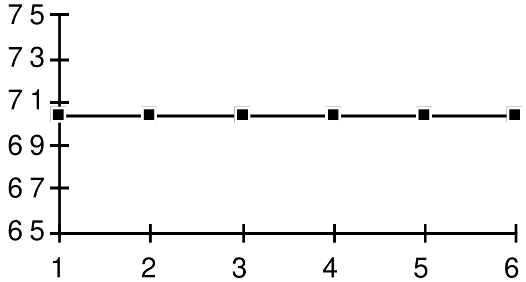


Figure 8.6a. Democracy - Counts.

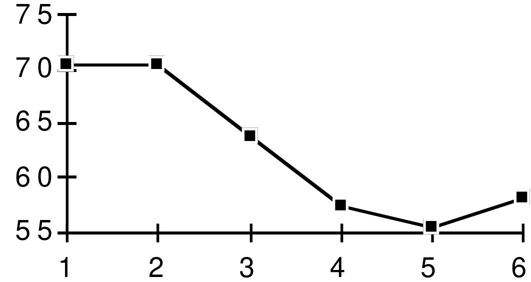


Figure 8.6b. Republic - Counts.

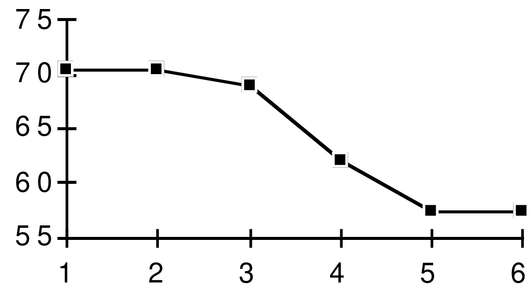


Figure 8.6c. Democracy - One.

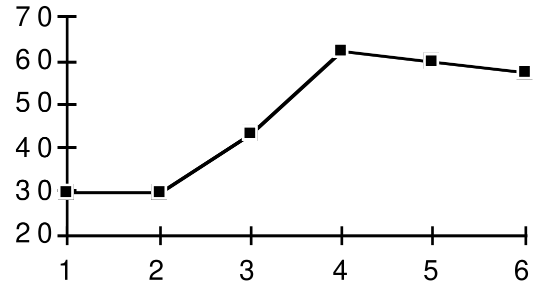


Figure 8.6d. Republic - One.

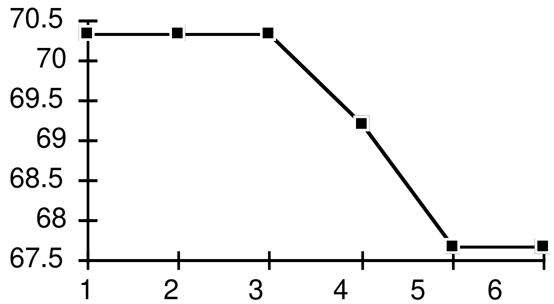


Figure 8.6e. Democracy - $p(o|f)$.

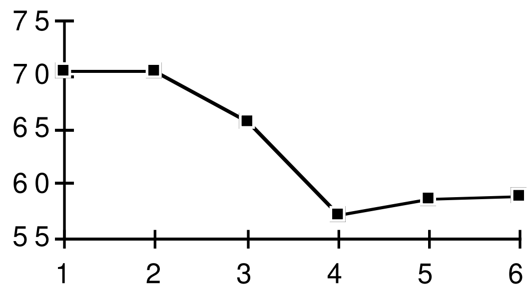


Figure 8.6f. Republic - $p(o|f)$.

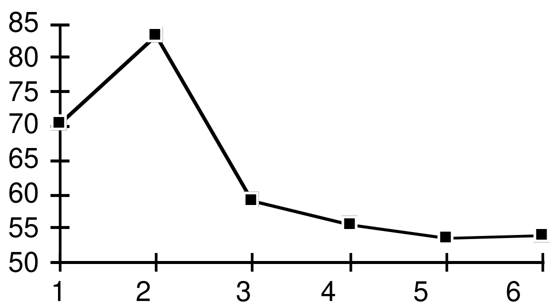


Figure 8.6g. Delta Rule.

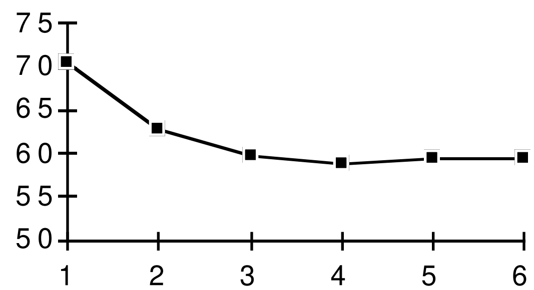


Figure 8.6h. Greedy.

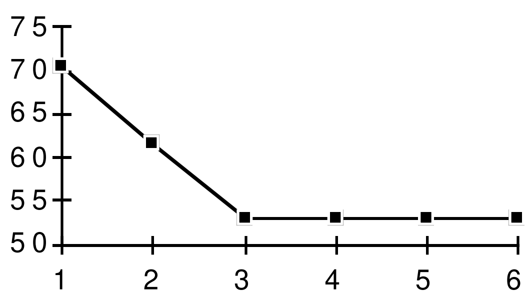


Figure 8.6i. Topdown.

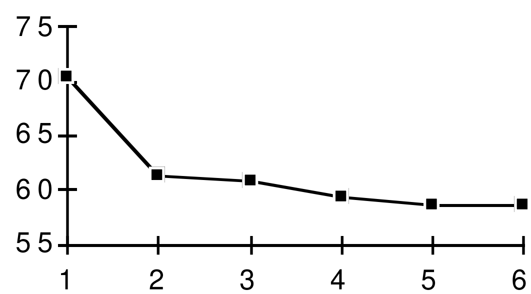


Figure 8.6j. Bottomup.

8.1.7 Monk 3 Data Set

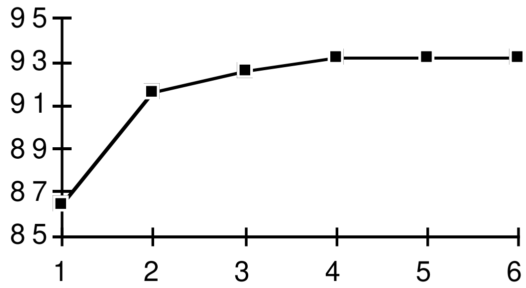


Figure 8.7a. Democracy - Counts.

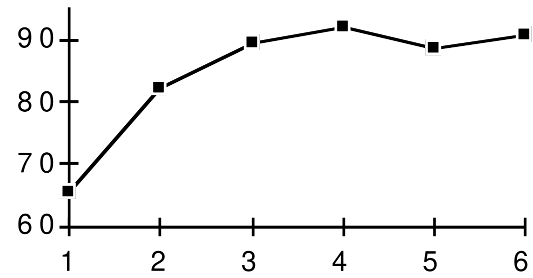


Figure 8.7b. Republic - Counts.

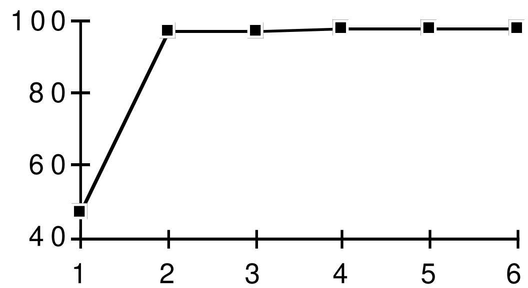


Figure 8.7c. Democracy - One.

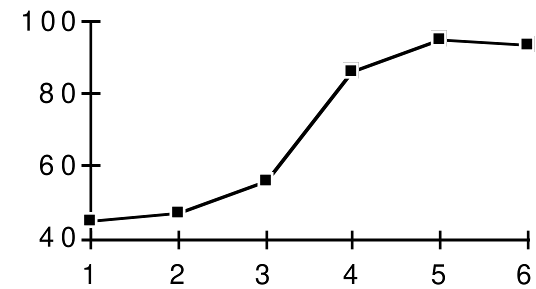


Figure 8.7d. Republic - One.

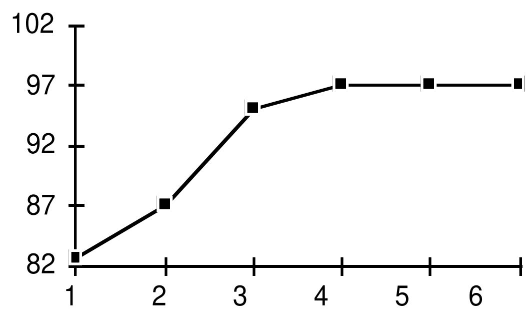


Figure 8.7e. Democracy - p(*o*|*f*).

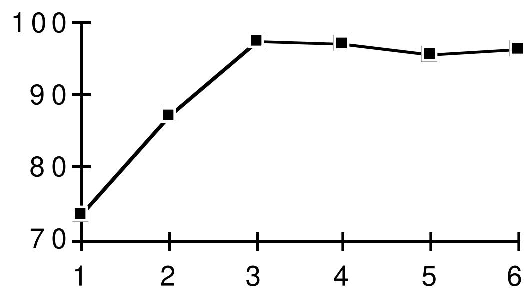


Figure 8.7f. Republic - p(o|f).

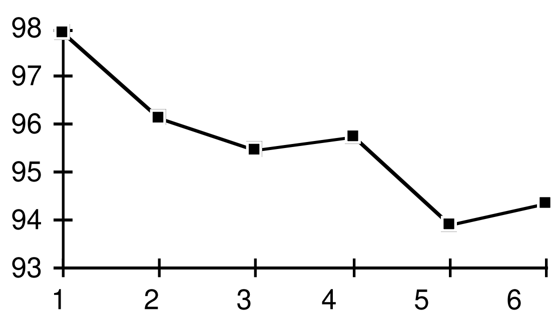


Figure 8.7g. Delta Rule.

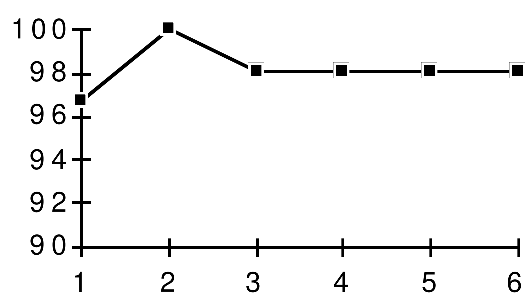
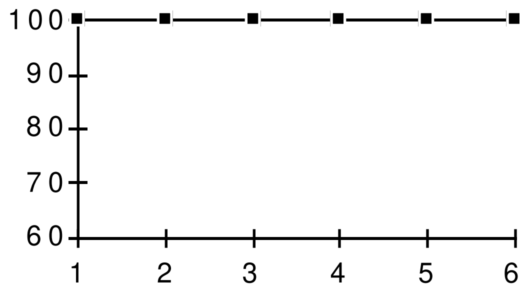


Figure 8.7h. Greedy.



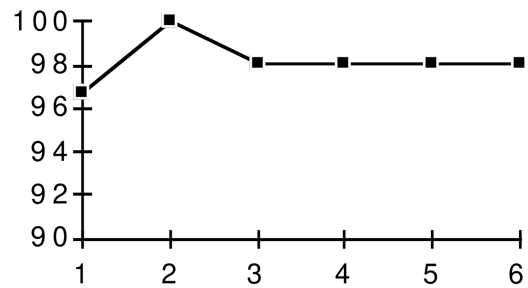
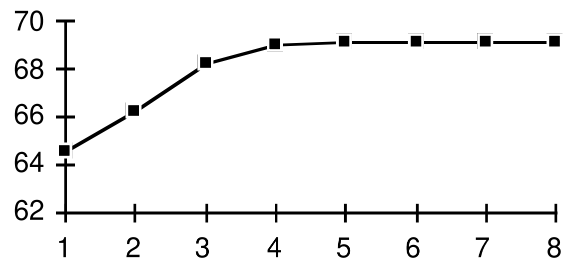


Figure 8.7i. Topdown.

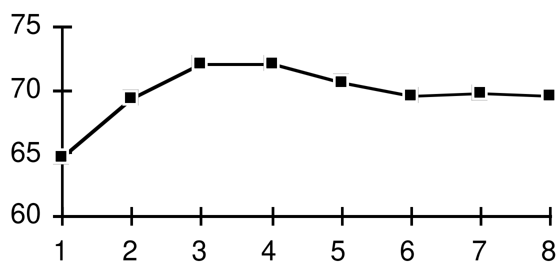
Figure 8.7j. Bottomup.

8.1.8 Pima Data Set



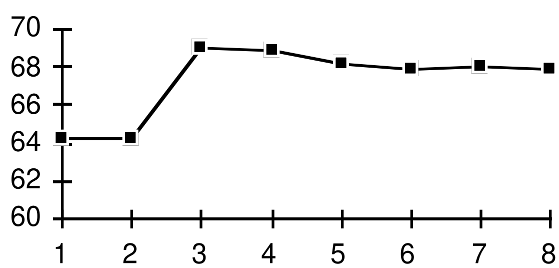
σ=1.9 σ=2.9 σ=2.9 σ=2.7 σ=2.7 σ=2.7 σ=2.7 σ=2.7

Figure 8.8a. Democracy - Counts.



σ=2.1 σ=3.1 σ=2.7 σ=2.4 σ=2.1 σ=1.7 σ=1.6 σ=1.4

Figure 8.8b. Republic - Counts.



σ=1.6 σ=1.6 σ=2.2 σ=2.5 σ=1.8 σ=1.7 σ=1.7 σ=1.6

Figure 8.8c. Democracy - One.

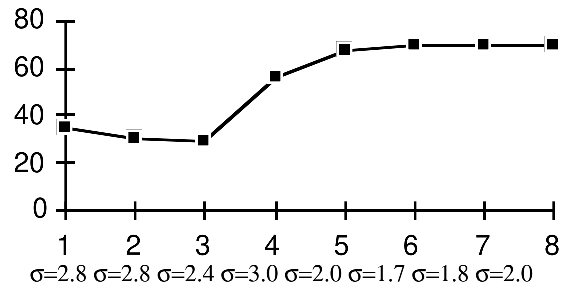


Figure 8.8d. Republic - One.

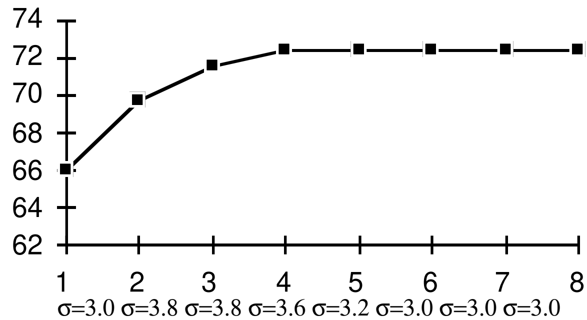


Figure 8.8e. Democracy - $p(o|f)$.

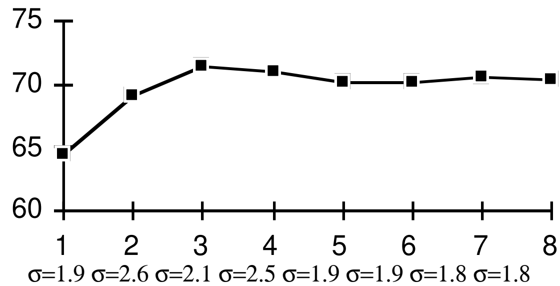


Figure 8.8f. Republic - $p(o|f)$.

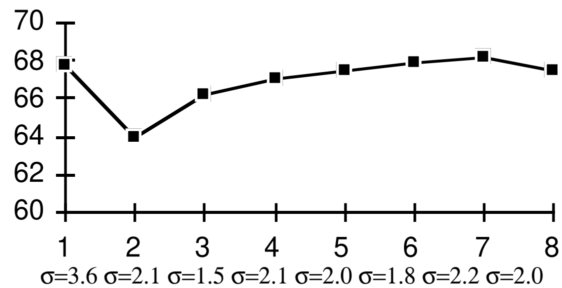


Figure 8.8g. Delta Rule.

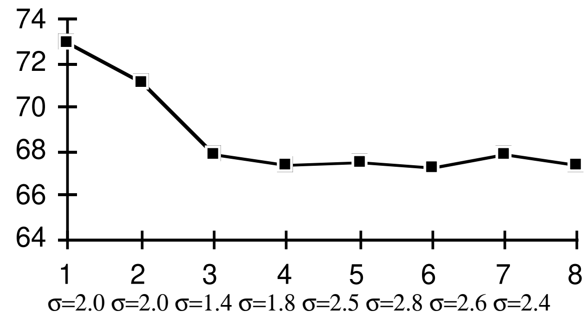


Figure 8.8h. Greedy.

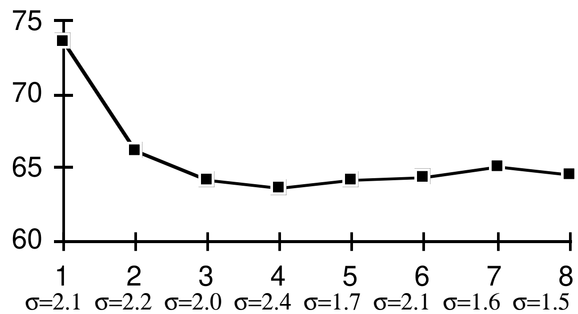


Figure 8.8i. Topdown.

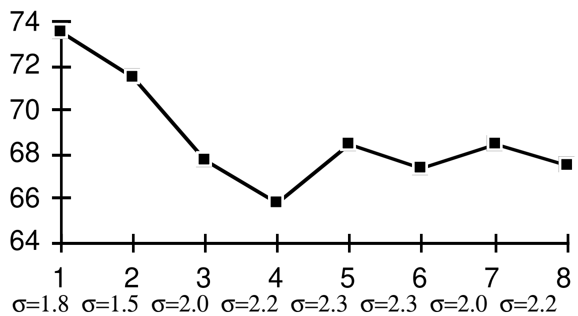


Figure 8.8j. Bottomup.

8.1.9 Tic-tac-toe Data Set

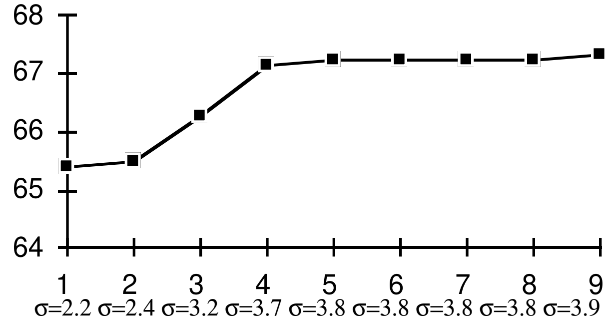


Figure 8.9a. Democracy - Counts.

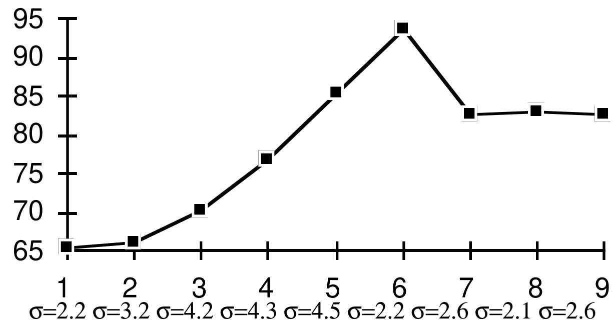


Figure 8.9b. Republic - Counts.

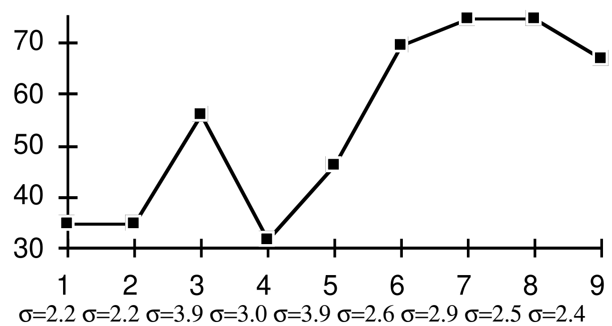


Figure 8.9c. Democracy - one.

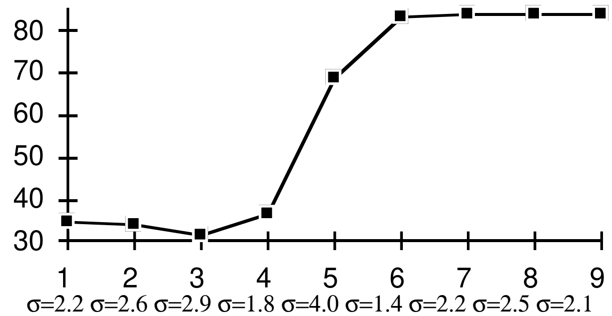


Figure 8.9d. Republic - one.

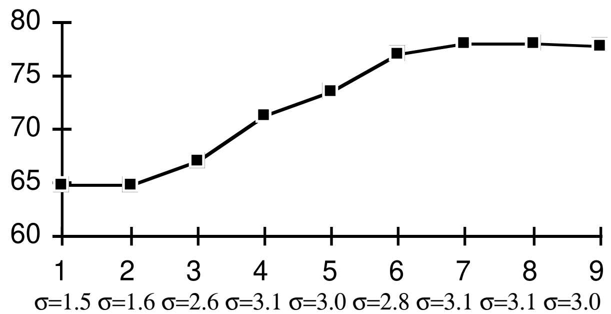


Figure 8.9e. Democracy - $p(o|f)$.

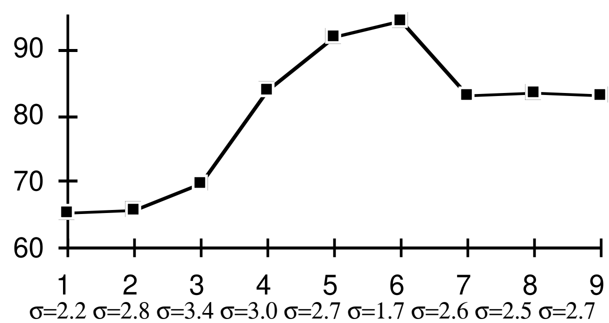


Figure 8.9f. Republic - $p(o|f)$.

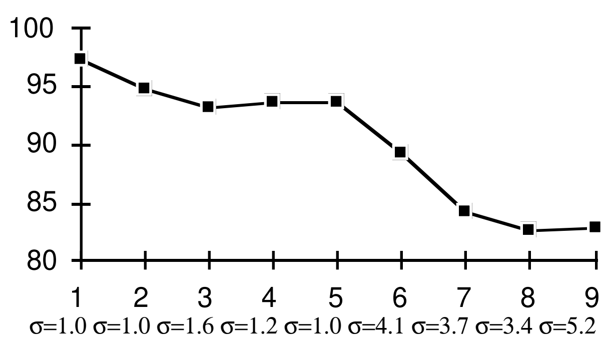


Figure 8.9g. Democracy - Delta Rule.

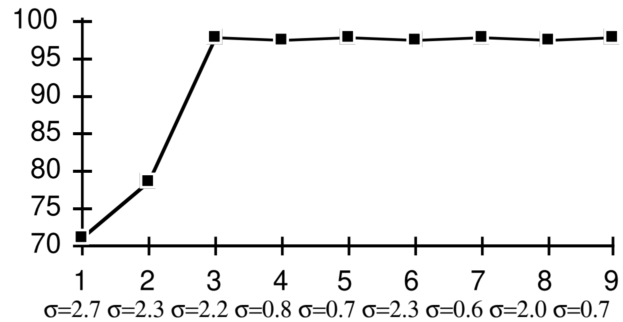


Figure 8.9h. Greedy.

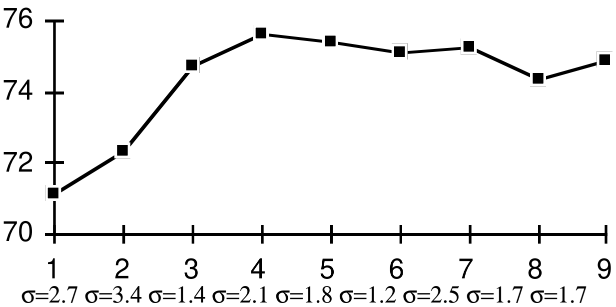


Figure 8.9i. Topdown.

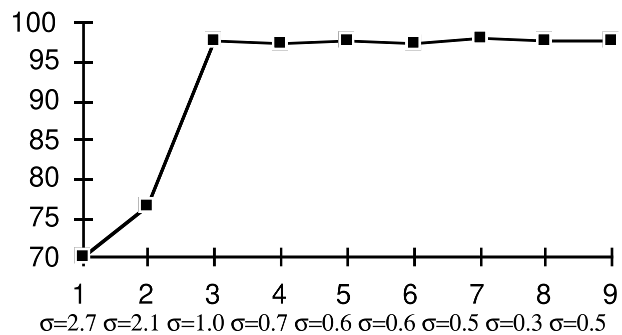


Figure 8.9j. Bottomup.

8.2 Results from C4.5

Figure 8.10a. C4.5 Bupa Indian data results.

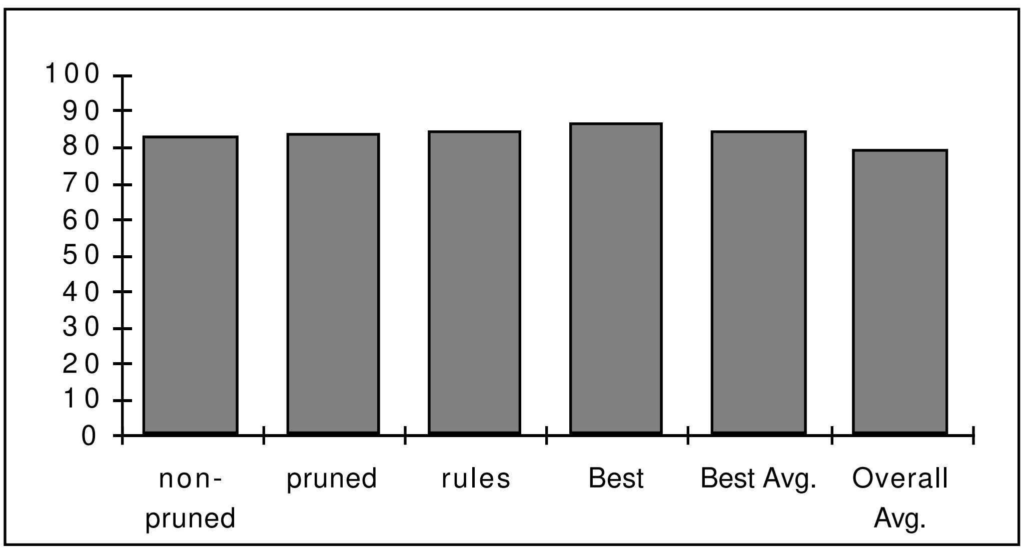


Figure 8.10b. C4.5 CPU data results.

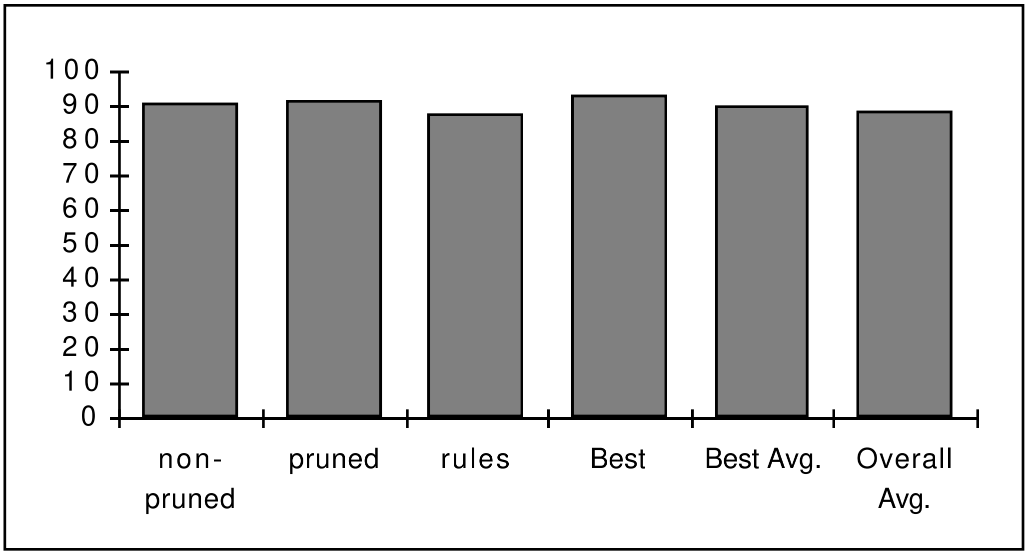


Figure 8.10c. C4.5 Iris data results.

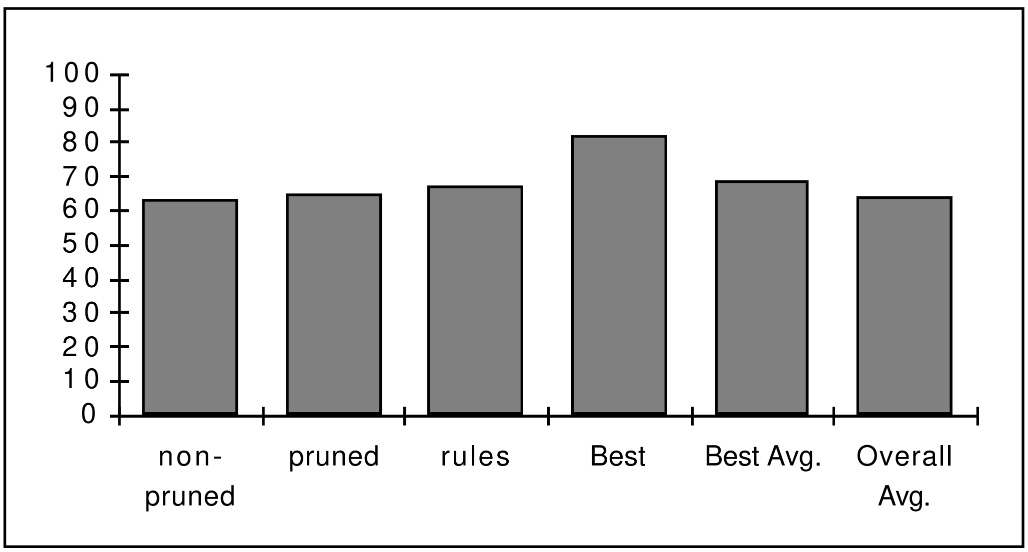


Figure 8.10d. C4.5 Lenses data results.

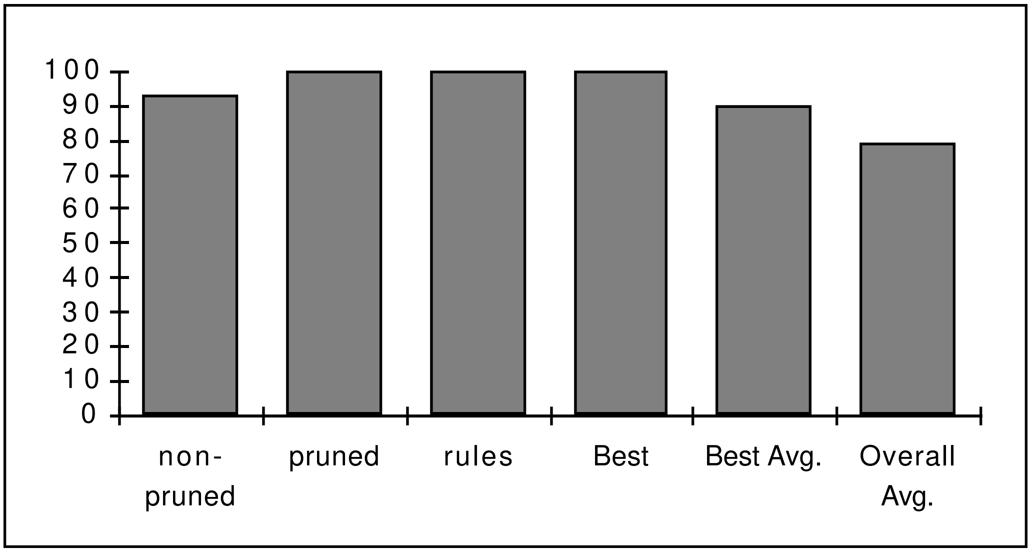


Figure 8.10e. C4.5 Monks-1 data results.

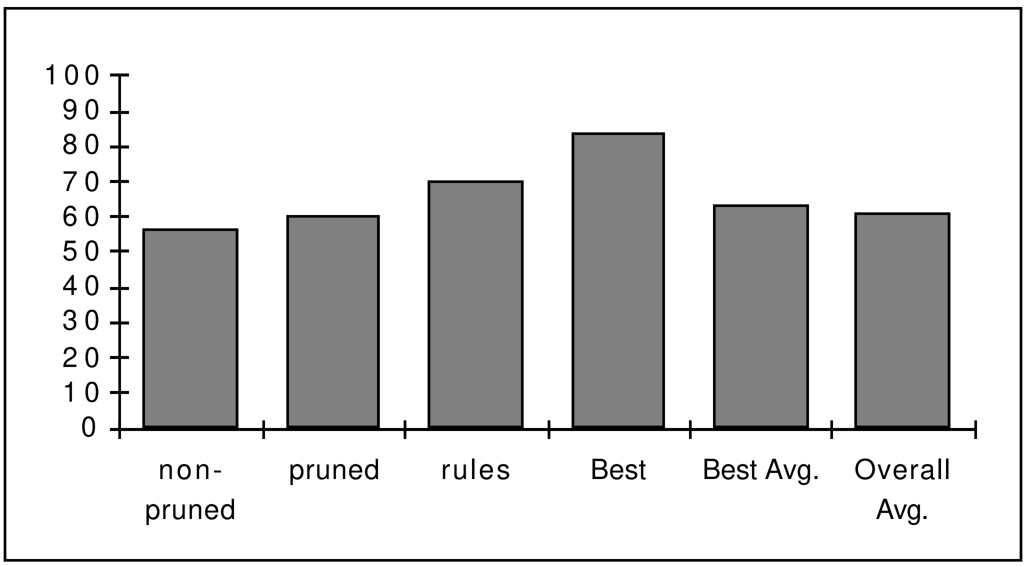


Figure 8.10f. C4.5 Monks-2 data results.

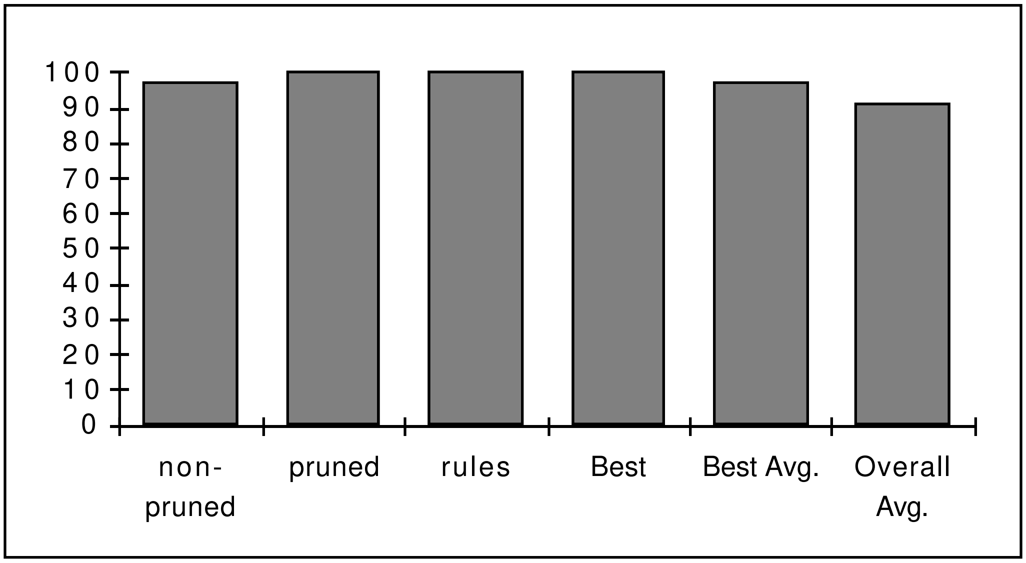


Figure 8.10g. C4.5 Monks-3 data results.

Figure 8.10h. C4.5 Pima data results.

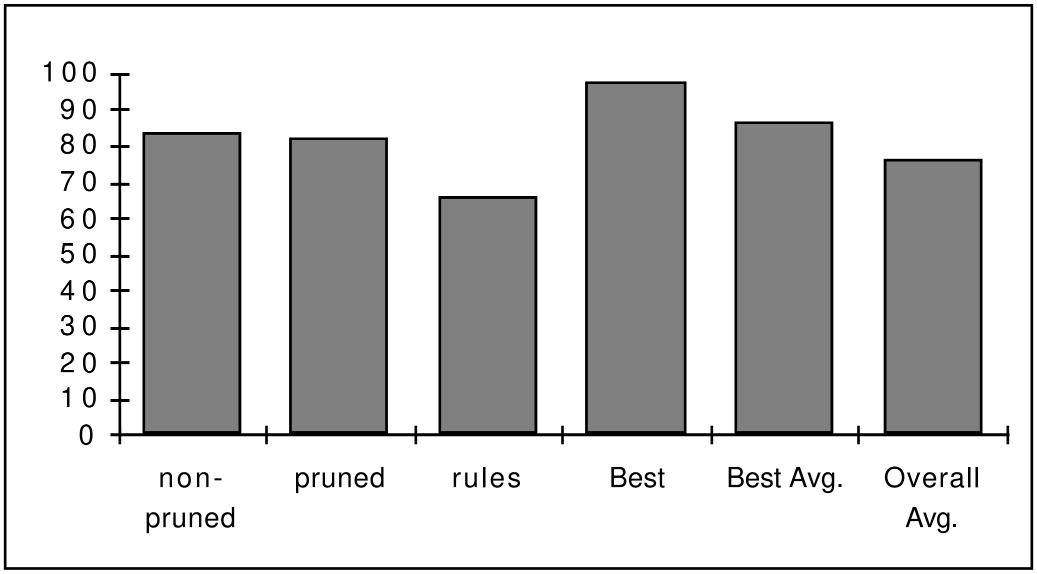


Figure 8.10i. C4.5 Tic-tac-toe data results.

9 Discussion

The following methods consistently returned better results:

- Delta Rule
- Greedy
- Topdown
- Bottomup
- Democracy-p(o|f)
- Republic-Counts

In the discussion that follows, we use the results obtained with these methods and ignore the other results for the sake of simplicity. The last two methods, Democracy-p(o|f) and Republic-Counts, are fairly representative of the simpler metric assignment methods. These simpler methods often did better as the upper bound on rule order was increased. The Delta Rule training method appeared to be the most robust when restricted to low-order rules, and in one case was able to come up with a solution which was much better than any of the other algorithms (on the monk2 data set). The Greedy, Topdown, and Bottomup methods were also good performers, and were able to generate final rule sets which were optimum in the number of rules with some of the data sets (on the monk1 and monk3 data sets). Thus, the results obtained with these methods are representative of all of the results, and restricting the discussion to them does not hinder the goals of this thesis.

9.1 Comparison With C4.5

Table 9.1 compares the best results obtained with C4.5 against the best results obtained with the 6 learning methods listed above. The best result for each data set is shown in bold. The "Average" column contains the average result across all data sets for each method. As can be seen from this table, C4.5 was never able to outperform all of the other learning methods (although it did tie for first place with several of the other algorithms on the monk1 and monk2 data sets). In fact, C4.5 can be seen to be in the lower half performance-wise on most of the data sets.

	Bupa	CPU	Iris	Lenses	Monk1	Monk2	Monk3	Pima	Tic-tac-toe	Average
C4.5	58.9	84.3	91.4	66.7	**100**	70.3	**100**	73.3	83.3	80.9
Delta Rule	**64**	85.6	90.7	70	92	**83.3**	97.9	68.1	97.2	83.2
Greedy	59.6	86.1	92.1	75	**100**	70.3	**100**	73	97.6	**83.7**
Topdown	60.8	80.8	93.2	**81.7**	**100**	70.3	**100**	**73.6**	75.6	81.8
Bottomup	57.9	85.5	89.8	70.6	**100**	70.3	**100**	68.5	**97.9**	82.3
Dem - p(o|f)	61.2	86.3	**93.9**	60	76.6	70.3	97.1	72.1	78	77.3
Rep - Counts	62.2	**86.5**	88	69.5	80.8	70.3	92.3	72	93.7	79.5

Table 9.1. Comparison of learning methods with C4.5.

These results are to be expected, since all of the other methods were allowed to look at all of the possible rules in the search space, and thus had much more information available for generating hypotheses.

A more interesting comparison with C4.5 can be seen in table 9.2. In this figure, the results shown for the other learning methods are restricted to the best results obtained with the search space restricted to low-order rules. In this case, the search space for rules was restricted to rules of order less than or equal to 2.

	Bupa	CPU	Iris	Lenses	Monk1	Monk2	Monk3	Pima	Tic-tac-toe	Average
C4.5	58.9	84.3	91.4	66.7	**100**	70.3	**100**	73.3	83.3	80.9
Delta Rule	**64**	84.1	90.4	70	92	**83.3**	97.9	67.7	**97.2**	**83.0**
Greedy	59.6	**86.1**	88.8	62.8	**100**	70.3	**100**	73	78.5	79.9
Topdown	60.8	80.8	92.2	**73.9**	**100**	70.3	**100**	**73.6**	72.3	80.4
Bottomup	57.7	85.5	89.3	66.7	**100**	70.3	**100**	68.5	76.6	79.4
Dem - p(o|f)	60.1	85.6	**92.6**	56.1	69.2	70.3	87.1	71.8	64.8	73.1
Rep - Counts	62.2	83.5	88	61	61	70.3	82.3	69.3	66.1	71.5

Table 9.2. Comparison with C4.5, rule order ≤ 2.

As can be seen from this table, the C4.5 algorithm was still unable to outperform all of the other learning methods on any given data set, although the relative performance of

C4.5 did increase slightly in comparison with the other learning methods when they were restricted to low-order (≤ 2) rules.

From the standpoint of generalization performance then, the learning methods explored in this thesis were able to do just as well as the C4.5 learning algorithm, exceeding it's performance in several instances, even when the search space was restricted to low-order rules. Since C4.5 is capable of generating higher order rules, and yet was still unable to outperform any of the above learning methods even when they were restricted to low-order rules, this is evidence of the power of these learning methods restricted to low-order rules. In addition to providing empirical evidence of the power of the learning methods explored in this thesis, this is also evidence that the simple heuristic of restricting the rule search space to low-order rules is not unreasonable.

9.2 Utility of Low Order Rules

Further indication of the utility of low-order rules can be seen in table 9.3. This table reports the best results obtained on each data set for each of the generalization methods. For each method, the first result reported is for unbounded order rule sets (the "best" row), and the second result reported is with rule order restricted to 2 or less (the "Order <= 2" row). The numbers in parentheses indicate the actual upper bound on rule order with which the reported result is associated. The winning result for each data set is shown in bold.

		Bupa	CPU	Iris	Lenses	Monk1	Monk2	Monk3	Pima	Tic-tac-toe
Republic - Counts	Best	62.2 (2)	**86.5 (5)**	88 (1)	69.5 (4)	80.8 (6)	70.3 (2)	92.3 (4)	72.0 (4)	93.7 (6)
	Order <= 2	62.2 (2)	83.5 (2)	88 (1)	61.1 (2)	61.0 (2)	70.3 (2)	82.3 (2)	69.3 (2)	66.1 (2)
Democracy - p(o|f)	Best	61.2 (3)	86.3 (4)	**93.9 (3)**	60.0 (3)	76.6 (5)	70.3 (1)	97.1 (4)	72.5 (7)	78.0 (7)
	Order <= 2	60.1 (2)	85.6 (2)	92.6 (2)	56.1 (2)	69.2 (2)	70.3 (1)	87.1 (2)	67.2 (2)	64.8 (2)
Delta Rule	Best	**64 (1)**	85.6 (4)	90.7 (3)	70 (1)	92.0 (2)	**83.3 (2)**	97.9 (1)	68.1 (7)	97.2 (1)
	Order <= 2	**64 (1)**	84.1 (2)	90.4 (2)	70 (1)	92.0 (2)	**83.3 (2)**	97.9 (1)	67.7 (1)	97.2 (1)
Greedy	Best	59.6 (1)	86.1 (2)	92. (4)	75 (3)	**100 (2)**	70.3 (1)	**100 (2)**	73.0 (1)	97.6 (3)
	Order <= 2	59.6 (1)	86.1 (2)	88.8 (2)	62.8 (1)	**100 (2)**	70.3 (1)	**100 (2)**	73.0 (1)	78.5 (2)
Topdown	Best	60.8 (1)	80.8 (1)	93.2 (4)	**81.7 (4)**	**100 (2)**	70.3 (1)	**100 (1)**	**73.6 (1)**	75.6 (4)
	Order <= 2	60.8 (1)	80.8 (1)	92.2 (2)	73. (1)	**100 (2)**	70.3 (1)	**100 (1)**	**73.6 (1)**	72.3 (2)
Bottomup	Best	57.9 (4)	85.5 (2)	89.8 (3)	70.6 (3)	**100 (2)**	70.3 (1)	**100 (2)**	68.5 (1)	**97.9 (7)**
	Order <= 2	57.7 (1)	85.5 (2)	89.3 (2)	66.7 (2)	**100 (2)**	70.3 (1)	**100 (2)**	68.5 (1)	76.6 (2)

Table 9.3. High-order vs low-order rules.

In most cases, the best overall results were obtained with the search space restricted to rules of order less than or equal to 2. The exception to this occurred with the Iris and Lenses data sets. While the Lenses data set seemed to exhibit the need for higher-order rules, the extremely small number of training examples, and the resultant high standard deviations associated with the results reported on this data set, make it difficult to be certain that this is the case. However, it did appear that the Iris data set required higher-order rules in order to achieve the greatest accuracy, although the Democracy-p(o|f) scheme (92.6) was able to come close to the best results (Democracy - $p(o|f)$ 93.9) using only 2nd and 1st order rules.

Even in the case where the best results for a given data set were obtained using high-order rules, such as the results reported on the CPU data set (the Republic-Counts method using 5th order rules won with 86.5 percent accuracy), at least one of the methods was able to come relatively close using only low-order rules (the Greedy method with 2nd order rules achieved 86.1 percent accuracy). In such cases the extra benefit gained by going to high-order rules is probably not enough justification for the large amount of extra work required to get there.

It is interesting to note that the Delta Rule method, which was in general one of the more accurate at predicting classification when restricted to low-order rules, did not benefit appreciably from the addition of higher-order rules. In fact, with several of the data sets the accuracy of the Delta Rule method dropped markedly with the addition of higher-order rules. An example of this can be seen in figure 9.4, which shows the results reported with the Delta Rule method on the Monk2 data set.

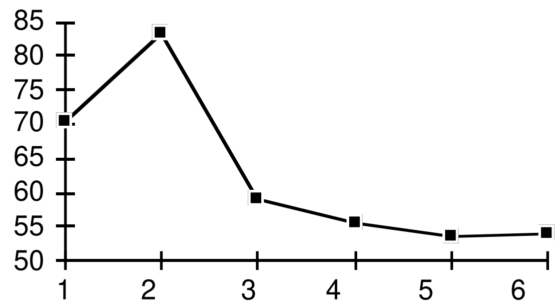


Figure 9.4. Delta Rule.

One reason for this tendency is that the training algorithm tends to favor the less general, higher-order rules. This is because higher-order rules are less likely to match counter examples in the training set, and the training algorithm will thus increment their associated metrics until all of the examples in the training set which they match are classified correctly. Thus, while the algorithm is able to learn the training set with 100 percent accuracy (or near 100 percent), the mapping that it learns does not generalize well on novel instances. In other words, the Delta Rule learning algorithm tends to memorize the training set when presented with higher-order rules.

Another possible reason for this tendency is that the algorithm for finding rules does not yet make any attempt to minimize the set of rules to only those rules that are needed to solve the problem. Rather, all possible rules bounded by the upper bound on rule order are saved and used by the algorithm. Some of these rules are likely to lead the training algorithm astray, causing it to "memorize" the training set rather than learn the desired general concept description. This problem could be alleviated somewhat by first deleting rules which did not pass a statistical significance test before beginning the training procedure.

Other methods were also often led astray by the addition of higher-order rules. For example, figure 9.5 shows results reported on the Pima data set with the Topdown algorithm. In this case, adding higher-order rules decreased generalization performance from the relatively good score of 74% accuracy using only 1st order rules, down to about 65% accuracy.

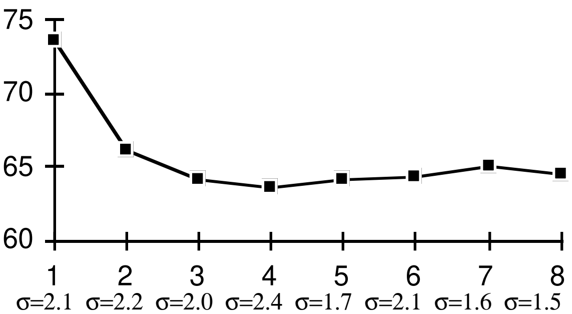


Figure 9.5. Topdown.

In addition, although the simpler metric assignment methods often continued to benefit from the addition of higher and higher order rules, the best results obtained with these methods did not significantly surpass the results obtained with the other methods (restricted to low-order rules) on any of the data sets used in this paper. Even with these methods the benefit of adding higher-order rules tended to level off as the bound on rule order was increased. Figures 9.6 and 9.7, which show the results reported on the Bupa and Monk3 data sets respectively for two different metric assignment methods, are an example of this tendency.

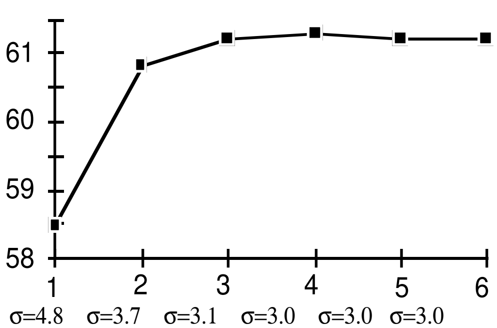


Figure 9.6. Democracy - p($o|f$).

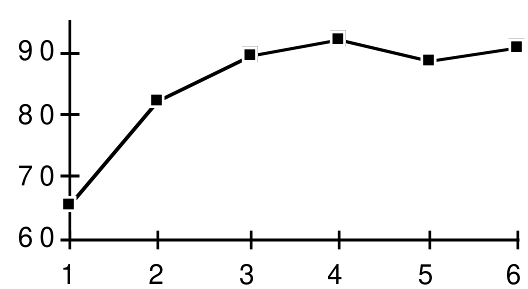


Figure 9.7. Republic - Counts.

The results show that restricting the search space to low-order rules does not adversely affect the over-all ability of these learning methods to generalize accurately on the data sets used in this thesis. Indeed, in some cases the addition of higher-order rules was actually detrimental to the performance of the generalization algorithm. There were, of course, cases where the generalization performance was enhanced by the addition of high-order rules. These tended to occur with the simpler metric assignment methods, which in general did not perform as well as the more complex methods (such as Delta Rule, Greedy, Topdown, etc.). Even so, the increase in generalization performance gained by the addition of higher-order rules often tended to level off with increasing rule order, implying that more and more work had to be done for an increasingly smaller amount of payback.

There are several possible interpretations for the results reported in this thesis. One interpretation is that there was simply not enough data in any of the training sets to reliably estimate the metrics for higher-order rules. Thus, utilizing higher-order rules on these data sets is about as likely to be harmful as it is beneficial. Another possible explanation is that the data sets which were hard to classify (such as Bupa and Pima) are mostly random, negating the benefit of higher-order rules, and the ones which are easy to classify (such as the Iris data) are so regular that higher-order rules do not add much. A third and final explanation could be that the methods used in this thesis for taking advantage of higher-order rules are inadequate, and that with better methods we would be able to demonstrate a significant benefit from the addition of higher-order rules.

None of these explanations is unreasonable, and they are probably all true to some degree. There is certainly not enough data in some of the data sets (the Lenses data, for example) to reliably estimate metric value's for higher-order rules. Many real-world data sets exhibit a great deal of regularity. There are data sets which have several input attributes that are largely uncorrelated with the output. And certainly the current methods for extracting and using higher-order rules have room for improvement. However, rather than being an argument for higher-order rules, all of these explanations can actually be used as arguments against them. If the current learning methods have a difficult time using higher order rules, and the data that we have is not sufficient to reliably extract them, then it is probably a good practice to avoid them, or at least be biased against their usage.

10 Conclusion and Future Research

The results of the experiments were encouraging. It would appear that a bias towards the use of lower-order rule sets can be beneficial on many real world problem domains and warrants further research. Although relatively simple methods were used for rule extraction and metric assignment, surprisingly good results were obtained on all of the data sets using only low-order rules. Assuming that the data sets used in this thesis are representative of most real-world data sets, then an approach to pattern recognition that limits itself (or highly favors) lower-order rules is not unreasonable.

The Delta Rule method exhibited the most robust performance when limited to low-order rules. Using the Delta Rule training method limited to first and second order rules, satisfactory solutions were obtained for all of the learning problems studied in this paper. It is likely that even better results could be obtained with rule sets limited to low-order rules by making improvements to the methods for metric assignment and conflict resolution, and by discarding rules which are not statistically significant.

Other methods, such as the Greedy, Topdown, and Bottomup methods, also exhibited excellent performance on many of the data sets when restricted to low-order rules, and showed a general lack of benefit from the addition of higher-order rules. In addition, all of these methods fared well in comparison tests with the C4.5 tree induction algorithm.

The general lack of benefit from the addition of higher-order rules, and the performance of the learning methods explored in this paper in relation to the results obtained via C4.5, leads to the conclusion that higher-order rules are not necessarily required in order to find a satisfactory solution to many learning problems. And, depending upon the problem to be solved and the learning and generalization methods used, the use of higher-order rules can actually be detrimental to classification accuracy.

From an intuitive standpoint, it would seem that higher-order rules will be more difficult to generate and less effective at generalizing new instances than low-order rules for a variety of reasons. These include the fact that higher-order rules tend to occur less often in the training set, they are less general, and they tend to be more susceptible to noise. The empirical evidence presented in this thesis supports this intuition.

While this thesis has demonstrated that a bias towards low-order rules can be beneficial, we believe that there are some real-world learning problems which will require higher-order rules in order to learn adequately. Also, it is believed that the generalization accuracy of many of the learning algorithms could be improved through statistical tests of rules and through other methods for reducing the size of the final set of rules. Other areas of possible improvement include different metric assignment methods and different methods for resolving conflicts between matched rules. Thus, future research will focus on the following areas:

- Efficient methods for extracting good higher-order rules.
    - Greedy branch and bound algorithms.
    - Coverage of the training set - generating rules as needed to cover the remaining set of mis-classified examples.
    - Set intersection, set difference of example attributes - an n^2 operation in the number of examples.
- Metric assignment methods.
    - Smarter greedy methods which find the optimal metric for the optimal rule at each step (greedier greedy).
    - Smarter greedy list building algorithms which combine aspects of both top down and bottom up list building techniques.
- Rule set minimization methods.
    - Deletion of statistically insignificant rules.
    - Stochastic minimization using training cycles of the Delta Rule algorithm.

- Smarter greedy algorithms for rule set minimization.

11 Bibliography

Afifi, A. A. and Virginia Clark. *Computer Aided Multivariate Analysis.* New York: Van Nostrand Reinhold Company, 1990.

Battiti, Roberto, and Anna Maria Colla. Democracy in Neural Nets: Voting Schemes for Classification. *Neural Networks*; Vol 7, No 4, pp. 691-707, 1994.

Beale, Russell and Tom Jackson. *Neural Computing*. Bristol: IOP Publishing Ltd, 1990.

Booker, L. B., D. E. Goldberg, and J. H. Holland. Classifier Systems and Genetic Algorithms. *Artificial Intelligence*, Vol. 40, 1989, pp. 235-282. North Holland, Amsterdam.

Brazdil, Pavel B. and Kurt Konolige. eds. *Machine Learning, Meta-Reasoning and Logics*. : Kluwer Academic Publishers, 1990.

Breiman, L., J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Belmont, CA: Wadsworth Int., 1983.

Buntine, Wray and Tim Niblett. A Further Comparision of Splitting Rules for Decision-Tree Induction. *Machine Learning*, vol 8, 1992, pp. 75-85.

Chidanand, A., F. Damerau, and M. W. Sholom. Automated Learning of Decision Rules for Text Categorization. *ACM Transactions on Information Systems,* Vol. 12, No. 3, July 1994, pp 233-251.

Clark, P., and T. Niblett. The CN2 Induction Algorithm. *Machine Learning,* Vol. 45, 1990, pp. 304-307.

Dasarathy, Belur V. ed. *Nearest Neighbor Norms: NN Pattern Classification Techniques*. Los Alamitos: IEEE Computer Society Press, 1991.

De Jong, Kenneth, William Spears and Diana Gordon. Using Genetic Algorithms for Concept Learning. *Machine Learning*, Vol 13, pp 161-188, 1993.

Donoho, Steven K. and David C. Wilkins. Using Apprenticeship Techniques to Guide Constructive Induction. *Knowledge Acquisition*, vol 6, pp 295-314, 1994.

Elio, Renee and Larry Watanabe. An Incremental Deductive Strategy for Controlling Constructive Induction in Learning from Examples. *Machine Learning*, vol 7, pp 7-44, 1991.

Fahlman, S. E. and C. Lebiere. The Cascade Correlation Learning Architecture. 1991.

Fukun ga, Keinosuke. *Introduction to Statistical Pattern Recognition*. San Diego: Academic Press, Inc., 1990.

Garey, Michael R. and David S. Johnson. *Computers and Intractability*. New York: W. H. Freeman and Company, 1979.

Greene, D. P. and S. F. Smith. Competition-Based Induction of Decision Models from Examples. *Machine Learning,* 13:229-257, 1993.

Hirsh, Haym. *Incremental Version-Space Merging: A General Framework for Concept Learning*. Boston: Kluwer Academic Publishers, 1990.

Martinez, T. R., "Adaptive Self-Organizing Logic Networks", Ph.D. Dissertation, UCLA Technical Report- CSD 860093, June, 1986.

Martinez, Tony and Jacques Vidal. Adaptive Parallel Logic Networks. *Journal of Parallel and Distributed Computing*. Vol 5, pp 26-58, 1988.

Martinez, Tony R., "Adaptive Self-Organizing Concurrent Systems", *Progress in Neural Networks,* vol. 1, ch. 5, ed. O. Omidvar, Ablex Publishing, pp. 105-26, 1990.

Martinez, T. R. and Campbell, D. M., "A Self-Adjusting Dynamic Logic Module", *Journal of Parallel and Distributed Computing*, v11, no. 4, pp. 303-13, 1991.

Martinez, Tony. ASOCS: Towards Bridging Neural Network and Artificial Intelligence Learning. *2nd Government Neural Network Workshop*, 1991.

Martinez, T. R. and Campbell, D. M., "A Self-Organizing Binary Decision Tree for Incrementally Defined Rule Based Systems", *IEEE Transactions on Systems, Man, and Cybernetics*, v21, no. 5, pp. 1231-8, 1991.

McClelland, James L. and David E. Rumelhart. *Parallel Distributed Processing*. Cambridge: The MIT Press, 1986.

McLachlan, Geoffrey J. *Discriminant Analysis and Statistical Pattern Recognition*. New York: John Wiley & Sons, Inc., 1992.

Michalski, R. S., I. Mozetic, J. Hong, and N. Lavrac. The Multi-purpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains. *Proceedings of the Fifth National Conference on Artificial Intelligence*, pp. 1041-1045, Philadelphia: Morgan Kaufmann, 1986.

Michalski, R. S., and J. Larson. Incremental Generation of VL_1 Hypotheses: The Underlying Methodology and the Description of the Program AQ11. *Technical Report ISG 83-5*, Urbana: University of Illinois, Computer Science Department, 1983.

Mitchell, T. M. Generalization as Search. *Readings in Machine Learning,* pp. 96-107, San Mateo, Morgan Kaufmann, 1990.

Mooney, Raymond, Jude Shavlik, Geoffrey Towell and Alan Gove. An Experimental Comparison of Symbolic and Connectionist Learning Algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, San Mateo, Calif., 1989.

Ozden, G. and W. A. Wallace. Induction of Rules Subject to a Quality Constraint: Probabilistic Inductive Reasoning. *IEEE Transactions on Knowledge and Data Engineering,* Vol 5, No 6, pp 979-984, 1993.

Pagallo, Giulia and David Haussler. Boolean Feature Discovery in Empirical Learning. *Machine Learning* vol 5, pp 71-99, 1990.

Pham, D. and M. Aksoy. RULES: A Simple Rule Extraction System. *Expert Systems with Applications*, Vol. 8, No. 1, pp. 59-65, 1995.

Quinlan, J. R. Induction of Decision Trees. *Machine Learning*, 1:81-106, 1986.

Quinlan, J. R. Generating Production Rules from Decision Trees. *Proc. Int'l Joint Conf. Artificial Intelligence* (IJCAI-87), Morgan Kaufmann, San Mateo, Calif., 1987, pp. 304-307.

Quinlan, R. J. *Programs for Machine Learning.* San Mateo: Morgan Kaufmann Publishers, Inc., 1993.

Raedt, L. D. and Maurice Bruynooghe. Interactive Concept-Learning and Constructive Induction by Analogy. *Machine Learning*, vol 8, pp 107-150, 1992.

Redding, N. J., A. Kowalczyk and T. Downs. Constructive Higher-Order Network Algorithm That Is Polynomial Time. *Neural Networks,* 6:997-1010, 1993.

Romaniuk, S. G. and L. O. Hall. Divde and Conquer Neural Networks. *Neural Networks,* 6:1105-1116, 1993.

Rosenblatt, F. The Perceptron: A probabilistic Model for Information Storage and Organization in the Brain. *Psychological Review*, Vol. 65, No. 6, 1958.

Schaffer, Cullen. A Conservation Law for Generalization Performance.

Schalkoff, Robert. *Pattern Recognition: Statistical, Structural, and Neural Approaches.* New York: John Wiley & Sons, Inc., 1992.

Van Horn, Kevin S and Tony R. Martinez. The BBG Rule Induction Algorithm. *Proceedings of the 6th Autstralian Joint Conference on Artificial Intelligence*. World Scientific Publishing Co., Singapore 1993.

Van Horn, Kevin S and Tony R. Martinez. The Minimum Feature Set Problem. *Neural Networks*, vol 7, no 3, pp 491-494, 1994.

Weiss, Sholom and Ioannis Kapouleas. An Empirical Comparison of Pattern Recognition, Neural Nets, and Machine Learning Classification Methods. *Proceedings of the Tenth International Joint Conference on Artificial INtelligence*. Morgan Kaufmann, San Mateo, CA, 1989.

Weiss, Sholom and N. Indurkhya. Optimized Rule Induction. *IEEE Expert,* December 1993, 61-69.

Wnek, Janusz and Ryszard S. Michalski. Hypothesis-Driven Constructive Induction in AQ17-HCI: A Method and Experiments. *Machine Learning*, vol 14, pp 139-168, 1994.

Wolpert, David H. Combining Generalizers Using Partitions of the Learning Set. *Lectures in Complex Systems*. Addison-Wesley, 1993.

Learning and Generalization with Bounded Order Rule Sets

Tim Andersen

Department of Computer Science

M.S. Degree, April 1995

ABSTRACT

This thesis deals with the problem of inducing useful rules, or extracting critical, higher-order conjunctions of attributes, from a set of preclassified examples, where little or nothing is known about the underlying functional form of the distribution from which the examples were taken. The approach taken in this thesis differs from that normally used in that it does not limit the size of the rule search space. Rather, every possible conjunction of input attributes is considered by the learning algorithm as a potential rule component. In so doing, this thesis is attempting to determine (mainly from an empirical standpoint) how generalization performance is affected when certain areas of the search space are ignored, as compared to when the entire search space is considered. In dealing with the above question, this thesis studies several methods for inducing rules and using them for classification of novel examples. This thesis also uses results obtained with the C4.5 rule-induction method for comparison purposes, and to support the main points of the thesis. The results show that higher-order rules are not required to approximate many real world learning problems. In addition, the difficulty of generating optimal rule sets is discussed, where the measure of optimality is the complexity or size of the rule set and/or the degree of predictive accuracy, and the problem of NP-completeness is discussed in relation to these two optimality measures.

COMMITTEE APPROVAL:

____
Tony R. Martinez, Committee Chairman

____
Theodore A. Norman, Committee Member

____
William A. Barrett, Committee Member

____
David W. Embley, Graduate Coordinator