Learning Classification Rules Using Lattices

Mehran Sahami

Robotics Laboratory
Department of Computer Science
Stanford University
Stanford, CA 94305
Telephone: USA (415) 725-8784
Fax: USA (415) 725-1449
sahami@cs.Stanford.EDU

Abstract

This paper presents a novel induction algorithm, Rulearner, which induces classification rules using a Galois lattice as an explicit map through the search space of rules. The construction of lattices from data is initially discussed and the use of these structures in inducing classification rules is examined. The Rulearner system is shown to compare favorably with commonly used symbolic learning methods which use heursitics rather than an explicit map to guide their search through the rule space. Furthermore, our learning system is shown to be robust in the presence of noisy data. The Rulearner system is also capable of learning both decision lists as well as unordered rule sets and thus allows for comparisons of these different learning paradigms within the same algorithmic framework.

Research Area: inductive learning

Keywords: lattices, decision lists, rule induction

1 Introduction

Research in rule induction by means of search [Michalski, 1969; Mitchell, 1982; Clark & Niblett, 1989] has been on-going for some time. Such search methods generally attempt to find rules which can either be further specialized or generalized to fit a given data set. While some of these systems, such as Version Spaces [Mitchell, 1982], make direct use of the data to be learned during rule induction, such methods are highly sensitive to noisy data and quickly degrade in performance in these cases. In other systems, the search through the space of possible rules is generally guided by heuristics aimed at finding a good fit to the data as opposed to using the data to build an explicit map through the space of possible

rules to induce. In this capacity, such algorithms are only *data-driven* to the extent that the heuristics employed in them make use of the data to be learned.

We present the Rulearner system which seeks to combine both the direct use of data with robustness in the presence of noise during the rule induction process. Since the algorithm uses a Galois lattice constructed from training data [Oosthuizen, 1988; Oosthuizen & McGregor, 1988] as an explicit guide through the rule space, the algorithm is directly *data-driven* as it does not simply heuristically fit the training data, but actually uses commonalties inherent in the data to induce classification rules. Furthermore, it can readily be proven that for any given set of instances a unique lattice will be constructed [Kourie *et al.*, 1992] so that the induction performed using such a lattice will not suffer from ordering effects of the training data. Moreover, our system is capable of inducing both an unordered (and non-overlapping) set of classification rules as well as an ordered set of rules, also called a decision list [Rivest, 1987]. This allows for the Rulearner system to be used in making direct comparisons between these two learning paradigms within a single algorithmic framework. Lastly, it is also possible to augment a lattice constructed from training data with domain knowledge [Oosthuizen, 1988] prior to rule induction with Rulearner, but this enhancement is beyond the scope of this paper.

Several experiments with the Rulearner system are presented comparing it with the commonly used symbolic learning systems C4.5 [Quinlan, 1993] and CN2 [Clark & Niblett, 1989] on the standard machine learning test-bed of the MONK'S Problems [Thrun *et al.*, 1991] as well as the real world domain of the Yugoslavian Breast Cancer data set [Kononenko *et al.*, 1984; Oosthuizen, 1993]. We chose to compare Rulearner with these algorithms (as opposed to the other lattice based algorithms described in the later section on related work) as they are the most widely used (and often most effective) induction algorithms for decision trees and decision lists respectively, thus allowing us to examine the performance of our algorithm within both of these frameworks against successfully fielded systems.

2 Lattice Construction[1]

A lattice is defined to be a directed acyclic graph in which any two nodes, *u* and *v*, have a unique *join* (a node "higher" in the graph to which a *u* and *v* are connected by minimal length paths) and a unique *meet* (a node "lower" in the graph which is connected to *u* and *v* by minimal length paths) — referred to respectively as *least upper bounds* and *greatest lower bounds* in formal mathematics. In order to construct such a lattice, we begin by first creating a single node for every possible feature that an instance in the training set may have. We refer to these as *feature nodes*. Then for each instance vector in the training set, we create a new node (referred to as an *instance node*) and connect this node to the corresponding feature nodes for every feature that is present in the instance. While these connections are being made, it is important to make sure that the "lattice property" is maintained (that is that every node has a unique join and a unique meet). For example, as seen in Figure 1, if we connect the instance node ABC and the instance node ABD to their respective feature nodes, we will violate the lattice property as the two instance nodes no longer have a unique join (both nodes A and B fit the definition of the join of the two instances). Therefore, we must create a new node (called an *internal node*), representing the feature combination AB, as shown in Figure 2, and make the appropriate connections with regards to this node. Now the lattice property is once again restored as AB is the unique join of ABC and ABD. Note that in a proper lattice, we also require a node at the very top of the lattice (to which all the feature nodes are connected) and a node at the very

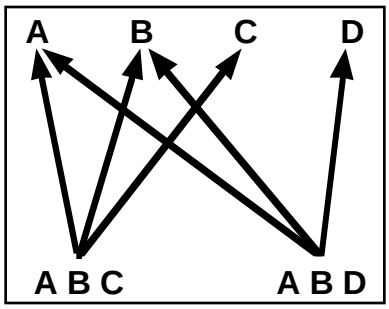

Figure 1

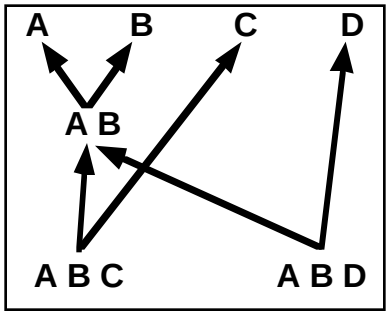

Figure 2

[1]The lattice construction algorithm given here is based on [Oosthuizen, 1988].

bottom of the lattice (which is connected to all the instance nodes), but these nodes are left off in our discussion for the sake of brevity.

The process of adding instance nodes to the lattice and restoring the lattice property is repeated until all instances in the training set have been added to the lattice. Thus the generation of such a lattice is simply a straight-forward mechanical process. It is very important to note (and it can be easily proven but is beyond the scope of this paper) that a unique lattice will always be generated from any given set of data. The order of adding instance nodes to the lattice during its construction will not effect the final resultant lattice as the same unique lattice structure will always be constructed from the same data set. In our work, all lattices were generated as explained above using the GRAND program [Oosthuizen, 1988].

We should also point out some useful definitions in regard to the lattice structure:

Definition: The *upward closure* of node *u* in the lattice, denoted UC(*u*), is defined to be the set of all nodes that can be reached from *u* following upward arcs in the lattice, including the node *u* itself. Thus, UC(ABC) in Figure 2 is: {ABC, AB, A, B, C}.

Definition: The *downward closure* of a node *u*, denoted DC(*u*), is defined to be the set of all nodes which contain *u* in their upward closure. So, DC(B) in Figure 2 is: {B, AB, ABC, ABD}.

Definition: The *cover* of a node *u*, denoted Cover(*u*), is the number of instance nodes in DC(*u*). For example, Cover(A) is 2 in Figure 2.

These definitions will prove useful in the next section which explains how the lattice is used to perform rule induction in the Rulearner system.

3 The Rulearner Algorithm

The Rulearner algorithm takes as input a lattice, L, and a set of instance classification labelings, C, which correspond to the instance nodes in L. The algorithm produces a set of symbolic classification rules as output. The user is also able to specify a noise parameter to the system as a percentage figure by which each induced rule can misclassify some portion of the training instances that it applies to. Furthermore, the user can configure the system to induce either a decision-list or an unordered set of rules, and can also decide whether the classification rules induced should only identify one given labeling (i.e. derive only rules with a "positive" labeling) or if the rules should attempt to classify all instances in the training set. While the basic framework of the algorithm is general enough to properly deal with more than two classification labelings, we present the algorithm here as a binary classifier for easier understanding.

```
PROCEDURE FindRules(L, C)
BEGIN
  Initialize all nodes in the lattice to be "active"
  Label all nodes in the lattice (using the noise parameter)
  WHILE (there are still "active" nodes in the lattice)
  BEGIN
    u ← node with most instances in DC and non-MIXED labeling
    MakeRuleFromNode(L, C, u)
    FORALL (v ∈ DC(u) where v is an instance node)
    BEGIN
      FORALL (w ∈ UC(v))
      BEGIN
        Decrement the cover of w
        IF (cover of w ≤ 0)
        THEN mark w "inactive"
      END
      Mark v as "inactive"
    END
    IF (Decision-List)
    THEN re-label "active" nodes
  END
END

PROCEDURE MakeRuleFromNode(L, C, u)
BEGIN
  antecedent ← feature nodes in UC(u)
  label ← label of node u
  Produce rule "antecedent implies label"
END
```

Table 1: Pseudo-code for the Rulearner algorithm. Note that L is the input lattice and C is the classification labelings for the instances which were used in building L.

The algorithm begins by initializing all nodes in the lattice to be "active," indicating that all nodes begin as possible candidates for rule induction. Then all nodes in the lattice are labeled, using the noise parameter if applicable. The labeling process involves first labeling all instance nodes (whose labelings are given in C) and then filtering these labelings up the lattice. A node is given a particular label, let's say POSITIVE, if the instance nodes in its downward closure are all of the class POSITIVE (allowing also for a certain percentage of mislabelings given by the noise parameter). If there are insufficient instance nodes of any given class in the downward closure of a node to give that node a particular label, then the node is labeled MIXED. This process is carried out for every node in the lattice. After this initialization phase, we begin the rule induction process.

As long as "active" nodes remain in the lattice, there are still candidate nodes for rule induction and hence a new rule is induced. We identify the node in the lattice which has the most instances in its downward closure and has a non-MIXED labeling. If several nodes have the same number of instances in their downward closure, we prefer the node which has the fewest features in its upward closure — an application of Occam's Razor. We then produce a rule from this node by having all the feature nodes in the upward closure of the node imply the labeling that the node has. Intuitively, this corresponds to finding a minimal set of features that covers a large portion of the instance space with a given labeling. An important factor in this minimal set of features is that it is directly derived from commonalities in the underlying data (which was used to build the lattice) and hence the antecedent of the rule induced is directly driven by the data in the training set.

After such a rule is derived, we must then update the appropriate nodes in the lattice so that we can effectively record that a given portion of the instance space has been classified. We do so by examining each instance node v in the downward closure of the node u we just used to derive a rule from. For every such node v, we decrement the cover of each node in UC(v) by one to effectively denote that the instances covered by the most recently derived rule are no longer of concern to us. Moreover, we mark as "inactive" any nodes

whose cover becomes 0, indicating that these nodes are not worthy of consideration for rule derivation since they no longer cover any instances which have not already been classified by some previous rule. Finally, we also mark every node in DC(u) "inactive" since no more specific instance of the rule we just derived will cover any instances that are not already covered by the more general rule.

At this point, if we are generating a set of unordered rules, we can simply repeat the process above by finding a new "active" node which has the largest cover and a non-MIXED labeling. If, however, we are deriving a decision-list, then prior to repeating the process above, we re-label all "active" nodes in the lattice. In this way, some nodes which may have previously been labeled as MIXED may now be labeled with a specific class if enough instances in the downward closure of that node have already been classified by a previously derived rule and are now marked "inactive." Thus in the decision-list format, we may have rules that overlap in the portion instance space that they classify, but since these rules are applied in an ordered manner, there is no ambiguity as to how to classify any given instance. The rule derivation process is repeated until no more "active" nodes remain in the lattice, indicating that we have derived enough rules to properly partition the instance space based on the data inherent in the lattice.

4 Experimental Results

4.1 The MONK'S Problems

The Rulearner system was first tested on the MONK'S Problems, a standard machine learning test-bed of binary classification problems. Each of the MONK'S Problems is defined to have a unique training and complete testing set, so that comparisons between algorithms will not be unduly effected by slight variations in the training data.

During these experiments, several configurations of each system were used to capture the best performance of those systems compared to Rulearner. For CN2 both

MONK'S Problem #1	
Algorithm	Accuracy
CN2 (decision-list)	100.0%
CN2 (unordered rules)	98.6%
C4.5 (unpruned tree)	76.6%
C4.5 (pruned tree)	75.7%
C4.5 -s (unpruned tree)	94.4%
C4.5 -s (pruned tree)	100.0%
Rulearner (decision-list)	100.0%
Rulearner (unordered rules)	100.0%
Rulearner (unordered, pos rules only)	100.0%

Table 2

descision-lists (ordered rule sets) and unordered rule sets were used. Moreover the CN2 rule significance threshold was set to produce the best results on a known noiseless domain. For C4.5, which fared very poorly when attribute grouping (the -s flag) was not used, we include results which make use of this additional feature. We also test three basic configurations of the Rulearner system. In the first configuration, a decision-list was derived. The second configuration derived an unordered set of rules to classify both positive and negative instances. Finally, we derived a rule set to classify only the positive instances, with instances not covered by these rules being considered negative by default. Note that an unordered set of rules that all have the same consequent classification is equivalent to an decision-list of the same form with a default case added. The noise parameter in this experiment was set at 0% for all Rulearner configurations.

The first MONK'S Problem (Table 2) represents a concept that is easily representable in DNF and contains no noise in the training set. As a result, most of the symbolic learning systems fare very well with this problem. Surprising, however, is the fact that several configurations of the other learning methods were unable to perform this classification task with complete accuracy. This would seem to imply that the solutions offered by these methods were not the most parsimonious, as only four rules are required to achieve 100% accuracy on this task. All three configurations of the Rulearner system not only performed with 100% accuracy but also learned the minimal concept description for the problem.

MONK'S Problem #2	
Algorithm	Accuracy
CN2 (decision-list)	72.9%
CN2 (unordered rules)	75.7%
C4.5 (unpruned tree)	65.3%
C4.5 (pruned tree)	65.0%
C4.5 -s (unpruned tree)	69.0%
C4.5 -s (pruned tree)	70.4%
Rulearner (decision-list)	74.5%
Rulearner (unordered rules)	74.8%
Rulearner (unordered, pos rules only)	70.4%

Table 3

The second MONK'S Problem (Table 3) provides a much greater challenge to symbolic learning systems. This is due primarily to the fact that the concept to be learned in this problem is not easily representable in DNF, being an *exactly k*-of-*n* function. Once again, there is no noise in the training set, and thus the noise parameters for the learning algorithms are set accordingly. Here we find that all the symbolic learning methods hover close to 70% in their accuracy, reflective of the fact that the instance space appears very much like a checkerboard and rules that attempt to partition this space into large regions will group large numbers of positive and negative instances together. This is further supported by the observation that all three learning algorithms generate rather lengthy rule sets or large decision trees, showing that each rule only covers a small area of the total instance space. The decision tree paradigm seems to suffer the most drastically in dealing with this sort of partitioning of the instance space. While it is questionable how often such *exactly k-of-n* functions actually appear in real domains, these experiments show that symbolic learning methods must integrate a bias for more than just functions easily representable in DNF if they hope to fare well on a wide range of problems.

Interestingly, we find that the Rulearner configurations which partition the instance space into both positive and negative regions explicitly are noticeably more accurate than the method with only attempts to partition positive regions in the instance space. This is

MONK'S Problem #3	
Algorithm	Accuracy
CN2 (decision-list, chi-square = 0.0)	93.3%
CN2 (unordered rules, chi-square = 0.0)	90.7%
CN2 (decision-list, chi-square = 4.0)	94.4%
CN2 (unordered rules, chi-square = 4.0)	87.5%
C4.5 (unpruned tree, CF = 15%)	92.6%
C4.5 (pruned tree, CF = 15%)	97.2%
C4.5 (unpruned tree, CF = 25%)	92.6%
C4.5 (pruned tree, CF = 25%)	97.2%
Rulearner (decision-list, 5% noise)	94.4%
Rulearner (unordered rules, 5% noise)	94.0%
Rulearner (decision-list, 10% noise)	94.4%
Rulearner (unordered rules, 10% noise)	95.1%

Table 4

reflective of the different biases in these configurations with regard to how much each configuration is allowed to expand the region around any cluster of positive instances. The set of rules which only classify positive instances has a tendency to enlarge the region classified as positive since there are no competing negative classification rules which are also attempting to enlarge the regions classified as negative. Reducing the competition between classifications (as results when only positive classification rules are derived) may in some cases be desirable when the consequences of a false negative are far more costly than a false positive prediction. Thus the Rulearner system can be used with different rule induction biases when faced with an asymmetric cost function for incorrect classifications.

The third MONK'S Problem (Table 4) provides an interesting test case for symbolic learning methods as the concept to be learned is easily representable in DNF, but 5% of the training instances are misclassified due to noise. Note that the testing set does not contain any such noise. To maximize the performance of the learning algorithms, but still maintain brevity, we tested certain basic configurations of each algorithm using different noise toleration parameters to produce the best results for each algorithm. Here we see that all three algorithms are clearly able to learn in the presence of noise. More importantly, however, we see the importance of pruning as reflected in the results from C4.5. Since CN2 and Rulearner currently do no pruning of their induced rules in a post-processing

phase, it appears that this could be a promising venue to further increase the accuracy of both of these algorithms on noisy data, especially since Rulearner outperforms C4.5 before pruning. Another point of interest with regard to this problem is the fact that when the noise parameter in the Rulearner systems was increased from 5% to 10%, there was an improvement in the system overall. This is indicative of the fact that the classification noise inherent in the training data is not evenly distributed over the instance space but may be localized within a few regions. Thus by allowing for more than 5% noise in the system, we can compensate for regions of the instance space where noise may be clustered while not degrading the performance of the algorithm in classifying regions of the instance space which contain relatively little or no noise.

4.2 Breast Cancer Data

All three algorithms were also tested on a standard real world domain: predicting the recurrence of breast cancer from the Yugoslavian Breast Cancer data set. In this domain, a standard data set is partitioned into a training and testing set, 70%/30% respectively. Three such partitions of the data were made and the results shown reflect the mean and standard deviation of the algorithms over the partitionings. Since the original data set contained some duplicate feature vectors with differing classifications, one of the duplicate instance vectors was randomly discarded from the data. Also, some of the data in the breast cancer data set was originally numeric, but was partitioned into equally-sized high grain symbolic features prior to learning as an implementation necessity. Again, several configurations of each system were tested, each with a variety of noise toleration parameters. In Table 5 we see the results of these experiments.

Foremost, it is important to note the difficulty of learning inherent in this data set in that none of the algorithms tested perform significantly better than the default rule, which is simply to predict the majority classification for all instances. In results reported by other researchers on this same data set [Clark & Niblett, 1989], there are actually many other

Breast Cancer

Algorithm	Accuracy
Default Rule (majority)	71.7 ± 7.2%
CN2 (decision-list, chi-square = 0.0)	65.0 ± 1.4%
CN2 (unordered rule set, chi-square = 0.0)	69.7 ± 4.9%
CN2 (decision-list, chi-square = 4.0)	73.3 ± 6.0%
CN2 (unordered rule set, chi-square = 4.0)	71.3 ± 4.5%
CN2 (decision-list, chi-square = 8.0)	72.1 ± 4.0%
CN2 (unordered rule set, chi-square = 8.0)	72.9 ± 5.3%
C4.5 (unpruned tree, CF = 15%)	67.7 ± 9.8%
C4.5 (pruned tree, CF = 15%)	75.2 ± 7.6%
C4.5 (unpruned tree, CF = 25%)	67.7 ± 9.8%
C4.5 (pruned tree, CF = 25%)	73.3 ± 4.5%
C4.5 (unpruned tree, CF = 35%)	67.7 ± 9.8%
C4.5 (pruned tree, CF = 35%)	72.5 ± 4.6%
Rulearner (decision-list, 20% noise)	70.5 ± 6.9%
Rulearner (unordered rules, 20% noise)	72.1 ± 4.0%
Rulearner (decision-list, 30% noise)	74.9 ± 5.8%
Rulearner (unordered rules, 30% noise)	74.1 ± 6.4%
Rulearner (decision-list, 40% noise)	74.8 ± 6.6%
Rulearner (unordered rules, 40% noise)	73.6 ± 7.8%

Table 5

induction algorithms which perform even worse than the default prediction rule on this data set, so the results given here should actually be considered encouraging. Nevertheless, this data set gives us some critical insights into the applicability of such rule induction methods. Foremost, we again see the importance of pruning as the results with C4.5 clearly indicate, showing a jump in predictive accuracy by as much as 8% on a very difficult data set when pruning is employed. As mentioned previously, since CN2 and Rulearner do no currently employ a rule pruning scheme, the addition of such a post-processing mechanism to the algorithms should produce even better results in the future. In spite of this shortcoming, the Rulearner system still performs on par with the pruned trees produced by C4.5 and seems to outperform both CN2 and the unpruned trees from C4.5, although not by much in some cases.

In comparing the decision-list and unordered rule sets produced by CN2 and Rulearner we find that neither paradigm seems to be a clear winner. Of interest, however, is the fact that both learning paradigms — decision-lists and unordered rule sets — produce the best

results when using the same noise toleration parameter (30%) in Rulearner. Thus it appears that both paradigms are equally sensitive to noise in the training data.

5 Complexity Issues

Several interesting complexity issues arise when dealing with the problem of producing lattices. In the worst case, the lattice can contain one node for every combination of features — forming the power set of the feature space — and thus will have a time and space complexity that is exponential in the number of features. This would seem to render the construction of such lattices impossible. Nevertheless, a great deal of empirical evidence [Oosthuizen, 1994; Carpineto & Romano, 1993] has shown lattice construction to require polynomial time and space — with bounds as low as $O(m^2)$, where m is the number of instances, reported. In our experiments, lattice construction fell well within polynomial bounds concurring with previously reported results setting an empirical upper time and space bound at $O(mn^3)$, where m is the number of instances and n is the number of attributes per instance. The actual running time for all of our experiments was quite reasonable with lattice construction and rule induction (along with calculating extensive diagnostic information) taking less than 1 hour per data set on a mid-range workstation.

6 Related Work

The interested reader should also be aware of previous systems such as CHARADE [Ganascia, 1987] and GRAND [Oosthuizen, 1988] which have also made use of lattices to guide the formation of classification rules. These systems, however, differ markedly from ours in their induction mechanisms. In the CHARADE system, a Galois connection is defined between two *implicit* lattices which contain hierarchies of relations (concept descriptions) and instances, respectively. Rules are then induced by by finding the most specific description in the relation lattice lattice which satisfies a basic initial description. Since instance classifications are simply considered an additional feature of each instance,

we can set our initial description to be `(class = 1)`, for example, and then use the Galois connection to find the set of instances, S, in the instance lattice that satisfies this description. The algorithm then finds the *most specific* rule in the relation lattice which encompasses the instances in S — a different bias than that used in Rulearner.

In the same spirit, the GRAND system constructs an explicit lattice from the training data presented to it. As in CHARADE, each instance classification is treated as merely another feature of each instance vector and is thus included in the construction of the lattice. This lattice is then traversed beginning from feature nodes which represent a particular class to find the meet of this class node with various subsets of other features and induce rules based on the features found at such meets. While the lattice construction method we use is based directly on [Oosthuizen, 1988], our induction method differs considerably in that we do not include instance classifications as explicit features in our lattices, but rather use instance classifications as informational metrics which are used along with the instance lattice to induce rules. This makes our system substantially different from that of Oosthuizen who relies heavily on the existence of classification features in the lattice for rule induction as outlined above.

7 Future Work

A number of venues for exploration are still open in the development of Rulearner and related systems. As indicated by several experiments, the next step in helping to make Rulearner a more effective learner is to incorporate a post-processing rule pruning phase in the induction process. Such a mechanism has proven effective in systems such as C4.5 and there is no reason that similar techiniques could not be effective when applied to the rules generated by Rulearner. Furthermore, Rulearner could be made even more robust in the presence of noise by simply requiring a minimum cover value at a node before inducing a rule from that node in the lattice. In this way we could help to avoid producing very specific rules that are likely to be fitting noise in the training data. Combining these line of

research, another route for future work will involve using Rulearner to induce all — even redundant — rules from the instance lattice and then apply pruning to this extended rule set to produce an "optimal" rule set with respect to the training data. There are considerable complexity issues involved in this latter line of research which would also need to be considered. The effect of the noise parameter in Rulearner also needs to be more rigorously tested using methods such as cross-validation as opposed to the simple manual hill-climbing search used for noise parameters in the experiments reported here. Finally, much more detailed empirical experiments comparing unordered rule sets versus decision-lists need to be performed to discover what the true advantages of each paradigm are.

Acknowledgments. The author is grateful to Nils Nilsson who proposed many of the early insights that compose the foundations of the Rulearner system. The author is also indebted to Deon Oosthuizen for many thought provoking discussions regarding the use of lattices in machine learning and also for providing both the GRAND program for the construction of lattices as well as the breast cancer data set. Additional thanks are also extended to Peter Clark for providing CN2 and to George John, Pat Langley, Ronny Kohavi and two anonymous reviewers for their insightful pointers regarding existing rule induction systems. The author is supported by a Fred Gellert Foundation ARCS fellowship.

References

[Birkhoff, 1967]
Birkhoff, G. *Lattice Theory, 3rd Edition*. Providence, Rhode Island: American Mathematical Society, 1967.

[Carpineto & Romano, 1993]
Carpineto, C. and Romano, G. GALOIS: An Order-Theorestic Approach to Conceptual Clustering. In *Proceedings of the International Machine Learning Conference*, pp. 33-40, Amherst, Massachusettes, 1993.

[Clark & Niblett, 1989]
Clark, P. and Niblett, T. The CN2 Induction Algorithm. *Machine Learning, 3*:261-83, 1989.

[Ganasci, 1987]
Ganascia, J. G. CHARADE: A Rule System Learning System. In *Proceedings of the Tenth International Joint Conference on Artificial Intelligence, Volume 1*, pp. 345-347, Milan, Italy, 1987.

[Kononenko *et al*., 1984]
Kononenko, I. *et al*. *Experiments in Automatic Learning of Medical Diagnostic Rules*. Ljubljana, Yugoslavia: E. Kardelj University, Electrical Engineering Technical Report, 1984.

[Kourie *et al.*, 1992]
Kourie, D. G. *et al*. *Lattices in Artificial Intelligence: Complexity Issues*. University of Pretoria Technical Report CSTR 92/08, 1992.

[Michalski, 1969]
Michalski, R. S. On the Quasi-minimal Solution of the General Covering Problem. In *Proceedings of the Fifth International Symposium on Information Processing*, pp. 125-128, Bled, Yugoslavia, 1969.

[Michalski & Stepp, 1983]
Michalski, R. S. and Stepp, R. E. Learning from Observations: Conceptual Clustering. In R. S. Michalski, J. G. Carbonell, T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach*, Tioga, Palo Alto, 1983.

[Mitchell, 1982]
Mitchell, T. M. Generalization as Search. *Artificial Intelligence*, 18(2): 203-226, 1982.

[Nilsson, 1992]
Nilsson, N. J. N-Cube Lattices and Their Role in Machine Learning. *Working paper*, Department of Computer Science, Stanford University, Stanford, CA, 1992.

[Oosthuizen, 1988]
Oosthuizen, G. D. The Use of a Lattice in Knowledge Processing. PhD Thesis, University of Strathclyde, Glasgow, 1988.

[Oosthuizen, 1989]
Oosthuizen, G. D. Machine Learning: A Mathematical Framework for Neural Network, Symbolic, and Genetics-Based Learning. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pp. 385-390, 1989.

[Oosthuizen, 1993]
Oosthuizen, G. D. personal communication, 1993.

[Oosthuizen, 1994]
Oosthuizen, G. D. *The Application of Concept Lattices to Machine Learning*. University of Pretoria Technical Report CSTR 94/01, 1994.

[Oosthuizen & McGregor, 1988]
Oosthuizen, G. D. and McGregor, D. R. Induction Through Knowledge Base Normalization. In *Proceedings of the European Conference on Artificial Intelligence*, Munich, 1988.

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

[Quinlan, 1993]
Quinlan, J. R. 1993. *C4.5: Programs For Machine Learning*. San Mateo, CA: Morgan Kaufmann.

[Rivest, 1987]
Rivest, R. L. Learning Decision Lists. *Machine Learning, 2*:229-246, 1987.

[Thrun *et al*., 1991]
Thrun, S. B. *et al. The MONK'S Problems.* Carnegie-Mellon University Technical Report CMU-CS-91-197, December, 1991.