Proceedings of the 10th International Symposium on Computer and Information Sciences, pp. 419-426, 1995.

Learning and Generalization with Bounded Order Rule Sets

Tim L. Andersen and Tony R. Martinez

Computer Science Department, Brigham Young University, Provo, Utah 84602
e-mail: tim@axon.cs.byu.edu, martinez@cs.byu.edu

ABSTRACT

All current rule-based methods found in the literature use some form of heuristic(s) in order to limit the size of the rule search space examined by the learning algorithm. This paper is an attempt to determine (mainly from an empirical standpoint) how generalization performance is affected when certain areas of the rule search space are ignored, as compared to when the entire search space is considered. This is done by exhaustively generating all rules for several small real-world problems and then determining how accuracy decreases as the size of the search space is iteratively reduced. The results show that higher-order rules are not required to approximate many real world learning problems. In dealing with the above question, several methods for inducing rules and using them for classification of novel examples are tested.

Key Words: Rule Induction, Machine Learning.

1 Introduction

The problem of rule induction has been extensively studied in the literature (Elio '91; Pagallo '90; Wnek '94; Quinlan '86; Raedt '92; Donoho '94; Michalski '86; Clark '90). Since the number of possible rules for any given learning problem is exponential in the number of input attributes, all of the rule-based methods found in the literature use some form of heuristic(s) in order to limit the size of the rule search space examined by the learning algorithm.

The main goal of this paper is to demonstrate, mainly from an empirical standpoint, the degree (lack) of benefit gained when utilizing higher-order rules for classification and decision making tasks. For the purposes of this paper it is assumed that rules are stored in disjunctive normal form (DNF). The order of each rule is then defined to be the number of attributes in the rule's conjunction. The results of the research show that many real world classification problems can be solved using only relatively low-order (order ≤ 2) rules, and that the use of higher-order rules does not appear to significantly increase classification performance, and can often be detrimental to generalization accuracy. These results justify a search bias which favors lower-order, more general rules, allowing for a vast reduction in the rule search space.

In order to demonstrate the utility of low-order rules, the learning algorithm is first run using the set of all possible rules, where the rules are in DNF. In other words, we exhaustively generate all attribute conjunctions and the learning algorithm then tests each conjunction to determine if it will be included as part of a rule in the final set of rules. Obviously, it is generally not possible to generate the set of all possible rules for data sets which have a large number of input attributes, and so this does not constitute a realistic learning method. For this paper we thus limit the choice of data sets to those which have relatively few inputs, and for which it is possible to generate 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 (the number of terms in each rule's conjunction), and the learning algorithm is ran using this bounded-order set of rules. This continues until all possible upper bounds on rule order have been tested. The results are then analyzed and compared with results obtained with the C4.5 tree induction algorithm (Quinlan '86).

Several methods for using rules for classification are developed and studied. The rule-based classification methods analyzed in this paper are based upon several variations on voting, where each rule votes according its associated metric (a relative strength indicator). The different variations on voting are:

- Democracy - all matching rules are allowed to vote.
- Monarchy - the matching rule with the greatest metric wins.
- Republic - only the most specific matching rules are allowed to vote.

The different methods used to assign metrics to rules are:

- Count - number of times the rule matched an example from the training set.
- p(o|f) - probability of output *o* given that the rule is matched.
- Delta Rule - rule metrics assigned by iterative training procedure.
- Topdown - rule metrics ordered in a top down list building approach.
- Bottomup - rule metrics ordered in a bottom up list building approach.
- Greedy - greedy assignment via a training algorithm similar to Delta Rule.

A more complete and formal definition of these methods is given in section 2. The results of the research are then reported in section 3. Finally, conclusion and discussion are given in section 4.

2 Methods and Approach

It is recognized that there are a number of interdependent factors which could influence the outcome of the research. For example, the choice of learning method could have an affect on how much benefit was observed when utilizing higher-order rules. However, the main goal of the research was not to find the optimum learning method. Rather, the goal of the research was to use reasonable methods which compare favorably with existing rule induction algorithms, 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. It is felt that the methods used here are sufficient to demonstrate the main points of this paper. The methods used in this paper are formally defined in sections 2.1 and 2.2.

2.1 Conflict Resolution

A conflict occurs whenever an example is matched by at least two rules which predict different output classifications. In this case, some method must be implemented for resolving the conflict between the matched rules. The three methods for conflict resolution which are used in this paper are called Democracy, Monarchy, and Republic. These are essentially variations on rule voting, and are defined below.

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 paper 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.

Given the following: a problem domain D, a set E of examples taken from D, a set F of rules defined on D, for each $e \in E$, choose the predicted output class o which maximizes the summation (ties being resolved arbitrarily):

$$\sum_{f \in F \wedge match(f,e) \wedge f.class=o} f.metric \quad .$$

Figure 3.1. Democracy voting algorithm.

Monarchy: The single greatest rule voting method, or Monarchy 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.

For each example $e \in E$, define F_e to be the set of rules from F which match e. In other words, let $F_e = \{f | f \in F \wedge match(f,e)\}$. Then for all $e \in E$, if $F_e = \varnothing$ the predicted output classification is undefined, else choose the predicted classification to be $f.class$, where the following is true about the rule f (ties are resolved arbitrarily) $(f \in F_e) \wedge (\forall y)(y \in F_e \rightarrow f.metric \geq y.metric)$.

Figure 3.2. Monarchy voting algorithm.

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 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. 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.

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

$$\sum_{f \in F \wedge match(f,e) \wedge f.class = o \wedge (\forall y)(y \in F \wedge match(y,e) \rightarrow y \text{ is not more specific than } f)} f.metric \; .$$

Figure 3.3. Republic voting method.

2.2 Metric Assignment

The above conflict resolution methods require that each rule be assigned a metric value. The six methods used to assign metrics are called Count, p($o|f$), Delta Rule, Greedy, Topdown, and Bottomup. These are defined as follows.

Count: This method counts the number of times a rule matches examples from the training set that have the same output class as the rule, and then sets the rule's metric equal to this number. 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. Formally, for each $f \in F$, $f.metric = |\{e | e \in E \wedge e.class = f.class \wedge match(f,e)\}|$

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. Formally, for each $f \in F$,

$$f.metric = \frac{|\{e | e \in E \wedge e.class = f.class \wedge match(f,e)\}|}{|\{e | e \in E \wedge match(f,e)\}|}.$$

Delta Rule: With this method, an algorithm similar to the simple perceptron training algorithm (Rosenblatt 1958) is used to assign metrics to rules. This algorithm has the advantage that it is guaranteed to converge to a solution if the elements in the training set are linearly separable. While there are many training sets which do not exhibit linear separability, the constructive induction 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. 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.

while $(\exists e)(e \in E \wedge vote(e, F) \neq e.class)$
for each $e \in E$ do
if $vote(e, F) \neq e.class$ do
let $W = \{ f | f \in F \wedge match(f, e) \wedge f.class \neq e.class \wedge vote(e, F) = f.class \}$
let $V = \{ f | f \in F \wedge match(f, e) \wedge f.class = e.class \}$
for each $f \in W$ let $f.metric = f.metric - c$
for each $f \in V$ let $f.metric = f.metric + c$

Figure 3.4. Formal Delta Rule training method.

Greedy Training Methods: This paper 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 o 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.
- Since the algorithm terminates when no gain can be made in classification performance, the maximum number of steps which the algorithm will take, and tl the maximum number of rules contained in *OPT*, is bounded above by $|E|$, where E is the set of training examples.
- The algorithm naturally minimizes the size of the set *OPT*.

Greedy: 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 *OPT* and *POOL* are evaluated, and each rule is assigned a score which is based upon how much that rule would increase classification performance if its metric was set to *opt_met*. 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. 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 simply increased to *opt_met*).

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. 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*.

Topdown: With the Topdown method, all examples are initially 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.

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*. A rule is said to be positive in relation to an example if it matches the example and predicts the same output as the example. 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 rule with the largest score to add to the bottom of the list. In the case where the scores of two or more rules are equal, 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.

Bottomup: The main difference between the Bottomup approach and the Topdown approach is that Bottomup adds rules to the top of the list (as it's name implies).

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.

At each step of the algorithm, 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. 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. This process continues until no further gain can be made in classification accuracy.

3 Results and Discussion

Several data sets taken from real world problems were selected for testing. All of these data sets were obtained from the UC Irvine Machine Learning Database. Since the learning algorithms explored in this paper require discrete valued attributes, all of the continuous valued attributes were discretized utilizing a simple equal interval discretization technique. In addition to the real world data sets, 3 artificial data sets, called the Monk data sets, are also used.

For each run, the 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. 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 on the test set is reported.

We first compare the results with those obtained using the C4.5 tree induction and rule generation algorithms. 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 Delta Rule training method did relatively well in comparison with C4.5. A comparison is then made with the results obtained using only low-order rules versus the results obtained with high-order rules.

3.1 Comparison With C4.5

Table 4.1 compares the best results obtained with C4.5 against the best results obtained with the 6 learning methods listed above. The last two rows in the table, Dem-p(o|f) and Rep-Counts, are (respectively) the results obtained with the Democracy conflict resolution method in conjunction with the p(o|f) method for metric assignment, and the Republic conflict resolution method in conjunction with the Counts method for metric assignment. 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 4.1. 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 4.2. 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 4.2. 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. From the standpoint of generalization performance then, the learning methods explored in this paper 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.

3.2 Utility of Low Order Rules

		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 4.3. High-order vs low-order rules.

Further indication of the utility of low-order rules can be seen in table 4.3. 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.

In most cases, the best overall results were obtained with the search space restricted to rules of order less than or equal to 2. 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. 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 4.4, which shows the results obtained with the Delta Rule method on the Monk2 data set.

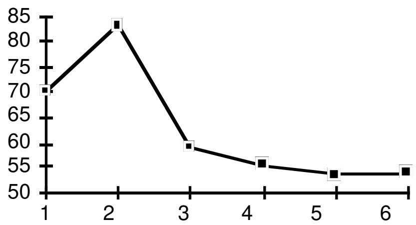

Figure 4.4. Delta Rule.

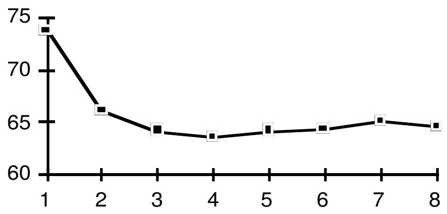

Figure 4.5. Topdown.

Other methods were also often led astray by the addition of higher-order rules. For example, figure 4.5 shows results 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.

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 4.6 and 4.7, which show the results on the Bupa and Monk3 data sets respectively for two different metric assignment methods, are an example of this tendency.

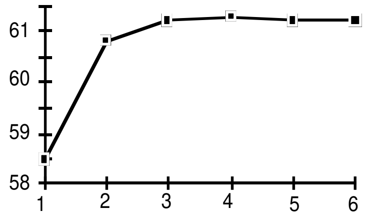

Figure 4.6. Democracy - p($o|f$).

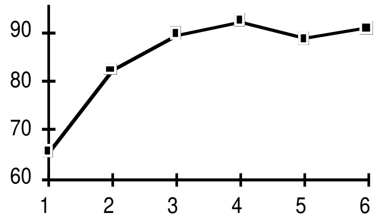

Figure 4.7. Republic - Counts.

4 Conclusion and Future Research

There are several reasons why the restriction of the search space to lower-order rules is not 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 they contain, they offer themselves as ready candidates for exclusion.

Another reason for favoring lower-order DNF rules is that they are by nature more general. 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, which makes it 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. 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.

Thus, 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. The empirical evidence presented in this paper supports this intuition. Indeed, in some cases the addition of higher-order rules was actually detrimental to the performance of several of the learning algorithms.

There are, of course, several other possible interpretations for the results reported in this paper. 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. Another possible explanation is that the data sets which were hard to classify are mostly random, negating the benefit of higher-order rules, and the ones which are easy to classify are so regular that higher-order rules do not add much. A third and final explanation could be that the learning methods used in this paper are inadequate, and that with better methods we would be able to demonstrate a significant benefit from the addition of higher-order rules.

All of these interpretations are probably true to some degree. 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.

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.

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, but we also believe that such rules must be used carefully. Areas of possible improvement include better metric assignment methods, methods for rule set minimization, different methods for resolving conflicts between matched rules, and efficient methods for extracting critical rules. Thus, future research will focus on the following areas:

- Developing efficient methods for extracting good higher-order rules - branch and bound algorithms.
- Robust metric assignment methods. For example, methods which combine aspects of the Greedy and Delta Rule training methods.
- Rule set minimization methods, statistical measures, greedy methods.

6 Bibliography

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

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.

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.

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

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

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

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

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.