CHAPTER 8

Other Issues in Concept Induction

In the previous chapters we have examined a variety of methods for inducing logical, threshold, and competitive concepts, along with techniques for organizing them into larger knowledge structures like decision lists, inference networks, and concept hierarchies. However, our treatment of these algorithms made a number of simplifying assumptions about the nature of the training and test data.

In particular, we assumed that the domains were generally free of attribute and class noise, and that most features were actually useful in the prediction process. We also supposed that the aim of learning was to improve the accuracy of predicting a given class attribute, and that this attribute took on a small set of nominal values. Finally, we assumed that instances were described as a set of attribute-value pairs rather than in some more sophisticated formalism, and that each instance was completely described in terms of these attribute values.

In this chapter we consider some extensions to the basic algorithms that go beyond these assumptions, letting them deal with noisy domains, irrelevant features, numeric prediction, unsupervised data, first-order descriptions, and missing attributes. In each case, we attempt to formulate these extensions in the most general way possible, as they often apply not to specific methods but to entire classes of them.

8.1 Overfitting and pruning

As we noted in Chapter 1, some domains contain noise in attribute or class information, yet many of the algorithms we have considered assume noise-free data. Given either form of noise, constructing concept descriptions that cover all the training cases will produce an idiosyncratic knowledge structure that performs poorly on separate test data.

Researchers have explored a number of extensions designed to prevent such *overfitting* of the training data, and here we consider the main alternatives along these lines.

8.1.1 Pruning knowledge structures

Perhaps the most straightforward approach, often called *pre-pruning*, operates in conjunction with any induction method that moves from simple structures to more complex ones. This includes techniques for the divisive construction of concept hierarchies, separate-and-conquer methods for decision lists, and many schemes for determining the structure of inference networks. Rather than continuing to extend a knowledge structure by adding branches, rules, or nodes whenever these improve accuracy on the training set, one uses some more conservative test to determine whether this addition is likely to improve behavior on separate test cases. Only if this appears to hold does the algorithm continue; otherwise it halts and returns the current structure, even though this does worse on the training data than would the extended one.

For example, in constructing a decision tree, a pre-pruning version of the DDT algorithm from Chapter 7 would check, each time it is about to extend the tree downward by adding children to an existing node, whether this extension is actually desirable. If not, then the method would not add the children and it would halt tree construction, at least in this portion of the tree.

Another approach to minimizing the effect of noise on induction algorithms, often called *post-pruning*, operates in the opposite direction. Here one uses some induction method to construct a knowledge structure that minimizes error on the training set, ideally so that it correctly classifies all observed instances.[1] Naturally, this produces an overly complex and idiosyncratic knowledge structure if the data are noisy. However, one then goes back through the structure, pruning components that do not significantly aid in classification, again using some conservative test that estimates accuracy on test data.

For instance, one can use the DDT algorithm to construct a decision tree that correctly classifies all of the training cases (or as many as possible if some are inconsistent), then prune this tree back to avoid overfitting. The standard scheme uses a bottom-up greedy algorithm that, on each step, considers the result of deleting one of the lowest subtrees. If the evaluation function suggests that this removal would not hurt accuracy on a test set, then it is pruned; otherwise, the subtree is retained. The process then considers another of the lowest subtrees, tries pruning it, and continues until no further deletions are justified.

1. Some algorithms must be modified somewhat to handle inconsistent data sets that contain identical instances with different associated classes.

Table 8-1. A logical decision list from the cell domain (a) before pruning and (b) after post-pruning has removed the last `lethargia` rule and eliminated one condition in another rule.

```
(a) lethargia :- one-tail, thin-wall, two-nuclei.
    healthy :- one-tail, thin-wall, one-nucleus.
    lethargia :- two-tails, thick-wall, one-nucleus.
    lethargia :- two-tails, two-nuclei.
    healthy :- true.
(b) lethargia :- one-tail, thin-wall, two-nuclei.
    healthy :- one-tail, thin-wall, one-nucleus.
    lethargia :- two-tails, one-nucleus.
    healthy :- true.
```

Although pre-pruning and post-pruning were originally developed in the context of decision-tree induction, they can be combined with any learning technique that deals with more than the simplest structures. For example, one can use a similar greedy method to prune decision lists by deciding to remove (or not to add) individual conditions or by deciding to eliminate (or not to create) entire rules. Table 8-1 shows a logical decision list for the cell domain before and after a post-pruning scheme has been applied.[2]

One can adopt similar ideas in deciding when to stop removing nodes and links from (or when to stop adding them to) a multilayer inference network. One can even collect statistics on stored cases in a nearest neighbor framework, then remove instances that historically have made

2. One advantage of this technique is that it lets the learner remove conditions added early to a rule without sacrificing those added later, whereas post-pruning in decision trees typically removes tests only in the reverse order from which they were added. For this reason, some systems convert decision trees to decision lists and then prune the latter.

poor predictions or add ones that would have fared well. The idea of pruning is one of the most general within machine learning, although its incarnation naturally differs across frameworks.

8.1.2 Evaluation metrics for pruning

We have considered a number of pruning algorithms, but we have been intentionally vague in our treatment of the evaluation measures they use in making decisions. We have done so because, generally speaking, one can combine any pruning scheme with any evaluation metric. The literature contains a variety of such metrics, but they fall into three basic classes, which we now consider.

The simplest method involves some form of significance test on the training data. For example, in deciding whether to prune a subtree from a concept hierarchy, one can use a χ^2 test to determine whether the unpruned tree is significantly better at discriminating classes than the pruned version.[3] This test measures the degree to which different values of an attribute are associated with different classes. Like all statistical tests, χ^2 requires one to specify a confidence level. Thus, one might want to include the subtree if there exists a 5% chance that the observed difference is due to noise, or one might be more conservative and include it only if there is a 1% chance. In general, the latter criterion will require either greater differences or more data than the former. This general class of metrics is sometimes said to rely on *resubstitution error*, because it involves resubstituting the training data into the induced structure and using it to estimate the error rate.

Some statisticians argue that resubstitution error is a poor measure even when combined with statistical tests. Instead, they recommend holding some training data back as a separate *evaluation set*, then using relative accuracy on this set to make pruning decisions. This approach, which is closely related to techniques for *cross validation*, means that less data are available for training, but it can provide a better estimate of accuracy on novel test cases because the induction algorithm has not seen instances in the evaluation set. As with resubstitution estimates, one can combine this scheme with tests for statistical significance, but they are not strictly needed; one can simply continue to simplify or extend the concept hierarchy, decision list, or inference network until accuracy on the evaluation set decreases.

3. This statistic is not reliable for data sets containing four or fewer instances, but one can use Yates' correction in such circumstances.

The third broad class of metrics implement a tradeoff between accuracy on the training set and some measure of simplicity. We saw a very simple form of this idea in Chapter 2, in the context of the IHC algorithm, but more sophisticated versions are possible. The most common of these formulations, known as *minimum description length*, is based on information theory. In particular, it measures the total number of bits needed to encode the hypothesized knowledge structure H and the number of bits needed to encode exceptions to H in the training data D, then takes their sum. Mathematically, this is expressed as

$$-log\ p(H|D) = -log\ p(H) + -log\ p(D|H) \quad ,$$

where one desires to minimize the left-hand term. This measure requires one to specify a coding scheme for hypotheses and data but, having done this, one can combine it with any induction algorithm to determine when to stop adding structures (pre-pruning) or when to remove them (post-pruning).

A closely related idea can be used with algorithms for revising weights in multilayer inference networks. This variation, often called *weight decay*, incorporates a bias toward simplicity into methods like backpropagation by modifying the error term to prefer weights that are very small or large, but not intermediate in size (say around 1). A way to implement this idea is to modify the error function to be minimized (and which is used in determining the updating rule) so that it includes not only the mean squared difference between predicted and observed outputs, but also a term for the sum of the squared weights in the network. A generalization of this scheme rewards the weight-revision method for using weights that are close to any of a small set. Such modified error functions operate in a similar way to minimum description length for pre-pruning, in that they cause the learner to halt not when it minimizes the normal definition of error on the training set, but rather when it minimizes the revised measure that prefers fewer distinct parameters.

8.1.3 Dynamic pruning methods

The approaches described above all attempt to handle noise by altering the knowledge structure created during learning, typically by restricting the number of effective parameters. However, another viable approach involves inducing an unrestricted knowledge structure but then using it in some way that avoids overfitting. Such *dynamic pruning* has a quite different flavor from the methods considered so far, but it can have a similar effect on performance.

One simple technique for dynamic pruning involves the selective use of the components in a learned knowledge structure. For example, given an induced concept hierarchy, one might decide to halt the sorting of a test case before it reaches a terminal node because the statistics for a particular branch suggest that it is unreliable. Similarly, one can decide not to base a prediction on a particular rule in a decision list or on a given instance stored in case memory if its behavior on the training set suggests it cannot be trusted. This approach, designed mainly for incremental algorithms that must make decisions before seeing all the training data, can be adapted to emulate the effects of most pruning techniques and can incorporate most evaluation measures.

Another version of dynamic pruning relies on the notion of *voting*. This approach seems most natural with Bayesian schemes, as one can compute the probability of each concept description given the test case, then use these scores to make a weighted prediction. The k nearest neighbor method embodies a similar idea, especially with versions that weigh the votes of distant cases less than the votes of nearer ones. Both approaches are designed to minimize reliance on individual descriptions and thus should guard against overfitting effects.

However, one can easily adapt the voting scheme to other frameworks. Most techniques for determining weights in a multilayer neural network assign initial weights randomly. As a result, different runs typically produce networks with different final weights. In making predictions, one can simply take the majority vote of these alternative knowledge structures. A more sophisticated approach assigns a weight or probability to each network proportional to its accuracy on the training set, then bases the final prediction on their weighted vote.

The adaptation of this idea to logical concept induction is less obvious, as most such methods are deterministic in nature. However, rather than carry out a greedy search through the space of logical descriptions, one can instead carry out a beam search with some small beam width, which produces a set of alternative hypotheses as output. Again, one can simply take the vote of these descriptions to classify test cases or one can estimate their predictive accuracies and use these scores to weight their votes accordingly.

8.2 Selecting useful features

One of the central issues in induction concerns the selection of useful features. Although most learning methods attempt to either select attributes or assign them degrees of importance, both theoretical analyses and experimental studies indicate that many algorithms scale poorly to domains with large numbers of irrelevant features. For example, the number of training cases needed for simple nearest neighbor (see Chapter 5) to reach a given level of accuracy appears to grow exponentially with the number of irrelevant features, independent of the target concept. Even methods for inducing univariate decision trees, which explicitly select some attributes in favor of others, exhibit this behavior for some target concepts. And some techniques, like the naive Bayesian classifier (Chapter 4), can be very sensitive to domains with correlated attributes. This suggests the need for additional methods to select a useful subset of features when many are available.

8.2.1 Feature selection as heuristic search

Following the theoretical bias we have used elsewhere in this book, we can view the task of feature selection as a search problem.[4] As Figure 8-1 depicts, each state in the search space specifies a subset of the possible features, and one can impose a partial ordering on this space, with each child having exactly one more feature than its parents. As with any such problem, techniques for feature selection must take a stance on four basic issues that arise in the heuristic search process.

First, one must determine the starting point in the space, which in turn determines the direction of search. For instance, one might start with no features and successively add attributes, or one might start with all attributes and successively remove them. The former approach is sometimes called *forward selection*, whereas the latter is known as *backward elimination*. One might also select an initial state somewhere in the middle and move outward from this point.

A second decision involves the organization of the search. Clearly, an exhaustive search of the space is impractical, as there exist 2^a possible subsets of a attributes. A more realistic approach relies on a greedy method to traverse the space. At each point in the search, one considers

4. One can also adapt techniques for feature selection to determine attribute weights, but we will not focus on that issue here.

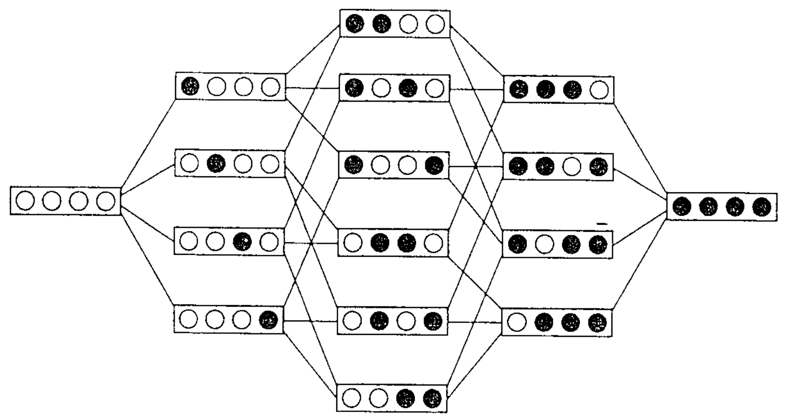

Figure 8-1. Each state in the space of feature subsets specifies the attributes that one might use during induction. Note that the states (in this case involving four features) are partially ordered, with each of a state's children (to the right) including one more attribute (dark circles) than its parents.

local changes to the current set of attributes, selects one, and then iterates, never reconsidering the choice. A related approach, known as *stepwise* selection or elimination, considers both adding and removing features at each decision point, which lets one retract an earlier decision without keeping explicit track of the search path. Within these options, one can consider all states generated by the operators and then select the best, or one can simply choose the first state that improves accuracy over the current set.

8.2.2 Filter and wrapper approaches to feature selection

A third issue concerns the evaluation strategy used to select among alternative subsets of attributes. One broad class of strategies considers attributes independently of the induction algorithm that will use them, relying on general characteristics of the training set to select some features and to exclude others. These are sometimes called *filter* methods, because they filter out irrelevant attributes before the induction process occurs. In contrast, *wrapper* methods generate a set of candidate features, run the induction algorithm with these features on the training data, and use the accuracy of the resulting description to evaluate the feature set. Within this approach, one can use accuracy on the training set or a more conservative estimate computed by cross validation.

The general argument for wrapper approaches is that the induction method planned for use with the feature subset should provide a better estimate of accuracy than a separate measure that has an entirely different inductive bias. The major disadvantage of wrapper methods over filter schemes is the former's computational cost, which results from calling the induction algorithm for each feature set considered. This cost has led some researchers to invent ingenious techniques for speeding the evaluation process that cache hypothesized structures or that reduce the number of training cases used for accuracy estimates.

Finally, one must decide on some criterion for halting search through the space of feature subsets. Within the wrapper framework, one might stop adding or removing attributes when none of the alternatives improves the estimate of classification accuracy, one might continue to revise the feature set as long as accuracy does not degrade, or one might continue generating candidate sets until reaching the other end of the search space and then select the best. Within the filter framework, one criterion for halting notes when each combination of values for the selected attributes maps onto a single class value. Another alternative simply orders the features according to some usefulness score, then determines some break point below which it rejects features.

Note that the above methods for feature selection can be combined with *any* induction algorithm to increase its learning rate in domains with irrelevant attributes. The effect on behavior may differ for different induction techniques and for different target concepts, in some cases producing little benefit and in others giving major improvement. But the basic idea of searching the space of feature sets is conceptually and practically distinct from the specific induction method that benefits from the feature-selection process.

8.3 Induction for numeric prediction

Most research in machine learning has focused on the induction of knowledge for classification purposes, and our treatment in previous pages has reflected this emphasis. However, some tasks instead involve *regression* – the prediction of numeric values – and there also exists a sizable literature on this topic. In this section we consider those methods from Chapters 5, 6, and 7 that lend themselves directly to the induction of regression knowledge, some other techniques that can be adapted to regression problems, and a new class of methods designed to find explicit functional expressions.

Naturally, overfitting can be as much an issue for regression as for classification, since noise also occurs in numeric domains. Most of the responses we considered in Section 8.1, including pre-pruning, post-pruning, and voting, also apply to learning for purposes of numeric prediction, as do evaluation schemes like significance tests, separate evaluation sets, and minimum description length, although some changes are needed in their details. The same comments hold for feature selection.

8.3.1 Direct approaches to regression

Some techniques, such as nearest neighbor methods, make no distinction between classification and regression problems. The unedited versions of this approach simply store training cases in memory during learning, and during testing they simply predict the value of the missing attribute. Whether this attribute takes on nominal values (as with class names) or numeric ones makes no difference to the induction process, and prediction requires only minor changes, such as replacing the voting process in k nearest neighbor with a weighted averaging scheme.

Yet other methods, although we introduced them in the context of classification, were actually designed with numeric prediction in mind. Two important examples are the LMS algorithm from Chapter 3 and the closely related backpropagation method from Chapter 6. Recall that the performance elements for these techniques compute a weighted sum of the observed, predictive attributes. The thresholds on these sums serve partly to transform numeric predictions into classification decisions.

The LMS and backpropagation algorithms themselves require no modification to learn from numeric data. One simply replaces the 1 or 0 that represents the class name in the training data with some real value. As before, the induction method computes the difference between the predicted and observed values, then uses this information to direct a gradient-descent search through the weight space in an effort to minimize mean-squared error or some other function.

As we noted in Chapter 6, one can use backpropagation with inference networks that are composed entirely of linear units, entirely of spherical units, or some combination of these representations. One method commonly used for numeric prediction, radial basis functions, computes a linear combination of the outputs of spherical units. These are sometimes called *mixture models* because they describe a regression function as a weighted mixture of Gaussian or similar distributions.

8.3.2 From classification to regression

One can also transform an induction algorithm designed for classification tasks into one that handles regression problems. The first step involves a simple change in representation; rather than associating a class name with each decision region, one instead associates some numeric value. Thus, one can attach real or integer values to the right-hand sides of rules in a decision list, to the terminal nodes of decision trees and concept hierarchies, or to the predicted attribute of a competitive description. The performance component typically remains unchanged, with the system determining the best rule, node, or competitor for a given test case and then simply predicting the requested value.

The induction process can also carry over nearly unchanged from classification domains. For example, one can adapt the DDT algorithm from Chapter 7 to select attributes in a univariate decision tree based not on their ability to discriminate among different classes, but on their ability to separate different numeric values that occur in the training data. The NSC method from Chapter 5 can be altered in a similar way to select conditions for rules that distinguish among different real or integer values.

Figure 8-2 (a) shows an example *regression tree* from the height/girth domain, which takes the form of a univariate decision tree but which specifies numeric values at its terminal nodes rather than class names. Figure 8-2 (b) shows the decision regions produced by this knowledge structure, each of which is a rectangle with an associated numeric score. Here the terminal nodes predict a person's health insurance premium as a function of his or her height and girth. The tree first divides people into those shorter and taller than 5.5 feet. It partitions the first group further into people with girth less than 2.5 feet (with the predicted rate 200) and those greater (with predicted rate 300). Similarly, the taller group is divided into those with girth less than 3.0 feet (with predicted rate 220) and those greater (with predicted rate 330).

The figure shows that this tree produces a step function over the dependent variable. Although this may be desirable for some domains (like predicting insurance costs), in others (like predicting a person's

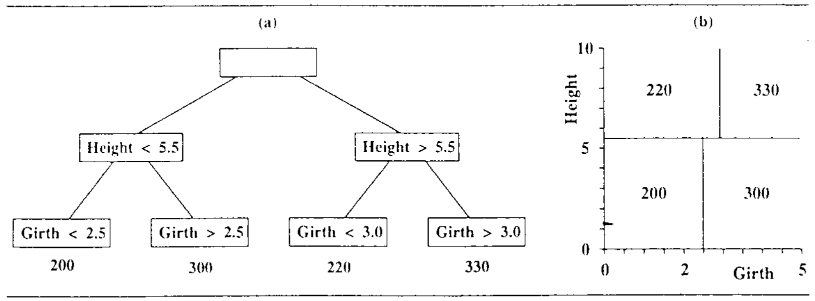

Figure 8-2. (a) The structure of a univariate regression tree that predicts insurance rates as a function of height and girth, and (b) the decision boundaries generated by this tree.

weight) a more continuous function seems desirable. One modification to this basic approach involves the use of multivariate *splines* during the prediction process. The spline interpolation process produces continuous curves with gradual transitions across the decision boundaries, which can greatly increase predictive accuracy in domains that involve continuous functions.

8.3.3 Inducing numeric functions

Many adaptations of classification learning schemes to numeric prediction assume that the target function can be divided into a set of decision regions, each with a constant value, and thus they focus on identifying the appropriate boundaries. Another line of research instead assumes that some more complex function holds over the entire instance space and attempts to discover its nature.

The simplest example of this approach is *linear regression*. Given a single predictive variable x and predicted variable y, this method assumes a function of the form $y = ax+b$ and algorithmically determines values for the slope a and intercept b that minimize squared error on the training data. The statistical literature abounds with variations on this method, some of which are designed to handle linear combinations of multiple predictive variables.

However, most work on this topic within the machine learning literature has focused on heuristic methods for finding nonlinear functions. One simple method of this sort, which we will call NLF (numeric law

Table 8-2. The NLF algorithm's rediscovery of Kepler's third law of planetary motion, relating a planet's distance D to its period P. The method defines new variables in terms of the observables until finding one with a constant value.

PLANET	D	P	D/P	D^2/P	D^3/P^2
MERCURY	0.387	0.241	1.606	0.626	0.998
VENUS	0.724	0.616	1.175	0.851	0.999
EARTH	1.000	1.000	1.000	1.000	1.000
MARS	1.524	1.882	0.810	1.234	1.000
JUPITER	5.199	11.880	0.438	2.275	0.996
SATURN	9.539	29.461	0.324	3.091	1.000

finder), induces numeric laws by searching the space of products and ratios of primitive variables, in an attempt to find some higher-order term that has a constant value. The algorithm starts by computing correlations among pairs of observable numeric attributes. NLF defines a new ratio term upon noting a positive correlation between two variables and a new product term for a negative correlation. If a new term T's values are constant, or if they are linearly related to another term, the method posits a law based on this relation. Otherwise, the algorithm considers T's correlations with other features and continues to search the space of higher-order terms.

Table 8-2 shows NLF's reconstruction of Kepler's third law of planetary motion, which relates a planet's period P to its distance D from the sun. Given observed values for these variables (in years and astronomical units), the algorithm notes that D increases with P, suggesting that their ratio *might* be constant. This relation does not actually hold, but upon computing values for the ratio D/P, NLF finds that this higher-order variable decreases as D increases, suggesting that it examine their product D^2/P. Again, this term is not constant, but its values increase as those of D/P decrease, suggesting their product D^3/P^2. Upon noting that the value of this term is always near 1, NLF postulates Kepler's law, that D^3/P^2 is constant.

One can view the discovery procedure embodied in NLF as constructing an inference network in which the nodes are arithmetic combinations of their children. In this sense, it bears some similarity to the methods described in Chapter 6 for constructing such networks, although the heuristics for directing search are quite different. Moreover, NLF attempts to find a functional description that is easily interpretable, whereas methods like cascade correlation are concerned only with predictive accuracy.

Other approaches to finding interpretable numeric laws are also possible. One such method attempts to fit predefined templates to training data, determining the best parameters as in linear regression but supporting more complex functional forms. One can also combine partitioning schemes with methods like linear regression or NLF to find numeric relations that involve different constants in different regions. One technique of this sort adapts the EM algorithm (Chapter 7) to determine a partitioned linear description of the training data. Another uses a version of the DDT algorithm (Chapter 7) to induce a decision tree in which the terminal nodes specify linear regression equations that incorporate multiple variables.

8.4 Unsupervised concept induction

Until now we have focused our attention on supervised induction tasks, in which the performance task is to predict the class of an instance. However, as we noted in Chapter 1, not all learning tasks involve a special attribute that corresponds to the class. For such *unsupervised* induction problems, we must specify some other performance problem.

One natural performance task involves the prediction of any unknown attribute values from the others, without knowledge at the outset about which attributes will be predicted or which the predictors. This task is sometimes called *flexible prediction*, because one can predict any feature from any others, and sometimes *pattern completion*, because one must complete a partial description of a pattern (i.e., an instance). An obvious performance measure for this task is simply the average predictive accuracy over all attributes. Another metric involves the ability to estimate the probability that each possible instance will be observed.

8.4.1 Simple approaches to unsupervised learning

Some induction methods require no modification at all to learn from unsupervised data and to perform flexible prediction. For example, the basic nearest neighbor method from Chapter 5 stores and retrieves training cases in the same way regardless of whether they have special

class attributes, although some edited versions do need class information. Most methods for constructing probabilistic inference networks (see Chapter 6) also operate in the same manner whether or not one attribute has different status from the others. And some partitioning schemes, such as k-means clustering and the EM algorithm (Chapter 7), also make no special use of class attributes, treating them as just another feature when they are available.

For other induction algorithms, one can always adapt a supervised method to an unsupervised task through a simple transformation. Given k attributes, one runs the algorithm k times, in each case with a different feature playing the role of the class attribute one aims to predict. The result is k different classifiers, each designed to accurately predict one attribute as a function of the others.[5] For example, one might use the NSC algorithm from Chapter 5 to construct k separate decision lists or the backpropagation method from Chapter 6 to determine weights on k different inference networks. Figure 8-3 (a) depicts the structures that result when three attributes are involved and there are two nonterminal symbols (hidden units) in each network.

8.4.2 More sophisticated unsupervised methods

This divide-and-conquer scheme may work well in domains where the performance task involves prediction of isolated attributes, but it can well have difficulty when asked to predict many features for each test case. In such situations, one may instead prefer a single knowledge structure that maximizes predictive ability across all attributes. Some supervised methods are easy to adapt along these lines. As we noted in Chapter 7, one can modify the evaluation metric used in constructing concept hierarchies to reward partitions that lead to improved prediction over all attributes rather than just the class name. The result is a hierarchically organized set of clusters that describe similar instances and that can be used in flexible prediction.

Research on unsupervised learning in inference networks has often taken a different approach. One idea here is to include each feature as both an input node and an output node, but to place them in a single knowledge structure that shares nonterminal nodes or hidden units. Figure 8-3 (b) illustrates such a structure that involves three input units

5. One can use a variant of this idea with numeric attributes, using the techniques discussed in Section 8.3 to find k knowledge structures for numeric prediction.

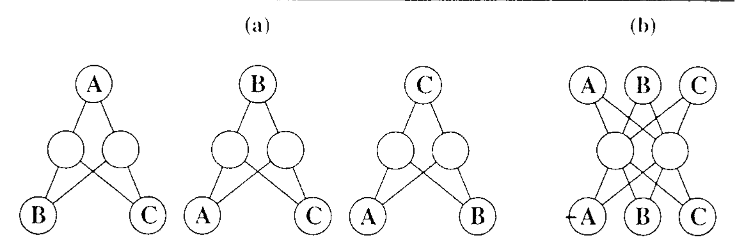

Figure 8-3. Knowledge structures for unsupervised learning that consist of (a) three separate inference networks, one for predicting each attribute from the others, and (b) a single inference network that shares hidden units for predicting various attributes.

and three output units. One then uses an induction method like backpropagation to reduce the predictive error on an output node as a function of other attributes, repeating this process for each output. This technique tends to create weights for links to nonterminal symbols that reflect clusters of instances, much as in unsupervised techniques for the induction of concept hierarchies.

Of course, these two approaches to unsupervised induction – creating separate knowledge structures and one shared structure – represent extremes along a continuum. Given an unsupervised learning task with k features, one may instead want to create more than 1 but fewer than k predictive structures. This makes sense in domains where certain subsets of attributes are correlated with each other but where the different subsets are uncorrelated. If one can identify the appropriate j sets, one can apply an induction algorithm to each in turn to produce j separate knowledge structures that, when taken together, should maximize predictive ability over all of the attributes.

8.5 Inducing relational concepts

Although our examples of induction have so far assumed attribute-value descriptions, we noted in Chapter 1 that learning can also occur over representations that involve relations among objects or entities. In other words, one can carry out induction not only over propositional descriptions but also over first-order statements. Work on relational induction dates back at least to 1970, but in recent years it has become a very

active area under the name *inductive logic programming*, as most approaches represent the result of learning as Horn clause programs.

In this section we examine the adaptation of methods from previous chapters to the task of relational induction. We will focus on logical and competitive schemes, as this area contains very little work that builds on threshold units. In our examples, we will use a version of the cell domain from previous chapters, revised so that each instance describes two cells using not their actual values but rather the relations between them. Our representation will assume 6 possible predicates – `tails>(X,Y)`, `nuclei>(X,Y)`, `shade>(X,Y)`, `tails=(X,Y)`, `nuclei=(X,Y)`, and `shade=(X,Y)` – that, respectively, describe situations in which `X` has more tails than `Y`, `X` has more nuclei than `Y`, `X` is darker than `Y`, and `X` and `Y` are the same on these dimensions.[6]

8.5.1 Induction of relational conjunctions

Much of the work on relational induction has focused on logical conjunctions and has emphasized exhaustive "incremental" methods like ISG and IGS from Chapter 2. In fact, one can use these very algorithms on relational data, provided that one replaces the scheme for generating new hypotheses in response to each training case. Recall that both methods retain a set of descriptions that are consistent with the data observed so far, but that IGS starts with a very general hypothesis and generates more specific successors, whereas ISG operates in the opposite direction.

We can extend the IGS algorithm to handle relational data by letting it entertain relational conjunctions as hypotheses and, when a candidate H misclassifies a negative instance N as positive, by letting the algorithm generate all minimal ways to add literals to H that prevent it from matching N. For example, suppose H is the two-literal description `tails=(X,Y)` $\wedge$ `tails=(Y,X)`, and that this hypothesis incorrectly matches the negative training case

```
tails=(a,b),tails=(b,a),nuclei>(a,b),shade=(a,b),shade=(b,a).
```

If we assume the learning algorithm knows the literals `nuclei=(X,Y)`,

6. Because equality is symmetric, an instance that includes `tails=(a,b)` will also include `tails=(b,a)`. Thus, for 2 cells, there are 12 possible relations, no more than 6 of which are present in any instance, giving $3^3 = 27$ possible instances.

nuclei>(X,Y), and nuclei>(Y,X) are mutually exclusive, that the same holds for color relations, and that equality relations are redundant, then IGS will generate four minimally more specific hypotheses:

```
tails=(X,Y) ∧ tails=(Y,X) ∧ nuclei=(X,Y)
tails=(X,Y) ∧ tails=(Y,X) ∧ nuclei>(Y,X)
tails=(X,Y) ∧ tails=(Y,X) ∧ shade>(X,Y)
tails=(X,Y) ∧ tails=(Y,X) ∧ shade>(Y,X) .
```

In more complex situations, some minimal changes will require the addition of two or more literals, but none occur in this example.[7] Of course, future negative cases may show that one or more of these revised hypotheses are still too general, leading to further specialization, or future positive instances may show them to be overly specific, leading to their removal.

We can alter the ISG algorithm in an analogous manner. Recall that this technique initializes memory to contain a single, maximally specific hypothesis based on the first positive training instance. Let us suppose that this hypothesis is

```
tails=(X,Y) ∧ tails=(Y,X) ∧ nuclei>(Y,X) ∧ shade>(X,Y) ,
```

where we have replaced the names of specific cells with the variables X and Y. Given a second positive instance

```
tails=(c,d), tails=(d,c), nuclei>(c,d), shade>(c,d) ,
```

the ISG algorithm would generate two minimally more general hypotheses that match this new case:

```
tails=(X,Y) ∧ tails=(Y,X) ∧ nuclei>(X,Y)
tails=(X,Y) ∧ tails=(Y,X) ∧ shade>(X,Y) .
```

Each revised conjunction is a generalization of the initial hypothesis that matches the new positive instance, but future positive cases may require further revisions or negative ones may lead to their elimination.

We can view the process of finding the most specific (or least general) description in terms of an exhaustive search through the space of partial matches. From this perspective, one starts with no mappings between the literals in hypothesis H and instance I, then selects the first literal

7. One can also compute differences between the current negative case and recent positive instances to constrain the search for more specific hypotheses.

in H and considers each way of mapping it into one of I's literals. Each successful mapping, which must involve the same predicate and produce consistent bindings, leads to a successor state that includes those bindings as additional constraints that must hold later in the search. The null mapping also generates a successor state, in which the H literal maps onto no I literal.

One then repeats this process for the second literal in H, generating successor states for each successful mapping onto the remaining literals in I, again including the null mapping. This scheme continues until all literals in H have been mapped onto those in I, at which point it removes all terminal nodes in the search tree that are strictly more specific than others, then returns this reduced set. Each such mapping specifies a subset of the literals in H that correspond to a most specific generalization with the training instance I.

Recall that, in the attribute-value version of ISG, there was always a unique way to make the current hypothesis more general, so that no search through the space of hypotheses was needed. In the relational formulation just given, this situation no longer holds, and ISG must carry out a breadth-first search in the same way as the IGS method. However, some recent approaches to inductive logic programming retain the idea of a maximally specific (or least general) generalization for relational domains by focusing on the idea of unification rather than matching.

For example, given the initial hypothesis and positive instance above, this scheme would produce the single revised hypothesis

```
tails=(U,V) ∧ tails=(V,U) ∧ tails=(W,Z) ∧
tails=(Z,W) ∧ nuclei>(U,V) ∧ shade>(Z,W) ,
```

which is the most specific description that unifies with both the previous hypothesis and the instance. This approach has the advantage that the hypothesis set does not grow as the algorithm encounters more positive instances. However, the size of the description itself can grow, whereas in the previous method, the size of the set grows but the size of hypotheses actually shrinks. Which approach does better in practice, in terms of computational cost and accuracy, is an open question, although the unification approach currently dominates the literature.

Naturally, heuristic versions of the above methods also exist. One can adapt the IHC algorithm from Chapter 2 to carry out incremental hill climbing through the space of relational conjunctions in a similar way, using operators that make the current hypothesis either more specific or more general. Nonincremental techniques like EGS and ESG are also possible, although these methods use training cases only to evaluate alternative hypotheses, not to generate them.

8.5.2 Induction of complex relational structures

Given techniques for inducing conjunctive relations, one can embed these methods in algorithms for learning decision lists, inference networks, and concept hierarchies, as we saw in Chapters 5, 6, and 7, respectively. In general, handling relational data with these more complex structures introduces no additional issues beyond those that arise in learning the conjunctions themselves. That is, once the subroutines for finding individual rules are replaced by a method for finding relational conjunctions, the algorithms will operate in relational domains.

One exception involves techniques for theory-based specialization, like the ICE algorithm described in Chapter 6. Recall that this approach assumes that the target concept is a specialization of a known inference network, and that each training case leads to a "proof" that, when compiled, produces a logical rule that covers the instance and similar ones. Although not emphasized in Chapter 6, most work on this topic assumes a relational representation for both instances and background knowledge, the latter described as a function-free Horn clause program.

In propositional domains (where predicates take no arguments), the compilation of proofs is straightforward, in that one simply takes the features that appear as terminal nodes in the proof tree and makes them the conditions of the new rule. The process is similar for relational inference networks, but here one must be concerned about how variables are shared across literals.

The standard approach uses a trace of the unifications made at each node in the proof tree to determine which arguments must be the same and which need not. Often this process leads to compiled rules with literals that share variables precisely when these arguments take on the same values in the training case, but this result does not always occur. In some cases, the method leads to the introduction of inequalities, not present in the original inference network, that must hold for the new rule to match. We will see an example of this effect in Chapter 10, in the context of learning search-control knowledge.

8.5.3 Competitive descriptions and structural analogy

Although most work on relational induction has assumed the use of logical interpreters, some efforts have dealt instead with competitive schemes, specifically variants of the nearest neighbor algorithm. Recall that this method computes the distance between a test instance and a stored training case, using some measure that rewards common features. For attribute-value representations, this computation is simple and efficient, involving the pairwise comparison of values for each attribute. However, the situation for relational descriptions is much more complex, in that it requires the computation of partial matches between the test instance and each relational case in memory.

Research on this topic, which is often called *structural analogy*, relies on techniques very similar to the relational version of the ISG algorithm described above. That is, analogical methods typically attempt to find the maximal partial match (most specific generalization) between the test instance (sometimes called the *target*) and the stored case (sometimes called the *source*). For example, suppose memory contains the source case

```
tails=(a,b), tails=(b,a), nuclei>(b,a), shade>(a,b) ,
```

and that one encounters the target instance

```
tails=(c,d), tails=(d,c), nuclei>(c,d), shade>(c,d) .
```

For this pair of instances, there are two structural analogies that constitute maximal partial matches:

```
tails=(a,b) ∧ tails=(b,a) ∧ nuclei>(a,b)
tails=(a,b) ∧ tails=(b,a) ∧ shade>(a,b) ,
```

where a maps onto d and b maps onto c in the first analogy and where these bindings are flipped in the second. Note that this example is identical to the one from Section 8.5.1 involving most specific generalizations, except that the partial match occurs between two instances, rather than between an hypothesis and an instance.

Because there can be more than one such match, the analogical system must decide which one to select for use. Work on structural analogy is often unsupervised and, to the extent that a performance task is specified, it involves the inference or prediction of literals omitted from the test instance. The task of analogical matching over relational descriptions is difficult enough that most work in the area has dealt with memories containing a small number of stored cases. Nevertheless, conceptually the approach has much in common with the simpler competitive methods, such as nearest neighbor and its relatives, that we considered in earlier chapters. Two important differences revolve around the idea of merging two separate analogies to increase predictive power and using background knowledge or higher-order relations to constrain the matching process or to bias the distance metric. These ideas are equally applicable to propositional versions of nearest neighbor, but they seldom occur in that paradigm.

8.6 Handling missing features

The algorithms we have considered in previous chapters were originally designed to accept complete training cases, in which all attribute values were present, and similarly to handle test cases that are complete except for those attributes to be predicted. However, there are many situations in which some features are absent from either training or test data, and which can require modifications of the basic algorithms. Here we discuss the issues that arise in such situations and consider some of the available responses.

8.6.1 Assumptions about missing features

Most work in machine learning assumes that, when attribute values are absent, the omission is due to some random process, such as failure to record a value during transcription. In other words, the standard view is that information regarding which features are present in the training data is not useful to the learning process. Our treatment will rely on this standard assumption, as nearly all research on missing attributes has occurred in that context.

However, this assumption is violated in some data sets, and we should discuss them briefly. In some situations, such as medical diagnosis, where the questions asked by doctors are correlated with the disease they conclude, the features that are absent can provide useful information for the learning process. In other situations, as in voting records, where a missing representative's vote may indicate intentional absence,

"missing" may be treated best as a separate value during both training and testing. Researchers must use their judgement in deciding which interpretation is most appropriate for a given domain.

A different issue arises in relational domains. Here one must decide whether the absence of a literal L in some instance I means that L is false for I or simply that it was not observed for some reason. For this issue, most researchers have assumed that absence implies falsehood, as this lets them treat each training and test case as complete and thus simplifies the algorithms for learning and performance. For this reason, most work on relational induction has not dealt directly with missing features, and we will not consider such algorithms here.

8.6.2 Missing features during testing

Before we can understand how to handle missing features during learning, we must first examine their effect on testing and performance. Many competitive techniques like nearest neighbor and probabilistic methods can simply ignore attributes that are missing and use only the observed features in computing a score for each competitor. However, for logical descriptions and threshold units the situation is more complicated. The former's reliance on an all-or-none matching scheme assumes that all important features are present;[8] the latter's partial match is more flexible but its thresholds are designed with all features in mind. As a result, the most common alteration for both approaches involves inferring the values of missing attributes and using these in making predictions.

The simplest such inference method uses the overall modal (most frequent) value for nominal attributes and Boolean features and uses the overall mean for numeric attributes. But one can also apply this idea locally. For example, consider a univariate decision tree that contains a test relying on a missing attribute; in this case one can use the modal or mean value for just those training instances used to construct that portion of the tree, which makes sense given that the test on this attribute occurs only in the context of those above it.

Another approach assigns missing values probabilistically, much as in the voting scheme described in Section 8.1.3. For example, given a test case with a missing feature that occurs at a node N in a uni-

8. Of course, the matching process for logical concepts is unaffected by a missing feature if that feature is not included in the logical description; our concern here is with attributes that occur in such descriptions but that have missing values.

variate decision tree, one sorts the case fractionally down all branches below the node N to determine which terminal nodes it reaches, then uses the predictions of these nodes in a weighted voting scheme to select the class. One can also use this idea with logical decision lists and logical inference networks. For knowledge structures that incorporate threshold units, which typically have separate input nodes for alternative attribute values, one can activate these nodes as a function of each value's probability, then predict the output (class) node as usual.

An even more sophisticated scheme involves the induction of knowledge structures for predicting an attribute's value from the values of others, then using these structures to infer the attribute's value (using either the most likely value or its probability distribution) when it is missing. This approach is very similar to that taken by some unsupervised learning methods, which seem better suited to data sets with missing features due to their emphasis on flexible prediction.

8.6.3 Missing features during training

Similar issues arise when features are absent during the learning process, and similar responses to this problem suggest themselves. For most probabilistic methods, one simply does not update counts for omitted attributes, and for nearest neighbor methods one simply stores abstract cases that make no commitment about these features. The competitive nature of such approaches means that they require little modification.

For logical and threshold techniques, all of the methods described for the testing situation are available during training. Thus, one can simply insert the overall mean or modal values, provided that the method is nonincremental and thus has these statistics available from the training data. Techniques for inducing decision trees and concept hierarchies can instead use local estimates, at least those that construct trees in a divisive or top-down manner.[9] Again, one can also use a probabilistic approach that generates fractional instances for use in computing evaluation scores or gradients. These probabilities may be based on simple summary statistics or on the predictions of classifiers induced specifically to infer missing values.

9. Incremental learning methods must retain running estimates of means and modes, basing their estimates on the data seen so far.

The issue of missing features has received much less attention in the literature than the other topics we have considered in this chapter. Accordingly, there have been fewer alternatives proposed and fewer experimental or theoretical comparisons than for other variations on the task of concept induction.

8.7 Summary of the chapter

In this chapter we considered six major extensions that cut across the induction methods described earlier. Some of these techniques are designed to improve the robustness of the induction process. In particular, pruning methods can greatly improve learning algorithms' resistance to overfitting on noisy data. Techniques for selecting subsets of features for use in prediction can significantly improve the learning rate in domains with many irrelevant attributes. And methods for inferring the values of missing attributes can aid learning when training or test cases are incomplete.

Other extensions we examined instead broaden the type of induction task addressed. For example, algorithms for learning regression knowledge let the performance element predict numeric values rather than class names. Unsupervised induction methods let one learn from data in which there is no special class attribute, and thus support more flexible prediction strategies. Finally, techniques for relational learning allow induction using instances and hypotheses described not as attribute-value pairs but as first-order relations.

Although these methods considerably extend the range and robustness of the induction process, they clearly build directly on the simpler algorithms we discussed in earlier chapters. Each technique is a straightforward variation on some more basic scheme, rather than being essentially different, as the literature on machine learning often suggests.

Exercises

1. Assume a pre-pruning variant of the DDT algorithm (Chapter 7) that extends the decision tree downward only if the new children reduce the number of classification errors made by the current node by more than two. Show the decision tree produced by this strategy on the training data in Figure 7-1 from Chapter 7, and compare it to the tree in Figure 7-2.

2. Use the DTL algorithm (Chapter 7) to generate a decision list from the decision tree that the (unmodified) DDT method induces from the data in Figure 5-4. Apply greedy post-pruning (Section 8.1.1) to this decision list, using accuracy on the training set (not on a separate evaluation set) as the evaluation metric, and considering rules and conditions in the order in which they occur.
3. Consider the stored training cases in Figure 5-9 (a). Show two test instances that the k nearest neighbor method, with $k = 2$, would classify differently from simple nearest neighbor. Report the distance computations involved in each decision.
4. Assume a simple logical representation of knowledge, known as a *determination*, that takes the form of a table in which each cell corresponds to a specific combination of attribute values and contains the predicted class. Further assume that each cell is filled with the most frequent class observed for that combination or, if none have been seen, with the most frequent class. Starting with a table that incorporates all four attributes from the cell domain, use the greedy wrapper method from Section 8.2 to search the space of tables that incorporate only some attributes. Use accuracy on the training data in Figure 7-1 as the evaluation metric, but change the `color` of the last `healthy` instance to `dark`. Show the score for each candidate set of features considered during the search.
5. Generate six instances that satisfy Galileo's law of uniform acceleration, $D = 9.8T^2$. Show the terms generated by the NLF algorithm (Section 8.3.3) in rediscovering this law.
6. Ignoring the cell wall thickness and the classes for the data in Figure 5-5, show a set of logical rules that, to the extent possible, predict each of the remaining attributes in terms of the other two features. Compute percentage accuracy for each predicted attribute.
7. Suppose one has an hypothesis about the definition of a new kinship relation, `parent(X,Y)` $\wedge$ `parent(Y,Z)` $\wedge$ `male(X)`, but that this matches the negative case `parent(joe,sam)`, `parent(sam,mary)`, `male(joe)`, `male(sam)`, `female(mary)`. Show the set of minimally more specific hypotheses about the kinship relation, assuming knowledge that `male` and `female` are mutually exclusive.
8. Suppose one's hypothesis about a new kinship relation, `parent(X,Y)` $\wedge$ `parent(Y,Z)` $\wedge$ `male(X)` $\wedge$ `male(Y)` $\wedge$ `female(Z)`, fails to match the positive training case `parent(joe,mary)`, `parent(mary,sam)`, `male(joe)`, `female(mary)`, `male(sam)`. Show the set of minimally more general hypotheses that match this case, based on the matching scheme in Section 8.5.1. Show the unique hypothesis generated by the unification approach described later in that section.
9. Assume that the `tails` attribute is unspecified for the even-numbered instances in Figure 7-1. Given the decision tree in Figure 7-2, compute the percentage accuracy on these instances if one infers the modal value when an attribute is missing from a test case. Show the decision tree that the DDT algorithm would construct if given these instances as training data and if it used the same inference strategy in this context.

Historical and bibliographical remarks

The notion of overfitting has roots in the statistical literature on higher-order regression, but Breiman, Friedman, Olshen, and Stone (1984) first raised the issue in work on decision-tree induction and used a post-pruning technique in their CART system. Quinlan (1986) popularized the idea within the machine learning community, with his ID3 algorithm for decision-tree induction initially using pre-pruning but his later C4.5 system (1993) emphasizing post-pruning. Mingers (1989b) reports an experimental evaluation of different methods for pruning decision trees, and Hassibi and Stork (1993) describe a post-pruning scheme for multilayer neural networks.

Quinlan and Rivest (1989) describe the use of minimum description length in pruning decision trees, while Iba, Wogulis, and Langley (1988) present a similar method for decision lists. Weigend, Huberman, and Rumelhart (1990) incorporate a penalty term into their error function for backpropagation in neural networks. Fisher (1989) reports a dynamic pruning scheme for use with probabilistic concept hierarchies, Aha (1992) describes a related method for handling noise in the nearest neighbor framework, and Buntine (1990) uses a weighted voting technique that combines predictions from multiple decision trees.

Research on feature selection, as separate from other aspects of induction, also has a long tradition in statistics, but mainly in the context of regression (e.g., Devijver & Kittler, 1982). Within machine learning, Kira and Rendell (1992) describe a filter approach that orders features by relevance, whereas Almuallim and Dietterich (1991) present an exhaustive method that generates minimal sets of predictive attributes.

John, Kohavi, and Pfleger (1994) provide the first general formulation of the wrapper approach, which they combine with decision-tree induction, as do Caruana and Freitag (1994). Moore and Lee (1994), Langley and Sage (1994b), and Aha and Bankert (1994) use the wrapper framework to select relevant features for nearest neighbor prediction, while Langley and Sage (1994a) use it to remove correlated attributes in naive Bayesian classifiers. Langley (1994a) reviews recent work in this area.

Work on regression also has a long history in statistics. Widrow and Hoff (1960) used the LMS algorithm to improve numeric predictions and, more recently, Weigend et al. (1990) and others have put more sophisticated neural network techniques to similar ends. Kibler, Aha, and Albert (1989) and Moore and Lee (1994) report uses of nearest neighbor in predicting real values; Jordan and Jacobs (1993) have used the EM algorithm to generate mixtures of Gaussians for regression purposes, and Moody and Darken (1991) have used backpropagation on radial basis functions with the same goal. Breiman et al. (1984) introduce the idea of a regression tree that can be induced using DDT-like methods, and Friedman's (1991) MARS system extends this approach to use splines for smoothing during prediction. Quinlan's (1993a) M5 is another variation of the DDT scheme that stores regression equations at the terminal nodes of trees, Weiss and Indurkhya (1993) describe a method that forms numeric decision lists, and Rao and Lu (1992) achieve similar effects with a variant of k-means clustering that uses regression equations rather than means. Gerwin (1974) did early work on the discovery of explicit functional expressions, but Langley, Bradshaw, and Simon's (1983) BACON popularized this approach within the machine learning community. More recently, Falkenhainer and Michalski's (1986) ABACUS, Nordhausen and Langley's (1990) IDS, and Zytkow, Zhu, and Hussam's (1990) FAHRENHEIT have extended this approach to broader classes of expressions and more situations.

There exists a large literature on unsupervised learning within the areas of numerical taxonomy and clustering (Everitt, 1980), aimed largely at data analysis. Michalski and Stepp (1983) were influenced by this work in their formulation of "conceptual clustering" and in development of their CLUSTER/2 system, which incorporated k-means clustering (Chapter 7) as a subroutine. Cheeseman et al.'s (1988) AUTOCLASS, which uses the EM algorithm (Chapter 7), and Hanson and Bauer's (1989) WITT, which uses agglomerative clustering, constitute other extensions of this tradition. A separate line of work, concerned with modeling the incremental nature of human learning, was initiated by Feigenbaum's (1961) EPAM model. Lebowitz's (1987) UNIMEM, Fisher's (1987) COBWEB, and McKusick and Langley's (1991) ARACHNE explore variations of this scheme that use more flexible representations, and Fisher, Pazzani, and Langley (1991) contains a number of related papers. Martin's (1989) CORA adapts a simple rule-induction method to unsupervised learning and flexible prediction, and Kohonen (1982), Rumelhart and Zipser (1985), and Grossberg (1987) have used competitive learning in neural networks to achieve similar effects.

Much of the machine learning research during the 1970s focused on the induction of relational conjunctions. Winston's (1975) heuristic method and Mitchell's (1977) exhaustive candidate elimination algorithm are perhaps the best known from this period, but Vere (1975) and Hayes-Roth and McDermott (1978) also contributed to this area. A few researchers, such as Anderson and Kline (1979) and Langley (1987), developed related methods for inducing logical decision lists. Plotkin (1970) introduced the notion of least general generalization based on unification, but Muggleton and Feng (1992) popularized the idea in the context of inductive logic programming. Quinlan's (1990) FOIL, a system for inducing logical decision lists, uses a relational variant of the HGS algorithm, whereas Muggleton and Feng's (1992) GOLEM incorporates a technique similar to the HSG algorithm. Proceedings of meetings on inductive logic programming have appeared annually since 1992, and a text (Lavrac & Dzeroski, 1993) even exists on the topic.

Research on structural analogy dates back to Evans (1968), but more recent work has been done by Greiner (1988), Falkenhainer, Forbus, and Gentner (1989), Thagard, Holyoak, Nelson, and Gochfeld (1990), and Gentner and Forbus (1991), the last two having addressed not only matching but also issues of retrieval. Hall (1989) reviews work on analogy through that date, some of which deals with analogical matching and other work with problem solving, which we discuss in Chapter 11.

Concern with missing attributes has come relatively late to machine learning and thus far has received only minor attention. Quinlan (1986) reports on a variety of methods for inferring missing features in the context of decision-tree induction, while Ahmad and Tresp (1993) compare different neural network techniques for handling this problem. Rao, Greiner, and Hancock (1994) describe a method that takes advantage of missing attributes to constrain the induction task.