in *Machine Learning: A Multistrategy Approach, Vol. IV*, R.S. Michalski and G. Tecuci, (Eds.), Morgan Kaufmann Publishers, San Francisco, CA, 1994

19

COMPARING SYMBOLIC AND SUBSYMBOLIC LEARNING:

Three Studies

Janusz Wnek
Ryszard S. Michalski
(George Mason University)

Abstract

This chapter reports on three studies comparing symbolic and subsymbolic methods for concept learning from examples. The first study compared five learning methods, three representing symbolic learning paradigm—decision tree learning (C4.5), rule learning (AQ15), and constructive rule learning (AQ17-HCI)—and the other two representing the subsymbolic paradigm—neural net learning using backpropagation (BpNet) and a classifier system employing genetic algorithm (CFS). All methods have been applied experimentally to learn several different *DNF-type* concepts (i.e., concepts representable by a simple DNF expression). The second study compared performance of a large number of learning programs on learning DNF-type concepts from data with and without noise and a non-DNF-type "m-of-n" concept. The third study compared genetic algorithm based learning (GABIL and Adaptive GABIL) with decision tree learning (C4.5) and decision rule learning (AQ14), on twelve DNF-type concepts. All studies have shown that generally, symbolic methods, in particular those applying constructive induction, outperformed subsymbolic methods in learning DNF-type concepts from data both without and with noise. In case of learning non-DNF-type concepts, symbolic methods without constructive induction performed worse, but those with constructive induction matched the performance of neural network methods.

19.1 INTRODUCTION

In view of a rapidly growing interest in multistrategy learning systems, it is important to develop insights into the performance of diverse learning methods and paradigms and to determine the areas of their most desirable applicability. To this end, this chapter presents various studies of the performance of symbolic and subsymbolic methods as applied to the same learning problems. The first study (Sections 19.2–19.4) involved a comparison of three symbolic and two subsymbolic methods. Symbolic methods were represented by C4.5, a decision tree learning program; AQ15, a decision rule learning program; and AQ17-HCI, a constructive decision rule learning program. Subsymbolic methods were represented by CFS, a genetic algorithm based classifier system, and BpNet, a neural network learning program using backpropagation algorithm. The other two studies (Sections 19.5–19.6) involved the same programs or their different variants.

An important difference between symbolic and subsymbolic learning approaches lies in the cognitive aspects of the employed knowledge representation. Knowledge represented by logic-based rules or decision trees (especially when the latter are small) is relatively easy to comprehend and relate to human knowledge. This is not the case with knowledge represented by classifier systems or neural networks. Although for some applications, it may not be important that the learned concept descriptions are understandable to people, e.g., in an adaptive controller of house temperature, in some other applications, e.g., in expert systems for human disease diagnosis, business or military decision making, this requirement is crucial.

Despite various attempts, there is no established universal measure of cognitive comprehensibility of concept representations (Michalski, 1983). Therefore, we will make a simplifying assumption that the comprehensibility of a concept can be estimated roughly by the number of rules needed to express it or the number of disjuncts in an equivalent DNF expression. In this measure, called the *R-complexity (rule-complexity)* of a concept, elementary conditions in the rules representing a concept (or the components of disjuncts in DNF) are assumed to be simple conditions involving *given* attributes. Based on this definition, one can distinguish between two general classes of concepts:

1. Concepts that can be expressed by a simple DNF expression using given attributes (or described by only few rules): we call them *DNF-type* concepts.
2. Concepts that require a very complex DNF expression: we call them *non-DNF-type*.

It is important to point out that concepts that have a long DNF expression using given attributes may have a short DNF expression if these attributes are replaced by other attributes or are transformed into certain combined attributes through the process of constructive induction (e.g., Wnek and Michalski [1991]). Thus, whether a given concept is DNF-type or not depends on the attributes (generally, descriptors)[1] that are available for constructing a concept representation. In other words, the R-complexity is defined with regard to the assumed concept representation space.

All three studies compared several methods by applying them to learning the same class of DNF-type concepts. We found that concepts generated by human subjects who are asked to create classes of entities and to express them linguistically usually fall into such a category. Given a concept representation, its R-complexity can thus be viewed as an approximate indication of the "cognitive" complexity of the concept. For representations other than rule-based, the R-complexity can be determined by converting them to logically equivalent sets of rules. When the description spaces are not too large, this can be done using the DIAV concept visualization method, outlined in Section 19.3.

Presented studies follow several other efforts to compare different learning methods and paradigms. For example, Fisher and McKusick (1989) compared ID3 and a neural net using a backpropagation (BP) algorithm on the problems of learning diagnostic rules for thyroid diseases, soybean plant diseases, and a few artificial problems. The comparison was based on the performance accuracy of descriptions as applied to testing examples and the training time. Their conclusion was that the neural net gave a better performance but required a significantly longer training time and more training examples than ID3.

Mooney et al. (1989) compared ID3 with perceptron and a backpropagation algorithm using the domain of soybean diseases, chess-end games, audiological disorders, and the Nettalk data set. Their conclusion was that the accuracy of classifying new examples was about the same for all three systems, but the neural net performed better than ID3 when there was noise in the data. Weiss and Kapouleas (1989) compared ID3, predictive value maximization, neural net using BP, and a few statistical methods. They found that the statistical classifiers performed consistently better in terms of accuracy in classifying testing examples.

Dietterich, Hild, and Bakiri (1990) compared ID3 with a neural net using BP on the task of text-to-speech mapping. Their major conclusion was that the neural net consistently outperformed ID3 in terms of the performance accuracy and attributed this result to the capture of better statistical information by the neural net.

Bergadano et al. (1992) compared POSEIDON (an extended version of AQ15 using a *two-tiered* concept representation) with exemplar-based and decision tree learning programs. Their study involved two real-world domains: labor contracts and U.S. congressional voting. In this study, descriptions learned by POSEIDON outperformed those produced by all other methods, both in terms of performance accuracy on new examples and in terms of the description's simplicity.

---

[1] By descriptor is meant an attribute or function whose value characterizes the entity.

The first study presented here differs from the above studies in that it experimentally analyzes five different methods and compares the learned descriptions in terms of their *exact error rate*, rather than a statistical error estimate, and also in terms of their R-complexity. In the study, the target and learned concepts were represented graphically by a novel technique of *diagrammatic visualization* (Wnek and Michalski, 1994). This technique permits one to display an *image* of the target and learned concepts and an *error image* that identifies all errors.

The chapter consists of seven sections. Section 19.2 briefly describes the five learning systems used in the first study. Section 19.3 presents the methodology used to compare the methods and describes training data and the concepts to be learned (five DNF-type concepts created by human subjects). The concepts are illustrated by the diagrammatic visualization technique (DIAV). Section 19.4 describes results of experiments with the methods. Sections 19.5 and 19.6 summarize two related studies done by other research groups. The first one involved three types of problems: learning a DNF-type concept, learning a non-DNF-type concept, and learning a DNF-type concept from noisy data. The results were obtained using a large number of learning programs, which were grouped into four categories according to the representational paradigms used: decision tree, decision rules, neural networks, and inductive logic programming (Thrun et al., 1991). The second study applied a decision tree learning program, a decision rule learning program, and two genetic algorithm-based programs to learn twelve DNF-type concepts (the original study was done by Spears and Gordon [1991]). Section 19.7 summarizes results from the comparison of the systems in learning DNF-type concepts.

19.2 LEARNING SYSTEMS INVOLVED IN THE FIRST STUDY

As mentioned earlier, the symbolic paradigm is represented by a decision tree learning program, C4.5, and two rule learning programs, AQ15 and AQ17-HCI. The subsymbolic paradigm is represented by a backpropagation neural network, BpNet, and a classifier system based on genetic algorithm, CFS. These programs are widely known and well described in machine learning literature. To serve the tutorial purpose of the book, we provide here a brief account of the basic algorithm underlying each program and give references to the literature for readers interested in further details.

19.2.1 Decision Tree Learning Program C4.5

C4.5 learns concepts by building a decision tree that correctly classifies supplied examples of the concepts. Each interior node of the tree is assigned an attribute, and the leaf nodes are assigned concept names. A branch down from an interior node represents a value of the attribute assigned to the node. Any path from the root to a leaf in the tree can be viewed as a decision rule for the class assigned to the leaf.

The input to the algorithm consists of sets of training examples for different concepts (or decision classes). In the first step, the algorithm selects a random subset of training examples from each set (a "window"). Then, for each attribute, the information gain, i.e., the information gained if the attribute were chosen for testing, is computed. The attribute with the highest score is assigned to the root node of the tree. Branches from this node represent different values of this attribute. End-nodes of these branches (current leaves) are assigned subsets of examples in which the attribute takes the value associated with the given branch. If a subset contains examples of only one decision class, then the end-node becomes a leaf of the decision tree. For all other subsets, the algorithm is repeated until all leaves in the tree are assigned single decision classes.

At this point, the created tree correctly classifies all examples in the window. Now the tree is used to classify remaining examples from the training set (outside the "window"). If the tree gives the correct answer for all examples, then the process terminates. If not, misclassified examples are added to the window, and the process continues until the trial decision tree correctly classifies all examples not in the window.

The entire process is repeated by default 10 times, and the best decision tree is selected. Because the examples used in the experiments had no noise, decision trees were not pruned. The C4.5 program (Quinlan, 1993) is a derivative of the ID3 program (Quinlan, 1986). In addition to decision tree generating, C4.5 is able to convert an unpruned decision tree into sets of generalized (pruned) decision rules. A tree is converted to rules by forming a rule corresponding to each path from the root of the tree to each of the leaves. All rules are then examined and some of them are generalized (pruned) by shopping conditions. Next, rules for each class are considered separately and redundant rules are removed. For uncovered examples, a default class is assigned.

We have tested all three representations learned by C4.5 using default parameter setting, i.e., the best tree was selected out of ten generated from the same training set, attributes were selected according to **gainratio** (ratio of information gain and potential information) criterion. As expected, because the training examples did not have noise, on average, unpruned decision trees performed best in terms of predictive accuracy. Pruned decision trees and decision rules were simpler but more erroneous. Thus, here we report the results obtained for unpruned decision trees only.

19.2.2 Rule Learning Program AQ15

The AQ15 program generates concept descriptions from concept examples. The descriptions are in the form of decision rules expressed in an attributional logic calculus, called *variable-valued logic*, VL1 (Michalski, 1973). A distinct feature of this representation is that it employs, in addition to standard logic operators, the *internal disjunction* operator (a disjunction of values of the same attribute), which can significantly simplify rules involving multivalued attributes. The program can optimize the rules according to a user-defined (or default) preference criteria, such as the overall simplicity or the evaluation and/or storage cost of the rules. The main procedure of AQ15 is based on the AQ algorithm that builds a concept description from a set of positive and negative examples (e.g., Michalski [1973]). Below is a simplified version of the AQ algorithm:

1. Randomly select a *seed* example from the set of positive training examples of the concept to be learned.
2. Generate a set of most general rules (a *star*) that cover the seed, and do not cover any negative examples (this operation employs the *extension against* generalization operator (Michalski, 1983).
3. Select the "best rule" from the star (according to the assumed preference criteria), and remove examples covered by this rule from the set of training examples.
4. If the set of training examples does not become empty, return to Step 1. Otherwise, the obtained set of rules constitutes a complete and consistent concept description.

The algorithm is repeated for each concept to be learned. It is biased toward finding a conjunctive description of a concept (a single rule) because if such a description exists for the given set of examples, it will be found in the very first step (the description will be a member of the first star generated). The AQ15 program has various parameters whose default values can be changed by a user according to the requirements of the domain. In all the experiments reported here, the preference criteria were to minimize both the number of rules and the number of conditions in them for each concept learned. Because training examples did not have noise, there was no need for any rule truncation procedure. For further details, see Michalski et al. (1986) and Bergadano et al. (1992).

19.2.3 Constructive Rule Learning Program AQ17-HCI

AQ17-HCI represents a recent major advance in the development of the AQ-based series of inductive learning programs, specifically, the above-described AQ15 system. The main new feature of it is an incorporation of a method for *hypothesis-driven constructive induction* (HCI). Constructive induction, as introduced by Michalski (1978), addresses the problem of changing the representation space so that it is more suitable for the learning problem at hand. This involves creating new attributes (or descriptors) that better characterize the concepts to be learned than the original descriptors. The last few years have witnessed an increasing interest in constructive induction methods because they can produce concept descriptions that are more accurate and/or simpler than the traditional *selective* induction methods (Pagallo and Haussler, 1990; Rendell and Seshu, 1990; Wnek and Michalski, 1991).

The HCI method generates problem-relevant descriptors by analyzing consecutively created inductive hypotheses (Wnek and Michalski, 1991). Below is a brief description of the algorithm used in AQ17-HCI. For the sake of simplicity, it is assumed that the training set consists of subsets of positive examples E^+ and negative examples E^-. If the training set consists of subsets representing different concepts, then E^+ represents the subset of training examples for the concept under consideration, and the union of the remaining subsets plays the role of E^-.

1. Divide randomly each of the training sets, E^+ and E^-, into two subsets: the primary set E_p and the secondary set E_s. $E^+ = E_p^+ \cup E_s^+$. $E^- = E_p^- \cup E_s^-$. (The primary training subset is to be used for rule generation, and the secondary subset is to be used for rule verification).
2. For each concept, induce the most specific (ms) cover of the set E_p^+ against the set E_p^-. (Such a cover is denoted COV_{ms} (E_p^+ / E_p^-) and represents the set of the most specific rules that characterize examples in E_p^+ but no examples in E_p^-).
3. Evaluate the performance of the rules on the secondary training set, Es. If the performance exceeds a predefined threshold, or all changes in the representation space were exhausted, go to Step 8.
4. Analyze the rules in order to identify possible changes in the representation space.
5. Change the representation space by removing irrelevant attribute values or attributes or by adding new attribute values or attributes.
6. Modify the training set of examples, E, according to the changes in the representation space.
7. Go to Step 2.
8. For each concept, induce a set of the most specific rules from all positive examples against all negative examples, i.e., a cover COV_{ms} (E^+ / E^-), and the most general cover of negative examples against positive examples, COV_{mg} (E^- / E^+).
9. Build final concept descriptions by generalizing the most specific positive rules against the most general negative rules, i.e., COV_{mg} (COV_{ms} (E^+ / E^-) / COV_{mg} (E^- / E^+)).

The AQ17-HCI program has two important features that place it within the class of multistrategy learning methods. The first one is an ability to change the representation space using HCI. This means that the method uses additional knowledge transmutations allowing abstraction and concretion, apart from inductive generalization and specialization (Michalski, 1994, Chapter 1 of this book). The second feature is an extended generalization heuristic, employed in steps 8 and 9, that additionally generalizes the most specific generalization of the training set. This extension was proposed by Wnek (1992).

19.2.4 Neural Net Program BpNet

A neural network is defined by a set of processing units. Units can be of three types: input, output, and hidden. The hidden units provide communication links between input and output units in the task of translating the input training/testing example into output classification.

Backpropagation, as originally introduced (Rumelhart, Hinton and Williams, 1986), is a learning algorithm for feed-forward networks (networks in which the interconnections form no feedback loops) based on gradient minimization. We consider a network of units in which a weighted sum of the inputs is performed, the result of this sum (also called the activation level of the unit) being fed through a non-linear element, with a differentiable input-output function, e.g., a sigmoid function.

Learning by backpropagation involves two phases. During the first phase, an example is presented and propagated forward through the network to compute the output values o_n for each unit. These outputs are then compared with the target values t_n, resulting in output errors e_n for each unit. The second phase involves a backward pass through the network (analogous to the initial forward pass) during which the error message is passed to each unit in the network, and the appropriate weight changes are made.

The two phases are repeated until the overall error reaches a predefined level. The output error for a given training example is given by

$$e_n = o_n - t_n$$

Where o_n and t_n are the output and the target values of the output unit n. The total squared error for that input example is

$$E = \sum_{n \in U} e_n^2$$

Where U denotes the set of input units. Thus, learning by backpropagation corresponds to gradient minimization of the average squared error. The average is computed over all examples in a given training set. The BpNet program is an implementation of the backpropagation algorithm (McClelland and Rumelhart, 1988).

19.2.5 Classifier System CFS

A classifier system is a parallel rule-based (production) system that was first introduced by Holland and Reitman (1978). The rules, called classifiers, have the same simple form, so it is easy to determine whether a condition part of a rule is satisfied. Because the rules can be active simultaneously, complex situations are expressed by combinations of rules. The classifiers can be modified by a general-purpose learning system (Holland, 1986; Riolo, 1988).

Classifier system learning and classification are done in cycles. In each cycle, an input example is translated into a message that has the same form as the condition part of the rules. Next, the message is compared with all rules. All matched rules compete with each other in order to become active and to yield new messages. The new messages can either store some intermediate information and then be used in the next cycle or produce a final classification if matched with the system's effectors. During the learning cycles, the final classification is compared with the target class of the example, and payoff is distributed among active rules. Payoff changes the rules' strength and bidding chances in the next cycle. In order to supplement the learning process, some classifier systems utilize genetic algorithms. The genetic operators (e.g., crossover, mutation) provide means for rule evolution.

The shell for the classifier system used in the experiments was developed by Riolo (1988). The CFS package of subroutines and data structures is domain independent and provides routines to perform the major cycle of the classifier system. The CFS system was run in the stimulus-response mode, i.e., without generating internal messages. Training cycles were repeated fifty times for each example. Payoff for correct and incorrect answers was set to 6 and −1, respectively, with a full payoff paid to all active classifiers. Final classification was produced by two effectors. The CFS package uses more then 150 control parameters. The population size of sixty classifiers, the number of training cycles, the payoff, and about 20 other parameters were determined experimentally. The remaining parameters were set to default values.

19.3 METHODOLOGY

The testing domain in this study is the world of robot-like figures in the EMERALD[2] system. For simplicity's sake, the robots are described by just six multivalued attributes (Figure 19.1A). The attributes are Head Shape, Body Shape,

[2] EMERALD is a large-scale system integrating several different learning programs for the purpose of education and research in machine learning (Kaufman, Michalski, and Schultz, 1989). It was developed at the Center for Artificial Intelligence at George Mason University. An earlier version, ILLIAN, was developed at the University of Illinois at Urbana-Champaign.

Smiling, Holding, Jacket Color, and Tie and can have 3, 3, 2, 3, 4, and 2 values, respectively. Consequently, the size of event space (the space of all possible robot descriptions) is $3 \times 3 \times 2 \times 3 \times 4 \times 2 = 432$. The space of all possible concepts in this representation space is $2^{432}-1$ ($\approx 10^{143}$). Undergraduate computer science students unfamiliar with machine learning were asked to create five concepts from a predefined set of robots in the EMERALD system (16 examples). Each concept represented a certain class of imaginary robots. Below are descriptions of the concepts ("Target concepts") used in the experiments with the total numbers of positive and negative examples:

C1:	*Head is round and jacket is red*	or	*head is square and is holding a balloon*	(84 positive, 348 negative)
C2:	*Smiling and is holding balloon*	or	*smiling and head is round*	(120 positive, 312 negative)
C3:	*Smiling and not holding sword*			(144 positive, 288 negative)
C4:	*Jacket is red and is wearing no tie*	or	*head is round and is smiling*	(117 positive, 315 negative)
C5:	*Smiling and holding balloon*	or	*holding sword*	(144 positive, 288 negative)

Each such concept represents a partitioning of the event space into robots that belong to the concept (positive examples) and those that do not (negative examples). Based on the concepts C1–C5 the students generated initial sets of training examples used in Experiment 1. Each initial training set consisted of approximately 6% of all positive examples (Pos1) and 3% of all negative examples (Neg1). The remaining sets for Experiments 2–5 were generated by adding to the initial set an appropriate number of randomly generated examples: Pos2 and Neg2 (10% positive and 10% negative), Pos3 and Neg3 (15% positive and 10% negative), Pos4 and Neg4 (25% positive and 10% negative), and Pos5 and Neg5 (100% positive and 10% negative). These additional experiments were performed in order to observe the convergence of the learned concepts to the target concepts.

The concepts C1–C5 are presented graphically in Figures 19.1B and 19.2 using a method for *diagrammatic visualization*. This method employs a *General Logic Diagram* (GLD) that is a planar representation of a multi-dimensional space spanned over multi-valued discrete attributes[3] (Michalski, 1973; Wnek and Michalski, 1993). Each cell in the diagram represents a combination of the attribute val-

[3]The system DIAV implementing the visualization method (Wnek and Michalski, 1993) permits one to directly display description spaces with as many as 10^6 cells (e.g., about twenty binary attributes). Larger spaces can also be displayed, but their representations have to be projected to subspaces.

A. ROBOTS Domain

Attributes	Possible Values
HS Head Shape	round, square, octagon
BS Body Shape	round, square, octagon
SM Smiling	yes, no
HO Holding	sword, balloon, flag
JC Jacket Color	red, yellow, green, blue
TI Tie	yes, no

+ Positive example

— Negative example

Cell A - a robot description:
[HS=r] & [BS=r] & [SM=y] & [HO=f] & [JC=g] & [TI=n]

Target concept C1 consists of two rules:
Rule R1: Head is round and jacket is red
Rule R2: head is square and is holding balloon

– cells representing rule R1: [HS=r] & [JC=r]
– cells representing rule R2: [HS=s] & [HO=b]

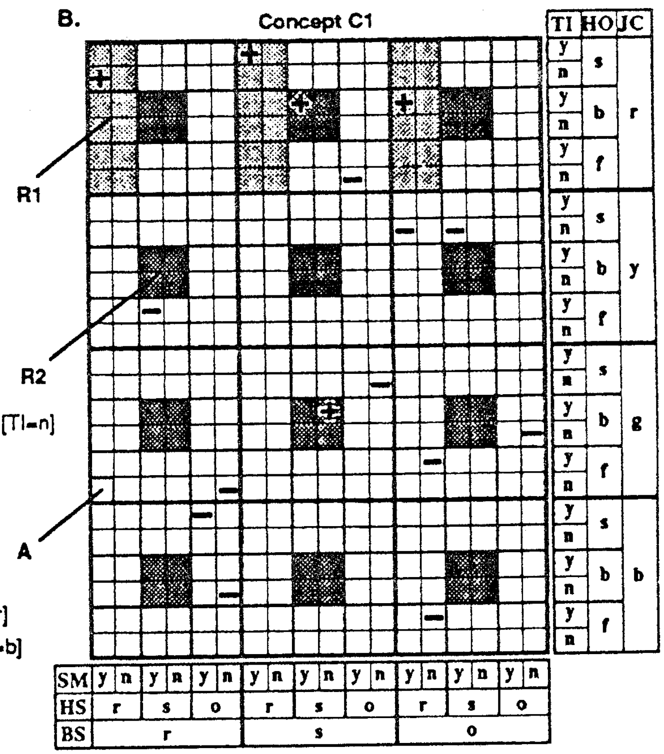

Figure 19.1: (a) Description of the ROBOTS domain; (b) A visualization of the target concept C1 and the initial training examples.

ues, e.g., a concept example. For example, the cell A in Figure 19.1B represents the following robot description:

Head Shape = round, Body Shape = round, SMiling = yes, HOlding = flag, Jacket Color = green, TIe = no

Positive and negative training examples are marked with + and −, respectively. Concepts are represented as sets of cells. The concept C1 can be viewed as consisting of two rules. They are represented in the diagram by shaded areas marked R1 and R2.

R1: Head Shape is round *and* Jacket Color is red.
R2: Head Shape is square *and* is HOlding balloon.

An important advantage of the diagrammatic visualization is that it permits one to display steps in learning processes as well as the errors in concept learning. The set of cells representing the target concept (the concept to be learned) is called *target concept image (T)*. The set of cells representing the learned concept is called *learned concept image (L)*. The areas of the *target concept* not covered by the *learned concept* represent *errors of omission* $(T \setminus L)$, and the areas of the learned concept not covered by the target concept represent *errors of commission* $(L \setminus T)$. The union of both types of errors represents the *error image*. In the diagrams, errors are marked by slanted lines.

Target and learned concepts are represented in the diagrams by shaded areas. However, if the target and learned concepts are both visualized in the same diagram, then the shaded areas represent learned concept. The location of the target concept is implicitly indicated by correctly learned concept and errors of omission. Because errors of commission are part of a learned concept, corresponding areas on the diagram are both shaded and slanted. Errors of omission are not part of the learned concept: thus, the corresponding slanted areas remain white in the background. The parts of the target concept that were correctly learned are shaded only.

The descriptions learned by the methods were compared in terms of the *exact error rate*, a representation-independent complexity. *Exact error rate* is the ratio between exact error and the size of event space. It is measured as a function of the number of training examples. *Exact error* is defined as the total number of errors of omission and errors of commission or, equivalently, the cardinality of the set-difference between the union and the intersection of the target and learned concepts.

$$Exact\_error\_rate = \frac{Exact\_error}{\#Event\_space}$$

$$Exact\_error = \#\,[(T \setminus L) \cup (L \setminus T)] = \#\,[(T \cup L) \setminus (T \cap L)]$$

There are many ways to define error rates in order to reflect certain inductive capabilities of a learning system. In the definition above, for simplicity, we do not

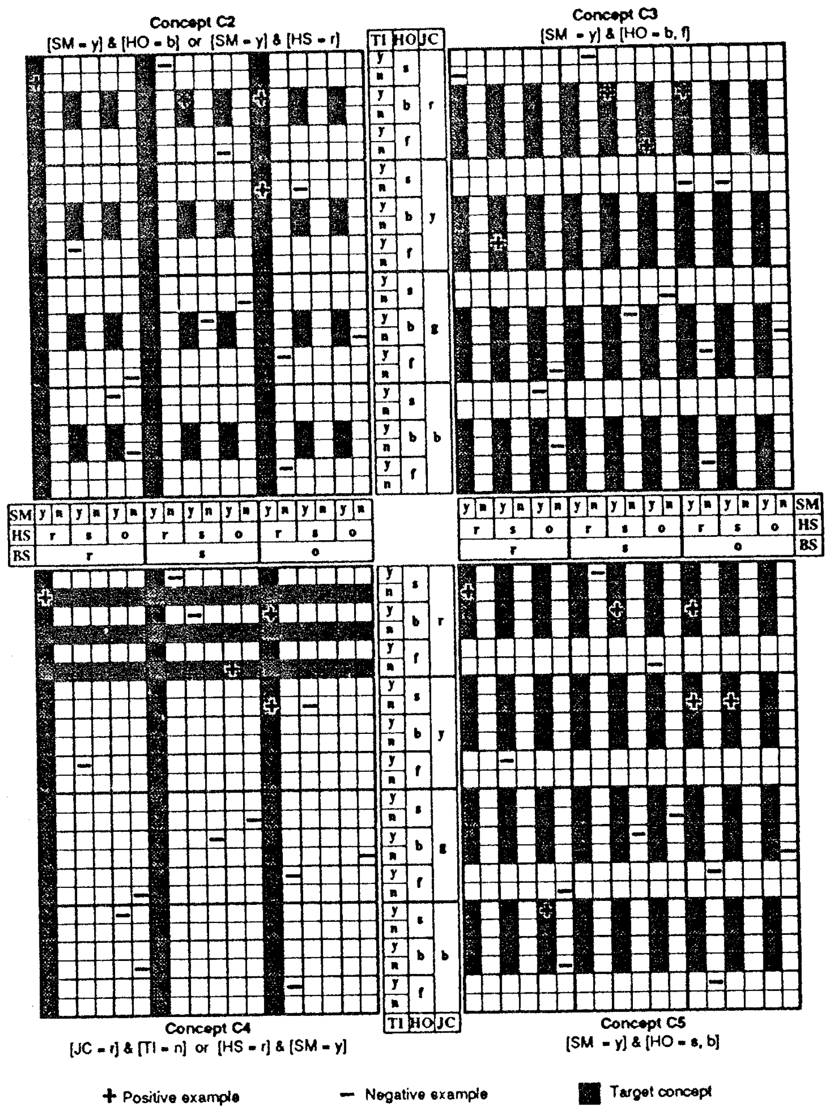

Figure 19.2: A visualization of the target concepts C2, C3, C4, and C5 and the initial training examples for each concept

make any distinction between errors of omission and errors of commission, which may be important in some real-world domains. Also, the domain is small and well-structured, thus suited to the representation of the specific objects of the domain. Therefore, we can avoid the well-known Hempel's paradox in which confirmation of a concept can be made by the lack of satisfaction of non-examples (Hempel, 1965; Kodratoff, 1994, Chapter 3 of this book).

In order to get complete insight into the performance of the tested methods, we used all examples from the event space to test the performance. Note, however, that training examples are often excluded from the testing phase. The same kind of testing was also used in the remaining two studies.

In addition to the exact error rate, we used a representation-dependent *R-complexity* (rule complexity) measure of a method performance. The *R-complexity* of a concept representation is defined by the number of conjunctive statements (rules) in the minimal DNF expression that is logically equivalent to the given representation. Because finding such a minimal DNF expression for any given representation may be difficult (it is generally an NP-hard task), we use an estimate of the R-complexity. For a method that learns a rule-based representation, the number of rules generated by the method is taken as such an estimate. For example, the R-complexity of the C1–C5 target concepts is 2, 2, 1, 2, and 1, respectively. For a decision tree learning method, the R-complexity is estimated by the number of leaves in the tree (because each leaf corresponds to a rule). For neural nets and classifiers, the R-complexity is estimated by determining the number of conjunctive statements needed to re-express the learned concept as a DNF expression.

19.4 EXPERIMENTS IN THE ROBOTS DOMAIN

19.4.1 Representations Learned

Figure 19.3 presents an example of representations learned by each method. In the figure, the representations were learned in Experiment 1 from 6% positive and 3% negative examples of the target concept C1:

Head is round and jacket is red, or head is square and is holding a balloon.

A Decision Tree Generated by C4.5. Figure 19.3A shows the best, unpruned decision tree selected out of ten different trees generated from the training set. The learned concept is described using two attributes: Jacket Color and Head Shape. The learned concept can be read as follows:

IF Jacket Color is red, and Head Shape is round or
Jacket Color is red, and Head Shape is square or
Jacket Color is green, and Head Shape is square
THEN C1

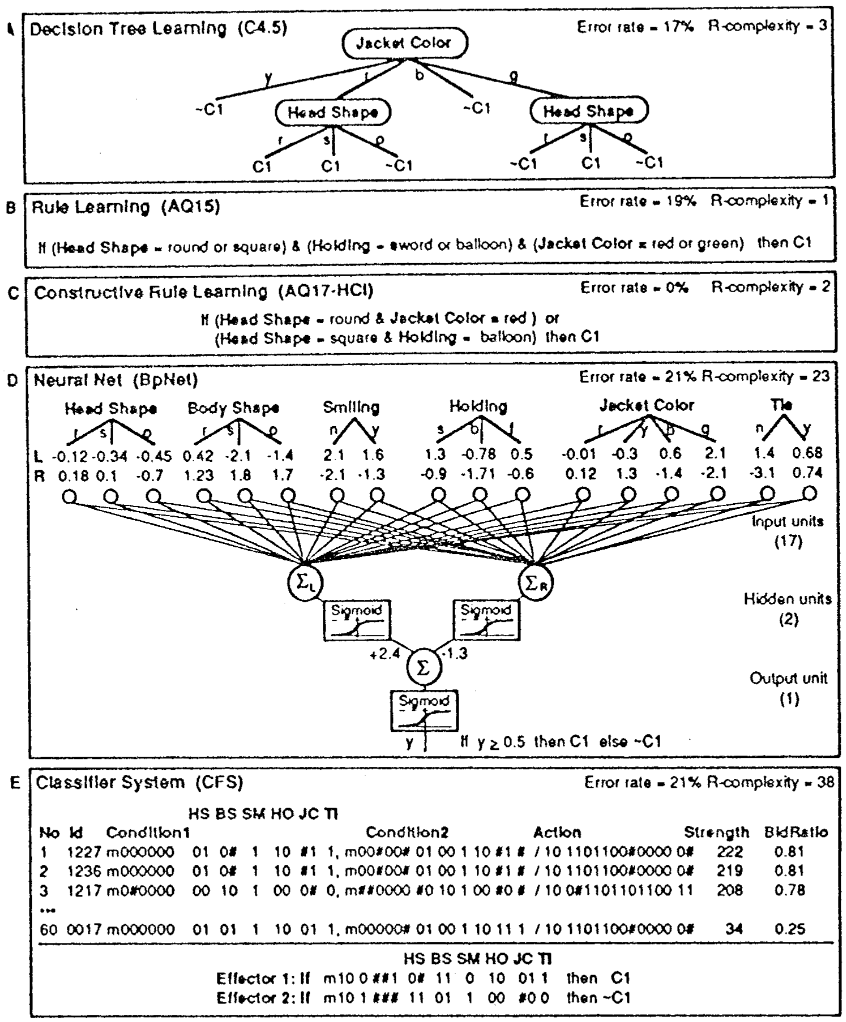

Figure 19.3: Representations of the concept C1 learned by different methods (from the initial set of examples consisting of 6% of positive and 3% of negative examples)

The exact error rate is 16.7% and the R-complexity of this tree is 3. After pruning, the tree is reduced to a root labeled ~C1. Such a tree classifies all examples as not belonging to concept C1 and, thus, produces 84 omission errors (19.4% error rate). The R-complexity of the tree is 1. The third representation learned by

C4.5 are decision rules obtained from the unpruned decision tree. After rule pruning and simplification, the final outcome consists of two rules: (1) If Head Shape is octagonal, THEN ~C1. (2) DEFAULT CLASS is ~C1. These rules are equivalent to the pruned decision tree and produce the same errors. The R-complexity is 2.

A Decision Rule Generated by AQ15. The method generated one rule. It consists of three conditions. Each condition tests one attribute. The internal disjunctions simplify the rule (Figure 19.3B).

A Decision Rule Generated by AQ17-HCI. The rule learned by AQ17-HCI is exactly the target concept (Figure 19.3C). It was generated in a transformed, smaller description space. Figure 19.4 shows steps in learning concept C1 by AQ17-HCI. The input to the method is a set of training examples in the original representation space, as shown in diagram A (the diagram also shows the target concept). The method divides the training set into primary and secondary examples and employs the AQ15 learning algorithm to induce rules from the primary set of training examples (diagram B). Because the performance test on the secondary training set is not satisfactory, the representation space is reduced to contain relevant attributes only, i.e., those attributes that are present or significant in the induced hypothesis. Therefore, the method changes ROBOTS original representation space by removing three irrelevant attributes: Body Shape, SMiling, and TIe (diagram C). In the new representation space, the number of training examples is decreased by 1. It is because two positive examples, E1 and E2, from the original event space have the same description in the new event space.

E1: (**round**, round, yes, **sword**, **red**, no)
E2: (**round**, square, yes, **sword**, **red**, yes)

Although such an abstracted problem is simpler for learning, the resulting hypothesis is still not accurate (diagram D). At this point, the training data set seems to be insufficient to allow proper learning. The lacking information can, however, be induced if both positive and negative hypotheses are considered at the same time. Figure 19.4, diagrams D and E, shows two covers, COV_{ms} (E^+/E^-) and COV_{mg} (E^-/E^+), that were generated using all initial training examples. AQ17-HCI generalized the positive concept description against the negative concept and, by this means, improved the learned concept. The concept C1 was learned precisely as shown in diagram F.

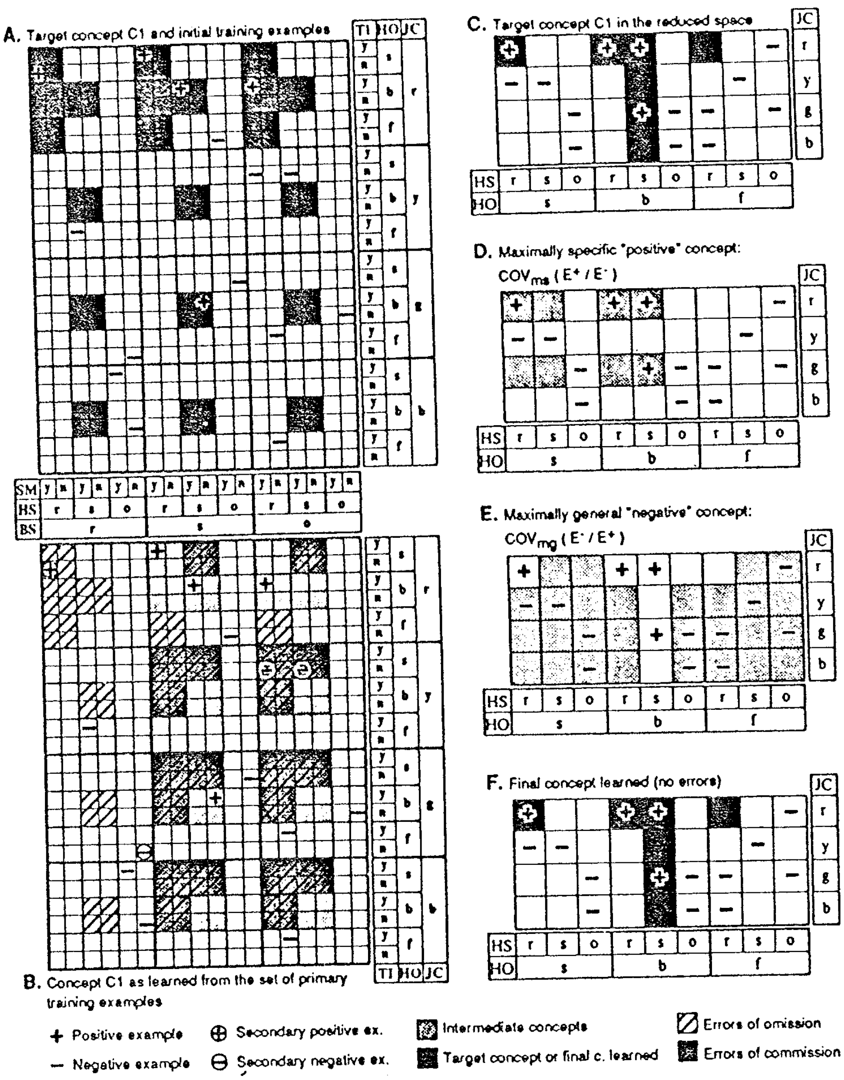

Figure 19.4: Steps in learning concept C1 by AQ17-HCI

A Neural Net Generated by BpNet. Figure 19.3D shows an architecture of the neural net used in the experiments. There were seventeen input units, all having either value 0 or 1, corresponding to attribute-values. All input units had connections to two hidden units. The number of hidden units was determined experimentally. The two hidden units were connected to one output unit. The network was trained by the BpNet backpropagation algorithm until it reached root mean square error below 0.0007. The final connection weights for the concept C1 are shown in the figure. Because of space limitations, the connections from the input units to the left hidden unit and the right hidden unit are specified in the rows marked L and R, respectively. The weights from the hidden units to the output unit are 2.4 and −1.3. An input example is classified as a C1 class member if it is translated into output value ≥ 0.5.

Classifiers Generated by CFS. Each line in Figure 19.3E represents one classifier in the following format: No, Id, Classifier, Strength, and BidRatio (Riolo, 1988). The total population for representing the concept consists of sixty classifiers. Each of the classifiers (condition-action rules) is in the following form:

condition1, condition2 / action

Each condition consists of a string of a fixed length (16) built from the tertiary alphabet {0, 1, #}. A condition string with prefix "m" is matched by any message that has 0s and 1s in exactly the same positions as the 0s and 1s in the condition string. The # in the condition is considered a "wildcard" symbol that can match a 0 or a 1. A classifier's condition-part is satisfied when both of its conditions are matched. When the condition-part of a classifier is satisfied, the classifier becomes active; i.e., its action-part produces one output message. The messages generated by active classifiers are compared to effectors in order to produce final classification. In Figure 19.3E, *BidRatio* is a number between one and zero that is a measure of the classifier's specificity, i.e., how many different messages it can match. *Strength* is meant to be a measure of a classifier's "usefulness" to the system. The higher a classifier's strength, the more it bids.

19.4.2 Summary of Results

Figures 19.4 and 19.5 present the results of learning concept C1 by the five learning systems using diagrammatic visualization. In comparison to the representations in Figure 19.3, these diagrams give a uniform image of the learning results. From the diagrams, one can easily determine learning accuracy (correct vs. error areas—black vs. shaded areas) and interpret the errors (why certain areas were covered or not). Most importantly, one can generate rules equivalent to the learned

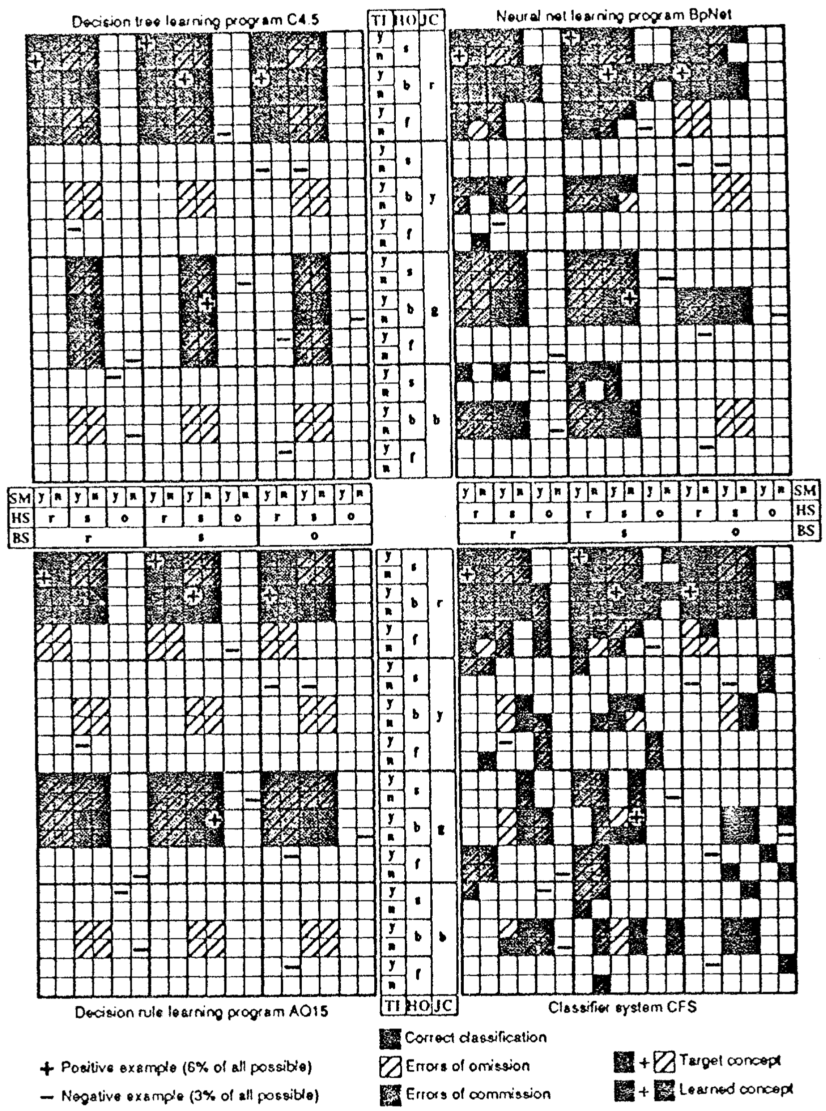

Figure 19.5: Concepts learned by different methods in the relation to the target concept C1

Table 19.1: The average error rate of learned descriptions

	The Relative Size of the Set of Training Examples in Each Experiment				
	Experiment 1 (6%, 3%)*	Experiment 2 (10%, 10%)*	Experiment 3 (15%, 10%)*	Experiment 4 (25%, 10%)*	Experiment 5 (100%, 10%)*
Genetic Alg. (CFS)	21.3%	20.3%	22.5%	19.7%	16.3%
Neural Nets (BpNet)	9.7%	6.3%	4.7%	7.8%	4.8%
Decision Trees (C4.5)	9.7%	8.3%	1.3%	2.5%	1.6%
Decision Rules (AQ15)	22.8%	5.0%	4.8%	1.2%	0.0%
Decision Rules (AQ17-HCI)	4.8%	1.2%	0.0%	0.0%	0.0%

*In each (x%, y%), x denotes positive training examples and y negative training examples.

representation and determine an R-complexity of the description. This feature is especially useful for subsymbolic systems that do not have easily understood knowledge representation, as shown in Figure 19.3.

The final concept description learned by AQ17-HCI exactly matches the target concept, and thus, there are no slanted areas in diagram F in Figure 19.4. The other four methods did not learn the concept C1 precisely; however, all the methods were consistent with the training examples (Figure 19.5). The error rate level is almost even for all of them (about 20%), but one can note differences in their generalization patterns. The symbolic methods yield regular, rectangular covers as opposed to irregular covers of subsymbolic methods.

Tables 19.1 and 19.2 summarize the results of all the experiments. For each learning program, the final result in Experiment 1 is an average over results from learning the five concepts from their initial training sets (column 1). In the remaining experiments, because additional examples were generated randomly, the testing was repeated 10 times for each concept. Consequently, for each learning program, the result is an average from 50 learning sessions (cols. 2–5). Pairs (a,b) in the top row of the tables denote the percentage of positive and negative examples used in experiments.

Table 19.1 shows the average exact error rate of the descriptions learned in five experiments, and Figure 19.6 presents corresponding learning curves. The error rate of the CFS-generated descriptions was much higher than that of the other descriptions, and what is most surprising, it did not improve much with the growth of the training sets. Differences between decision tree learning (C4.5), neural network (BpNet), and decision rule learning (AQ15) are relatively small, although only AQ15 precisely learned all concepts in Experiment 5. The 4.8% average error

Table 19.2: Numbers of rules representing concepts learned by different methods (R-complexity)

	The Relative Size of the Set of Training Examples in Each Experiment				
	Experiment 1 (6%, 3%)*	Experiment 2 (10%, 10%)*	Experiment 3 (15%, 10%)*	Experiment 4 (25%, 10%)*	Experiment 5 (100%, 10%)*
Genetic Alg. (CFS)	49	45	51	48	41
Neural Nets (BpNet)	35	26	12	22	12
Decision Trees (C4.5)	3.1	2.8	2.5	2.5	2.5
Decision Rules (AQ15)	2.6	2.2	2.0	1.6	1.6
Decision Rules (AQ17-HCI)	2.4	2.0	1.6	1.6	1.6

*In each (x%, y%), x denotes positive training examples and y negative training examples.

rate of the BpNet-generated concepts was primarily because of an inadequate learning of concepts C1 and C4. Also, decision trees generated by C4.5 produced some error even when 100% positive examples were given. This error may be reduced if the function for converting trees into rules is applied (this, however, involves pruning a tree and simplifying rules). For AQ15-generated descriptions, errors in experiments 2–4 were primarily the result of errors in learning concept C1. The results of the constructive rule learning program AQ17-HCI show that all the concepts were precisely learned when the program was given 15% positive and 10% negative examples. Also, it generated the best performing descriptions in the first experiment.

One interesting finding is that increasing the number of training examples in experiments 1 to 5 resulted in only a slight improvement in the performance of the CFS-generated descriptions (from 21.3% to 16.3%). Other interesting findings are that even with 100% positive examples, the neural net, the genetic algorithm, and, to a smaller degree, the decision tree method did not learn the concept precisely. The CFS classifier system does not seem well-suited for classification-type problems. To further test this finding, this chapter reports results involving other genetic algorithm-based systems.

Table 19.2 gives the average R-complexity of the descriptions learned from different training sets. This measure gives a clear division between symbolic and subsymbolic methods. The symbolic methods generate descriptions that are ten times simpler than the subsymbolic methods. The results in this table are correlated with the results in Table 19.1. Methods that better perform in terms of predictive accuracy consistently yield simpler concept descriptions.

As mentioned earlier, target concepts were generated by human subjects, and therefore, the study favored methods that use symbolic representations because

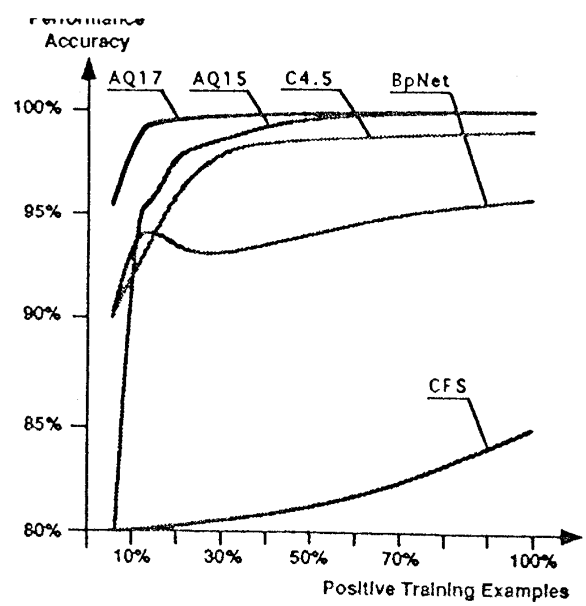

Figure 19.6: Learning curves for concepts in the ROBOTS domain for the fixed number of 10% negative examples

such representations are more closely related to human representations. Studying how systems learn such human-generated concepts is important for applications where knowledge that needs to be acquired is in such forms and/or applications where the knowledge learned needs to be understood by human experts. There are problem domains in which these factors are not relevant. Next, two studies present a wider range of problems involving both DNF-type and non-DNF-type concepts.

19.5 SECOND STUDY: THE MONK'S PROBLEMS

This study reports results from a performance comparison of different learning algorithms on three problems defined in the ROBOTS domain (Thrun et al., 1991). The so-called MONK's problems address three machine learning problems. Problem 1 is a DNF-type problem. Next is an "m-of-n," non-DNF-type problem. The concept to be learned requires a very complex DNF expression to describe it in terms of the available attributes. Problem 3 is a DNF-type, but the learning data set contains noise.

Problem M1: *Head shape is the same as the body shape, or color of the jacket is red.* Training set contains 124 randomly selected examples. There is no noise.

Problem M2: *Exactly two of the six given attributes take their first value.* For example, if attributes Head Shape and Body Shape take value *round*, which is the first value in their value set, then no other attribute may take the first value in its value set. Training set contains 169 randomly selected examples. There is no noise.

Problem M3: *Jacket is green and holding a sword, or jacket is not blue and body is not octagonal.* Training set contains 122 randomly selected examples. There is 5% noise in the data.

The tested algorithms fall into 4 categories:

- Neural Networks — **Backpropagation** (McClelland and Rumelhart, 1988), **Cascade Correlation** (Fahlman and Lebiere, 1990)
- Decision Trees — **ID3** (Quinlan, 1986), **Assistant Professional** (Cestnik, Kononenko and Brotko, 1987), **ID5R** (Utgoff, 1990), **IDL** (Van de Velde, 1989), **ID5R-hat** (Utgoff, 1990), **TDIDT** (Quinlan, 1986), **PRISM** (Cendrowska, 1988)
- Decision Rules — **AQ14-NT** (Pachowicz and Bala, 1991), **AQR, CN2** (Clark and Niblett, 1989), **AQ15** (Michalski et al., 1986), **AQ15-GA** (Vafaie and DeJong, 1993), **AQ17-DCI** (Bloedorn and Michalski, 1992), **AQ17-FCLS** (Zhang and Michalski, to appear), **AQ17-HCI** (Wnek and Michaleski, 1991), **AQ17** (Bloedorn, Michalski and Wnek, 1993)
- Inductive Logic Programming: mFOIL (Dzeroski, 1991)

Table 19.3 shows all reported results (Thrun et al., 1991). No one classifier based on genetic algorithms was tested as a separate program. In the AQ15-GA program, genetic algorithms are used in conjunction with AQ15. Genetic algorithms are used to explore the space of all subsets of a given attribute set, and AQ15 is used to build concept descriptions. This multistrategy approach improves performance accuracy of the symbolic learning system while the M3 problem is learned.

Problem M1 is of similar complexity to the C1–C5 ROBOTS problems, and it is easily learned by decision rule algorithms, AQ-15 and AQ17-HCI. Backpropagation and ID3 cannot learn concept M1 precisely; however, in both neural nets and decision trees paradigms, one can find programs that correctly learned descriptions (Cascade Correlation, Assistant Professional).

Problems M2 and M3 are difficult for selective decision rule and decision tree algorithms AQ15 and ID3. The learned descriptions have either high R-complexity (problem M2) or contain rules that cover noisy examples (problem M3). These problems were not learned as well by a hybrid of decision rules and decision trees, i.e., decision lists (CN2 algorithm [Clark and Niblett, 1989]). This suggests that techniques other than those implemented in these programs are required to solve this kind of problem.

The hypothesis-driven constructive induction method implemented in AQ17-HCI changes the representation space by narrowing and/or expanding the initial set of attributes. The method analyzes inductive hypothesis generated by a selective program and removes and/or generates new attributes. The new attributes are patterns found either in conditions or in the rules. This is sufficient to solve the M3 problem. Problem M2, however, still remains hard[4]. A solution lies in another type of change in the representation, i.e., attribute generation based on combining initial attributes using logical and/or algebraic operators (Bloedorn and Michalski, 1992). An initial integration of constructive induction methods was done in the AQ17 program. The multistrategy constructive induction program AQ17 learned all three MONK's problems.

19.6 THIRD STUDY: THE nDmC LEARNING PROBLEMS

This study uses another artificial domain to test twelve DNF-type concepts and is based on the experiments conducted by Spears and Gordon (1991). The experiments involved learning concepts in a designed domain defined by 4 nominal attributes, each having 4 distinct values. Therefore, the description space consisted of 256 examples (vectors of attribute values). There were twelve DNF-type target concepts, differing from each other in the number of rules (the number of disjunctions) and in the number of conditions in disjunctions (the conjunctions in the rules). All twelve concepts can be characterized by the formula nDmC, in which n, the number of disjunctions, varied from 1 to 4, and m, the number of conjunctions, varied from 1 to 3.

Spears and Gordon first compare three learning methods. Two symbolic methods represented by C4.5, a decision tree learning program, and AQ14, a deci-

[4] Problem M2 was later learned with 100% accuracy by AQ17-HCI as a result of detecting xor-patterns (Wnek, 1993).

Table 19.3: Summary of results for MONK's problems

Paradigm		Program	Prediction Accuracy: DNF-type (no noise)	Prediction Accuracy: Non-DNF (m-of-n)	Prediction Accuracy: DNF-type (noise)
Neural Nets	* (1)	Backpropagation	100%	100%	93%
	(1)	Cascade Correlation	100%	100%	97%
Decision Trees	(2)	Assistant Professional	100%	81%	100%
	* (3)	ID3	99%	68%	94%
	(3)	ID3 (no windowing)	83%	69%	96%
	(3, 4)	ID5R	82%	69%	95%
	(4)	ID5R-hat	90%	66%	—
	(4)	IDL	97%	66%	—
	(4)	TDIDT	76%	67%	—
	(5)	PRISM	86%	73%	90%
Decision Rules	(6)	AQ14-NT	100%	77%	100%
	(3)	AQR	96%	80%	87%
	(3)	CN2	100%	69%	89%
	(6)	AQ15	100%	77%	84%
	(6)	AQ15-GA	100%	87%	100%
	¶ (6)	AQ17-DCI	100%	100%	97%
	(6)	AQ17-FCLS	100%	93%	97%
	*¶ (6)	AQ17-HCI	100%	93%	100%
	¶ (6)	AQ17	**100%**	**100%**	**100%**
Inductive Logic Programming (2)		mFOIL	100%	69%	100%

* Programs compared in the first study. ¶ Constructive induction programs. Experiments were performed at the following laboratories: 1) School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA; 2) AI Laboratory, Josef Stefan Institute, Ljubljana, Slovenia; 3) Institute for Real-Time Computer Control Systems and Robotics and University of Karlsruhe, Karlsruhe, Germany; 4) Artificial Intelligence Laboratory, Vrije Universiteit Brussel, Brussels, Belgium; 5) AI-Lab, Institute for Informatics, University of Zurich, Switzerland; 6) Center for Artificial Intelligence, George Mason University, Fairfax, Virginia, USA.

sion rule learning program. Subsymbolic methods were represented by GABIL—Genetic Algorithms Batch Incremental Learner. They conclude that AQ14 is the best performer and uses some of AQ's strategies to improve GABIL. The resulting multistrategy system, Adaptive GABIL, is finally evaluated using the same problems.

Tables 19.4 and 19.5 show the results from testing the systems according to the prediction accuracy and the convergence criteria. The prediction accuracy is an

Table 19.4: Prediction accuracy in the four DNF categories

Paradigm (Program)	Prediction Accuracy			
	1DmC	2DmC	3DmC	4DmC
Genetic Alg.				
(GABIL)	96%	93%	90%	89%
(Adaptive GABIL)¶	97%	96%	95%	94%
Decision Trees				
(C4.5) *	98%	95%	89%	84%
Decision Rules				
(AQ14) *	99%	97%	96%	95%

*Programs compared in the first study
¶ Multistrategy learning program

average over all values on a learning curve. The convergence criterion is the number of events seen before a 95% prediction accuracy is maintained (Valiant, 1984). The results in the tables were averaged for each DNF category over three cases (m=1..3).

Problems labeled 1DmC and 2DmC are similar to problems C1–C5 defined in the ROBOTS domain as far as the complexity of descriptions is concerned. In learning such problems, the symbolic learning program AQ14 outperformed the other three programs both in terms of predictive accuracy and convergence to 95%. For the remaining problems, 3DmC and 4DmC, AQ14 maintains the best prediction accuracy. However, the Adaptive GABIL algorithm that combines symbolic

Table 19.5: Convergence to 95% in the four DNF categories

Paradigm (Program)	Convergence (no examples needed to achieve 95% accuracy)			
	1DmC	2DmC	3DmC	4DmC
Genetic Alg.				
(GABIL)	94	169	151	167
(Adaptive GABIL)¶	63	83	84	88
Decision Trees				
(C4.5) *	96	135	209	206
Decision Rules				
(AQ14) *	33	52	102	105

* Programs compared in the first study
¶ Multistrategy learning program

and subsymbolic strategies strongly outperformed the decision tree learning algorithm.

19.7 SUMMARY AND FUTURE WORK

From the multistrategy learning point of view, it is important that capabilities and limitations of different learning strategies and paradigms are well understood. The goal of this study was to make experiments that would help to develop insights into the performance of diverse learning approaches on selected classes of learning problems.

One finding is that symbolic methods outperformed subsymbolic methods in learning DNF-type problems. We found that the performance accuracy of symbolic methods was high, the convergence to the target concept was fast, and the learned descriptions matched or closely matched the target concepts and were easy to understand. In addition, preliminary results show that the symbolic methods performed very well with DNF-type problems with noise, which contradicts a sometimes expressed belief that neural nets are particularly good for such problems, and symbolic methods are not. The most surprising result, however, was that symbolic methods employing constructive induction performed on the par with neural nets on learning non-DNF-type concepts, such as "m-of-n." For such problems, neural nets were supposed to be superior because the problems are easily representable by such nets. Multistrategy induction methods (such as those implemented in the AQ17 family and Adaptive GABIL), although at an early stage of development, have already shown an improved performance over monostrategy methods.

The performance of the programs was analyzed using a *diagrammatic visualization* system, DIAV. This system, working on line with a learning program, turned out to be a very useful tool for visualizing learned and target concepts, comparing the learned concepts, and presenting errors in learning (an "error image"). The method was also exceptionally useful for visualizing concepts learned by subsymbolic methods and comparing them with concepts learned by symbolic methods, such as neural net learning and genetic algorithm learning. Concept images helped comprehending knowledge encoded in a neural network or in a population of classifiers. In addition, the visualization method enabled us to determine the R-complexity of the concepts learned by the subsymbolic methods.

Among important topics for the future is the application of the methods to a wider range of non-DNF-type problems, such as learning a text-to-speech mapping (Sejnowski and Rosenberg, 1987; Dietterich, 1990), and to randomly generated problems in order to evaluate an overall performance of the methods. Future research might also compare the performance of the methods in learning from noisy data and inconsistent examples and in learning imprecisely defined or flexible concepts (Bergadano et al., 1992).

ACKNOWLEDGMENTS

The authors thank Ken Kaufman, Gheorghe Tecuci, and Jianping Zhang for comments on earlier drafts of this chapter. They are grateful to Jayshree Sarma and Ashraf A. Wahab for performing experiments with backpropagation algorithm BpNet and genetic algorithm learning program CFS and to Rick Riolo for providing a copy of his CFS-C software.

This research was done in the Center for Artificial Intelligence at George Mason University. The Center's research is supported in part by the National Science Foundation under Grant No. IRI-9020226, in part by the Defense Advanced Research Projects Agency under the grant administered by the Office of Naval Research No. N00014-91-J-1854, and in part by the Office of Naval Research under Grant No. N00014-91-J-1351.

References

Bergadano, R., Matwin, S., Michalski, R.S. and Zhang, J., "Learning Two-tiered Descriptions of Flexible Concepts: The POSEIDON System," *Machine Learning*, Vol. 8, pp. 5–43, 1992.

Bloedorn, E. and Michalski, R.S., "Data-driven Constructive Induction in AQ17-DCI: A Method and Experiments," *Reports of Machine Learning and Inference Laboratory*, No. MLI-92-03, George Mason University, 1992.

Breiman, L., Freedman, J.H., Olhsen, R.A. and Stone, C.J., *Classification and Regression Trees*, Wadsworth International Group, Belmont, CA, 1984.

Cendvowska, J., "PRISM: An Algorithm for Inducing Modular Rules," in *Knowledge Acquisition for Knowledge-Based Systems*, B.R. Gaines and J.H. Boose (Eds.), Academic Press, New York, pp. 253–274, 1988.

Cestnik, B., Kononenko, I. and Bratko, I., "ASSISTANT 86: A Knowledge-Elicitation Tool for Sophisticated Users," *Progress in Machine Learning*, I. Bratko and N. Lavrec (Eds.), Sigma Press, Wilmslow, 1987.

Clark, P. and Niblett, T., "The CN2 Induction Algorithm," *Machine Learning*, Vol. 3, pp. 261–283, 1989.

Dietterich, T.G., Hild, H. and Bakiri, G., "A Comparative Study of ID3 and Backpropagation for English Text-to-Speech Mapping," *Proceedings of the 7th International Conference on Machine Learning*, Austin, TX, pp. 24–31, Morgan Kaufmann, San Mateo, CA, 1990.

Dzeroski, S., "Handling Noise in Inductive Logic Programming," MS Thesis, Dept. of EE and CS, University of Ljubljana, Slovenia, 1991.

Fahlman, S.E. and Lebiere, C., "The Cascade-Correlation Learning Architecture," in *Advances in Neural Information Processing Systems*, Vol. 2, Morgan Kaufmann, San Mateo, CA, 1990.

Fisher, D.H. and McKusick, K.B., "An Empirical Comparison of ID3 and Back-propagation," *Proceedings of IJCAI-89*, pp. 788–793, AAAI Press, Menlo Park, CA, 1989.

Hempel, C.G., *Aspects of Scientific Explanation*, The Free Press, New York, 1965.

Holland, J.H., "Escaping Brittleness: The Possibilities of General-purpose Learning Algorithms Applied to Parallel Rule-Based Systems," *Machine Learning: An Artificial Intelligence Approach*, Vol. II, R.S. Michalski, J.G. Carbonell, T.M. Mitchell (Eds.), pp. 593–624, Morgan Kaufmann, Los Altos, CA, 1986.

Holland, J.H. and Reitman, J.S., "Cognitive Systems Based on Adaptive Algorithms," *Pattern-directed Inference Systems*, D.A. Waterman and F. Hayes-Roth (Eds.), Academic Press, New York, 1978.

Kaufman, K.A., Michalski, R.S. and Schultz, A.C., "EMERALD 1: An Integrated System of Machine Learning and Discovery Programs for Education and Research," *Reports of Machine Learning and Inference Laboratory*, No. MLI-89-12, George Mason University, 1989.

Kodratoff, Y., "Induction and the Organization of Knowledge," in *Machine Learning: A Multistrategy Approach*, Vol. IV, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA, 1994.

McClelland, J. and Rumelhart, D., *Explorations in Parallel Distributed Processing*, MIT Press, Cambridge, MA, 1988.

Michalski, R.S., "AQVAL/1—Computer Implementation of a Variable-valued Logic System VL1 and Examples of Its Application to Pattern Recognition," *Proceedings of the First International Joint Conference on Pattern Recognition*, pp. 3–17, Washington, DC, October 30-November 1, 1973.

Michalski, R.S., "Pattern Recognition as Knowledge-guided Computer Induction," *Reports of Computer Science Department*, No. 927, University of Illinois, Urbana, 1978.

Michalski, R.S., "A Theory and Methodology of Inductive Learning," *Artificial Intelligence*, Vol. 20, pp. 111–116, 1983.

Michalski, R., Mozetic, I., Hong, J. and Lavrac, N., "The AQ15 Inductive Learning System: An Overview and Experiments," *Reports of Machine Learning and Inference Laboratory*, No. MLI-86-6, George Mason University, 1986.

Mooney, R., Shavlik, J., Towell, G. and Gove, A., "An Experimental Comparison of Symbolic and Connectionist Learning Algorithms," *Proceedings of IJCAI-89*, pp. 775–780, AAAI Press, Menlo Park, CA, 1989.

Pachowicz, P.W. and Bala, J., "Improving Recognition Effectiveness of Noisy Texture Concepts," *Proceedings of the Eight International Workshop on Machine Learning*, pp. 625–629, Morgan Kaufmann, San Mateo, CA, 1991.

Pagallo, G. and Haussler, D., "Boolean Feature Discovery in Empirical Learning," *Machine Learning*, Vol. 5, No. 1, pp. 71–99, 1990.

Quinlan, J.R., "Induction of Decision Trees," *Machine Learning*, Vol. 1, No. 1, pp. 81–106, 1986.

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

Rendell, L.A. and Seshu, R., "Learning Hard Concepts through Constructive Induction: Framework and Rationale," *Computational Intelligence*, Vol. 6, pp. 247–270, 1990.

Riolo, R.L., "CFS-C: A Package of Domain Independent Subroutines for Implementing Classifier Systems in Arbitrary, User-defined Environments," *Technical Report*, Logic of Computers Group, Division of Computer Science and Engineering, University of Michigan, 1988.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J., "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D.E. Rumelhart and J.L. McCLelland (Eds.), The MIT Press, Cambridge, MA, 1986.

Sejnowski, T.J. and Rosenberg, C.R., "Parallel Networks That Learn to Pronounce English Text," *Complex Systems*, Vol. 1, pp. 145–168, 1987.

Spears, W.M. and Gordon, D.F., "Adaptive Strategy Selection for Concept Learning," *Proceedings of the First International Workshop on Multistrategy Learning*, Harpers Ferry, WV, pp. 231–246, George Mason University Press, Fairfax, VA, 1991.

Thrun, S.B., Bala, J., Bloedorn, E., Bratko, I., Cestnink, B., Cheng, J., DeJong, K.A., Dzeroski, S., Fahlman, S.E., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R.S., Mitchell, T., Pachowicz, P., Vafaie, H., Van de Velde, W., Wenzel, W., Wnek, J. and Zhang, J., "The MONK's Problems: A Performance Comparison of Different Learning Algorithms," *Technical Report*, Carnegie Mellon University, December 1991.

Utgoff, P.E., "Incremental Learning of Decision Trees," *Machine Learning*, Vol. 4, pp. 161–186, Kluwer Academic, Boston, MA, 1990.

Vafaie, H. and DeJong, K.A., "Improving the Performance of a Rule Induction System Using Genetic Algorithms," in *Machine Learning: A Multistrategy Approach*, Vol. 4, R.S. Michalski and G. Tecuci (Eds.), Morgan Kaufmann, San Mateo, CA, 1994.

Valiant, L.G., "A Theory of the Learnable," *Communications of the ACM*, Vol. 27, pp. 1134–1142, 1984.

Van de Velde, W., "IDL, or Taming the Multiplexer," *Proceedings of the Fourth European Working Session on Learning*, Pitman, London, UK, 1989.

Weiss, S.M. and Kapouleas, I., "An Empirical Comparison of Pattern Recognition of Neural Nets and Machine Learning Classification Methods," *Proceedings of IJCAI-89*, p. 781–787, AAAI Press, Menlo Park, CA, 1989.

Wnek, J. and Michalski, R.S., "Hypothesis-driven Constructive Induction in AQ17: A Method and Experiments," *Proceedings of the IJCAI-91 Workshop on Evaluating and Changing Representations*, K. Morik, F. Bergadano and W. Buntine (Eds.), pp. 13–22, AAAI Press, Menlo Park, CA, 1991.

Wnek, J. and Michalski, R.S., "A Diagrammatic Visualization of Learning Processes," to appear in *Reports of Machine Learning and Inference Laboratory*, George Mason University, 1993.

Wnek, J., "Version Space Transformation with Constructive Induction: The VS* Algorithm," *Reports of Machine Learning and Inference Laboratory*, No. MLI-92-01, George Mason University, 1992.

Wnek, J., "Hypothesis-driven Constructive Induction," *Ph.D. Dissertation*, University Microfilms International, Ann Arbor, MI, 1993.

Zhang, J. and Michalski, R.S., "Combining Symbolic and Numeric Representations in Learning Flexible Concepts: The FCLS Systems," *Machine Learning*, to appear.