CHAPTER II

BACKGROUND

"We're right over Illinois yet,"
[said Huck Finn to Tom Sawyer in their boat.]
"and you can see for yourself that Indiana ain't in sight...
Illinois is green, Indiana is pink.
You show me any pink down here if you can. No sir, it's green."
"Indiana pink? why, what a lie!"
"It ain't no lie; I've seen it on the map, and it's pink."

S.L. Clemens, 1894.
(alias Mark Twain)
Tom Sawyer Abroad, ch. 3

The first efforts in machine learning centered on self-organizing systems that adapted to their environment by adjusting internal parameters through feedback. Such work included self-organizing and adaptive control systems[121, 136], early work on genetic algorithms and evolutionary approaches to learning [33], and neural modelling through the use of perceptrons [100]. Researchers in adaptive control focussed primarily on continuous systems and eventually formed their own separate branch that is today far removed from the machine learning field (as a subfield of artificial intelligence). Research in perceptron learning suffered a major setback as a result of Minsky and Papert's [78] theoretical investigation which pointed out serious limitations of single-layer linear threshold perceptron networks. Today, this field of machine learning is regaining strength under the banner of Connectionism [66]. Genetic Algorithms appear today in Holland's Classifier systems [47] and other systems [40].

One remarkably successful learning system of this early era is Samuel's checker player program [103] which provided a "proof of concept" that a machine learning program can acquire useful knowledge autonomously. Samuel's checker playing program adapted its playing strategy as it played games with opponents. The program was eventually able to consistently defeat its own author, and attained the regional championship level.

About the time of the demise of perceptrons, the field experienced a shift in its basic approach to the problem. Winston's thesis [133] on learning high-level conceptual descriptions ushered in the second stage of work in machine learning; a stage characterized by an emphasis on knowledge-intensive methods of representation and methodology. The dominant view was that substantial *a priori* knowledge in the system was needed to make it learn—to drive its "sophisticated" learning mechanisms [60, 63]. Major problems with the approaches of this era were due to the use of algorithms with high computational complexity and to a lack of analysis and understanding of the systems developed.

The third stage of machine learning research was fueled by the advent of expert system technology. The need for learning techniques to automate the construction of knowledge bases—the knowledge engineering process—turned attention back to finding methods for the automatic induction of rules. The time-consuming and often unsuccessful attempts by programmers to capture an expert's knowledge, and the inability of experts to describe their own skills in terms of rules or procedures, proved to be very substantial hurdles that were manifested in almost all attempts at building expert systems [113]. The hope is that this *knowledge acquisition bottleneck* [28] may potentially be circumvented by endowing programs with the ability to acquire knowledge on their own. Current approaches to machine learning span a wide spectrum of complexities: including pure syntactic induction, learning from advice and analogy, learning from observation and discovery, and explanation-based learning approaches that rely heavily on domain knowledge and semantics.

In this chapter we focus on reviewing the relevant literature on classification research. Other relevant work in machine learning is briefly reviewed in Chapter IX.

We begin with the Bayesian solution to the learning problem PE defined in Chapter I, since it is basic to all classification work. We then introduce previous work on decision tree generation for classification learning. We also include a comparison with an alternate, nonsymbolic approach, to learning classifications: neural networks. Neural networks and decision trees are currently the most popular methods used in industrial applications of machine learning.

2.1 The Bayesian Solution

Let TE be a training set of examples consisting of N $m+1$-tuples

$$\langle v_1, v_2, \ldots, v_m; C_j \rangle,$$

where each $v_i \in Range(A_i)$ is one of the values of the attribute $A_i \in A$, and $C_j \in C$ is one of the k classes. We assume that TE is drawn according to some probability distribution on the examples and classes. We denote the probability density of this distribution by $D(e, c)$. The set of examples is

$$E = \{\langle v_1, v_2, \ldots, v_m \rangle | v_i \in Range(A_i), A_i \in A\}$$

and the set of classes is

$$C = \{C_1, C_2, \ldots, C_k\}.$$

Given the distribution $D(e, c)$ on the sets E and C, we can define the two random variables V and L. V's range is the set of examples E while L's is the set of classes C. With this notation, the training set TE can be viewed as a set of N samples of the values of the pairs $\langle V; L \rangle$ that are governed by the joint density $D(e, c)$. Learning problem PE is equivalent to finding a classifying function f,

$$f : E \longrightarrow C$$

such that the error rate of f is minimum. That is, for $e \in E$,

$$\text{Prob}\{f(V) \neq L\} = \int_S \int D(e, c) de dc$$

is minimized. The set S over which integration takes place is the set of $m+1$-tuples $\langle e; c\rangle$ such that f's prediction for e's class differs from c:

$$S = \{\langle e; c\rangle | f(e) \neq c\}.$$

We use integration rather than summation (for the discrete variables) for convenience and to keep the notation general. Note that this definition allows for the possibility of the existence of tuples $\langle e; C_i\rangle$ and $\langle e; C_j\rangle$ such that $C_i \neq C_j$. In most practical domains, the vector e is sufficient to specify a unique class. In this case, the distribution D simply assigns a probability of 1.0 to one of the tuples above and zero to all the others.

Since we are dealing with classification problems, the classes are discrete. One of the integrals can therefore be immediately turned into a summation, so

$$\text{Prob}\{f(V) \neq L\} = \sum_{C_j \in C} \left(\text{Prob}\{L = C_j\} \int_{S^{C_j}} D^{C_j}(e) de \right)$$

where D^{C_j} is the distribution on E given that $L = C_j$, $D(e|C_j)$. The set S^{C_j} is the set of examples for which f does not assign the class C_j,

$$S^{C_j} = \{e | e \in E \wedge f(e) \neq C_j\}.$$

The solution to the problem, shown to be optimal by T. Bayes, is known as the *Maximum A Posteriori* (MAP) classifier. It is the classifier g that assigns the class C_j to example e, where C_j is the class in C for which

$$\text{Prob}\{L = C_j | e\}$$

is maximum among the classes in C. Note that this quantity can be evaluated using the joint density and Bayes rule since

$$\begin{aligned}
\text{Prob}\{L = C_j | e\} &= \frac{\text{Prob}\{L = C_j \;\wedge\; V = e\}}{\text{Prob}\{V = e\}} \\
&= \frac{\text{Prob}\{V = e | L = C_j\}\text{Prob}\{L = C_j\}}{\text{Prob}\{V = e\}} \\
&= \frac{\text{Prob}\{V = e | L = C_j\}\text{Prob}\{L = C_j\}}{\sum_{i=1}^{k} \text{Prob}\{L = C_i\}\text{Prob}\{V = e | L = C_i\}}
\end{aligned}$$

where all the probabilities above are, in principle, obtainable from the joint density $D(e,c)$. However, it may not be possible to always obtain the analytical solution to the required integration to get the marginal distributions.

Although the optimality of the solution was proven, it is usually not possible to compute this solution. The reason for this is that the joint density $D(e,c)$ is not known *a priori*. Note that if this density were known, then all other interesting conditional and non-conditional distributions can be obtained (see [132], p. 103). For example, to obtain the probability density of V given that $L = C_j$, $D(e|L = C_j)$, one would simply integrate

$$D(e|L = C_j) = \int_{S_{C_j}} D(e,c)dc$$

where S_{C_j} is the subset of the space for which L takes the value C_j. As mentioned earlier, one may not have enough knowledge about the spaces, or the analytical solution to the integration may be too difficult. In typical applications of the Bayesian approach, it is assumed that two distributions are given explicitly or obtainable easily: the probability of any value of V given the class (the value of L), and the prior probability of each of the classes, i.e. $\text{Prob}\{L = C_j\}$, for $j = 1, \ldots, k$. These two pieces of information represent the required *domain knowledge* of the problem. Note that once the densities $D(e|L)$ and $D(c)$ are given, a solution for the the class C_j that maximizes $\text{Prob}\{L = C_j|e\}$ is easily obtainable for any given example—any value e of V.

However, we are still left with the problem of obtaining the probability densities $D(e|L)$ and $D(c)$ since in a typical application one may only be given the training set. The straightforward approximation to circumvent this problem is to estimate the distributions from the training data. Besides the fact that this is computationally difficult since it requires estimating the probability of each possible combination of values and classes, little is known about the suboptimality of the answer obtained using these estimates [5]. Thus, we do not really know what kind of answer the Bayesian solution provides if the needed distributions are estimated from the data.

A recent and apparently successful addition to learning algorithms is AUTO-

CLASS [10]. Designed primarily for clustering applications, the AUTOCLASS program utilizes Bayesian classification techniques to induce both new classes and their characterizations. We discuss this algorithm in the next section after introducing parametric methods.

2.1.1 The PAC Learning Model

We introduced learning problems PE and CX in Chapter I. Both problems were defined in terms of a search for an optimal solution. Thus for every learning task, there exists a solution to the learning problem. Valiant [126] defined a class of learning problems by using a relaxed version of the definition of learning problem PE. Rather than requiring that the minimal error classifier be produced, two parameters were introduced: ϵ and δ representing an error rate and a confidence level, respectively. Instead of finding an optimal classifier, the learning algorithm is to produce, with arbitrarily high confidence, a classifier having an arbitrarily low error rate. The added restriction is that the classifier be produced in polynomial time [43]. Hence the goal is to find a Probably Approximately Correct (PAC) classifier.

More formally, given a class $\mathcal{C}$ of possible classifiers, and a fixed probability distribution on the examples, the following defines whether the class $\mathcal{C}$ is *learnable.*

Definition 2.1.1 A class of classifiers $\mathcal{C}$ is *PAC learnable* if for any δ and ϵ, $0 \leq \epsilon, \delta \leq 1.0$ there exist a polynomial learning algorithm G and an integer $poly(\frac{1}{\epsilon}, \frac{1}{\delta})$ such that for any training set with $N > poly(\frac{1}{\epsilon}, \frac{1}{\delta})$ examples, with probability at least $(1-\delta)$, G produces a classifier from $\mathcal{C}$ having error rate at most ϵ. The integer returned by $poly(\frac{1}{\epsilon}, \frac{1}{\delta})$ is polynomially related to the two arguments.

Note that this definition defines a class of PAC-learnable classifiers. Not every learning task need have a solution in a PAC-learnable class. Thus, work within the framework of PAC-learnability is mainly concerned with determining which classes of classifiers are learnable.

PAC learnability essentially characterizes the space of concepts learnable in the polynomial realm. Hence, we view it as a formal characterization of the class of concepts that our *learning machine*, defined in Chapter I, is capable of learning. Whereas our intuitive definition of learning did not specify the type of learning task, the PAC-learnability definition implicitly restricts the learning task to be that of concept learning. The parameters δ and ϵ give us the means to quantitatively express the "improvement in performance" we referred to in our intuitive definition of learning. For a critique of the PAC learning model along with other important notions in machine learning see [8].

2.2 Pattern Recognition Techniques

Other methods for learning classifiers from data have been used in the past. Most such methods, however, assume that the "variables" (attributes) are continuous-valued [46, 86, 89, 114]. Given that the attributes are continuous, one could do one of two things:

Parametric Estimation: which involves estimating the probability distributions of each of the classes over the variables. When a future example is to be classified, the *maximum likelihood principle* is applied to select the class with the highest probability given the data. This gives the Bayes optimal solution if the estimate of the distribution is correct and the classes have equal prior probabilities. If the distributions are unimodal, then finding the most likely class is easy. The difficulty arises when the distributions are multimodal. Partitioning the space into unimodal regions becomes necessary. Many issues come up in this paradigm: How to do the partitioning effectively? How to find a good parametric fit for the estimated distribution? What happens if the estimated distribution does not fit the data correctly? Maximum likelihood methods, however, are considered ill-founded when the classes have different prior probabilities [132]. If a meaningful set of priors can be obtained, then the MAP criterion can be used to obtain the solution (as explained in the discussion of the previous section).

K Nearest Neighbours (KNN): This method does not require estimating distributions. The training data are retained and a new example is compared against the K nearest neighbours to it. This is usually done by measuring the "distance" of the new example along the m dimensions of the attributes. The class assigned to the new example is typically the majority class among the K nearest neighbours in the training data [16, 85].

Both methods usually require that all attributes be continuous in order to be able to estimate distributions with smooth functions that are easy to deal with analytically, or in order to be able to measure some "distance" along one of the dimensions. The K nearest neighbour method is reincarnated in the AI/machine learning fields in generalized form called *case-based classification.* Stanfill and Waltz [118] give a good account of this approach. They provide measures that enable one to measure "distance" between values of discrete attributes.

One of the significant Bayesian classifiers that has met with some success is AUTOCLASS [10]. AUTOCLASS utilizes Bayesian classification methods to invent classes from data (conceptual clustering) and to estimate the "correct" parameters for the model used to describe the classes. If the classes are known a priori, AUTOCLASS can be used to learn classifiers. AUTOCLASS assumes a model (distribution) on the data and attempts to tune the proper parameterization for that model. If an attribute is not distributed according to, say a Gaussian distribution, then tricks can be used to make it so by applying a proper transformation to it[1]. AUTOCLASS scored a great success for machine learning approaches when it was applied to a conceptual clustering task in the domain of astronomy. AUTOCLASS invented new clusters of stars and galaxies (infrared sources) that differed significantly from that of NASA's experts. The significance of this is in the fact that the new clusters clearly reflected physical phenomena in the data. The AUTOCLASS infrared source classification became the basis of a new star catalog [10, 11].

[1]Peter Cheeseman, private communication, January 1991, NASA-AMES Research Center.

2.3 Decision Trees

As was stated in Chapter I, the focus of this dissertation is limited to investigating potential approaches to solving the learning problem within the framework of decision trees. Before describing what is involved in generating decision trees, we briefly pause to point out why we think the decision tree-based approach is a reasonable one.

2.3.1 Why the Decision Tree Approach?

As mentioned earlier, decision trees offer one approach to the concept learning problem. That, however, does not immediately make them the preferred approach. Why is the decision tree approach preferred to others such as neural networks, classification rules, and so forth? We do not have a compelling reason that rules out other approaches. As a matter of fact, the decision tree approach does impose limitations that may make many solutions unattainable. We propose that the following reasons make the study of decision trees, as an approach to the learning problem, a worthwhile endeavour:

1. A decision tree represents a set of classification rules for which the control problem is implicitly solved. This makes the use of a decision tree classifier more convenient. If one were exploring the space of classification rules directly, then the control problem becomes a significant issue to address [39].

2. The decision tree methodology is inherently efficient and is thus suitable for dealing with large data sets. By considering only a single dimension at a time, the learning problem is progressively partitioned into smaller subproblems having smaller sets of training data. This makes it more efficient at handling large data sets compared with an alternative algorithm that repeatedly examines the entire data set (c.f. learning one rule at a time, or KNN approaches).

3. The final set of rules produced is relatively easy to understand by humans since it is expressed in terms of simple high-level conditions. While this is not an

advantage over a rule learning approach, it is advantageous to neural networks, KNN, or other pattern recognition techniques.

4. Decision trees are an example of non-parametric approach: the basic decision tree generation framework we adopt does not require extra information not given in the training data. For example, we require no domain knowledge or prior knowledge of distributions on the data or classes.

These factors, taken together, make decision trees worth pursuing as a potential solution to the learning problem. Granted that each of these "advantages" comes at a price. For the first, the expressive power is restricted. For the second, the partitioning of data may also lead to problems in terms of loss of information and making decisions based on excessively small samples of data. For the third, comprehensibility may not be essential and only correctness may be. However, we take the position that it is interesting to find out how well one can do given this seemingly reasonable and promising framework. The issues of efficiency and convenience are also of great importance from an industrial applications perspective.

2.3.2 Decision Tree Generation

The formulation of a no lookahead, nonbacktracking algorithm for decision tree generation requires the specification of the following four rules [5]:

DRule 1: A rule for choosing an attribute to branch out of a node.

DRule 2: A rule for choosing a particular partition of the examples based on the selected attribute, i.e. a rule to select a condition involving the selected attribute and the set of outcomes of the test.

DRule 3: A rule to decide when to stop partitioning a node, thus deeming it a leaf node.

DRule 4: A rule for labeling a leaf node with a class.

In this dissertation, we are particularly concerned with the details of DRules 1, and 2. For DRule 3, we assume that a node continues to be partitioned until all of its examples are of one class or all of its examples have the same values for all of the attributes. In the former case, DRule 4 becomes trivial. In the latter, we assign multiple classes with confidence factors. If a single class prediction per leaf is required, we label the leaf with the majority class.

DRules 1 and 2 are essential in determining the structure of the tree, and consequently its effectiveness as a classifier. DRule 3's effect can easily be achieved by employing a good tree pruning algorithm after the tree has been generated [5, 92, 93]. In [5] it is recommended that the tree be generated without stopping early, and then pruning it upward to an "acceptable" level according to a cost measure. See [125] for a discussion of some pruning techniques and for a method that employs a different stopping rule during decision tree generation.

The basic functioning of a decision tree generation algorithm is depicted in the high-level flowchart of Figure 2.1. This flowchart serves to illustrate where each of the DRules fits within the algorithm. Note that for any given node, all continuous-valued attributes are discretized prior to formulating a test for each of the attributes. This allows us to assume that all attributes are discrete and formulate our decision tree generation algorithms accordingly. We shall pay particular attention to the continuous-valued attribute discretization process in Chapters V and VI. To generate a decision tree, a root node is initialized and the entire training set of examples is assigned as the set of examples for the node. The algorithm is then invoked passing it the new root and the training set.

One of the earliest decision tree generation systems was Hunt *et al*'s Concept Learning System (CLS) [48]. CLS generates trees that attempt to minimize the cost of classifying an object. The cost is defined in terms of two components: the cost of testing the value of an attribute, and the cost of misclassifying an object. CLS employed a computationally expensive search of all possible decision trees of some set depth. The search strategy is a lookahead strategy similar to minimax, and could require a substantial amount of computation [92].

Quinlan [91, 92] modified the CLS algorithm to make it more efficient by avoiding the expensive lookahead search. He employed a heuristic, hill climbing, nonback-

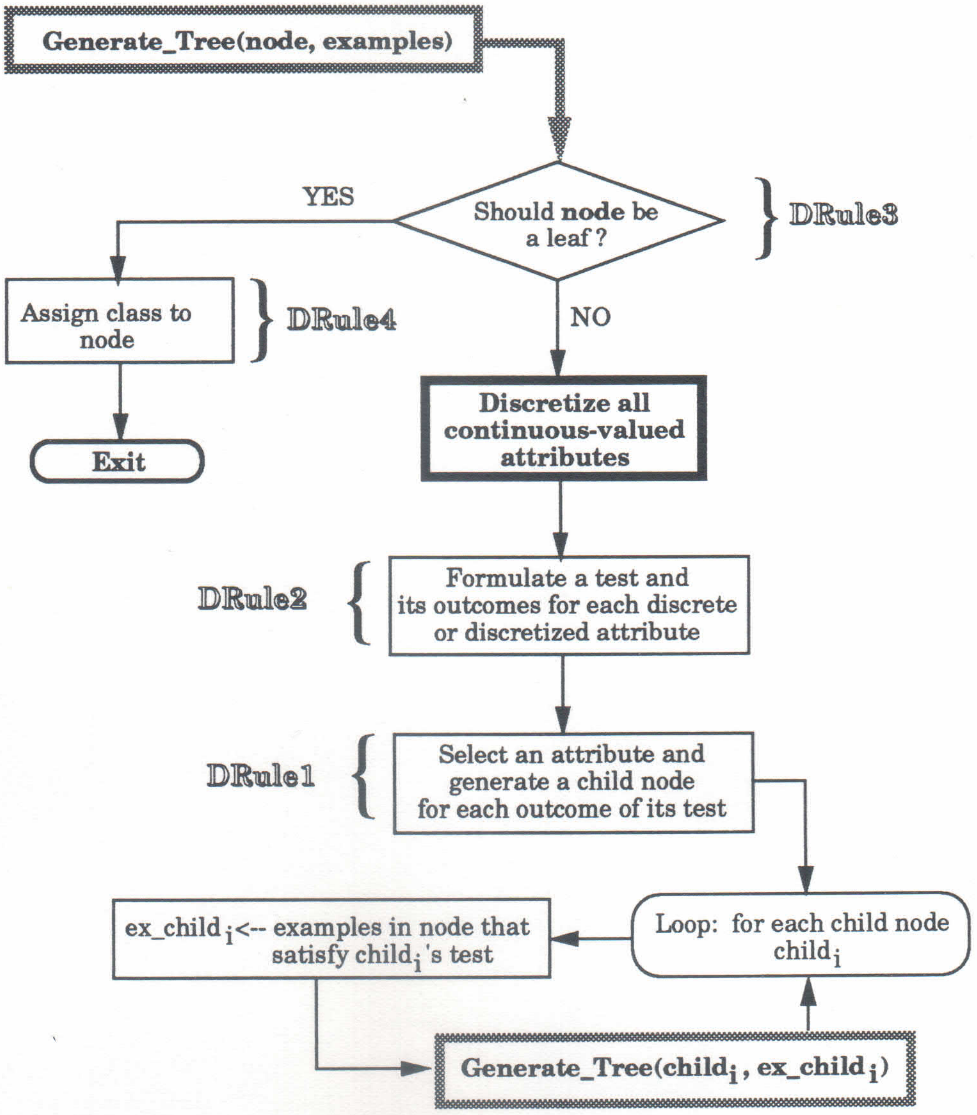
Generate_Tree(node, examples)
YES
Should node be a leaf ?
DRule3
Assign class to node
DRule4
NO
Exit
Discretize all continuous-valued attributes
DRule2
Formulate a test and its outcomes for each discrete or discretized attribute
DRule1
Select an attribute and generate a child node for each outcome of its test
ex_child$_i$<-- examples in node that satisfy child$_i$'s test
Loop: for each child node child$_i$
Generate_Tree(child$_i$, ex_child$_i$)

Figure 2.1: Flowchart of a Skeleton Algorithm for Decision Tree Generation.

tracking algorithm for generating the decision tree. The goodness of the generated tree depends entirely on the attribute selection measure employed. Quinlan used an information entropy minimization heuristic to select attributes. This heuristic appeared to work very well, and resulted in relatively compact trees. We discuss the details of this algorithm, named ID3 (Interactive Dichotomizer 3), in the next chapter. ID3 is now a popular, commercially available, program that is being used in many applications. As such, ID3 will serve as the basic decision tree generation algorithm that we will try to improve.

The use of the information entropy heuristic appeared earlier in the pattern recognition and information theory literature [5, 111, 131]. One of Quinlan's [90, 91, 92] important contributions was to introduce it to the AI field and successfully apply it to real-world learning problems. A more general class of *impurity measures*, of which the entropy heuristic is an instance, was used for decision tree generation by Breiman *et al* [5] (see Chapter VII for further details). Breiman et al claimed that interchanging any of the measures in this class of measures did not significantly affect the quality of the resulting decision tree. Their emphasis, however, was not on the tree growing step. Rather, they concentrated on schemes for pruning the generated tree effectively. Any tree pruning will, in general, cause the tree to be no longer consistent[2] with the training set.

Although loss of consistency may be acceptable if the overall error rate is reduced, pruning is a last resort solution from the tree generation perspective. In other words, if one had a "better" algorithm for decision tree growing, then one could always prune the generated tree later to further improve it. In this dissertation, we basically adopt the approach that one should attempt to generate a "good" tree in the first place. We are thus on a quest for a better decision tree growing algorithm. DRules 1 and 2 are therefore of great importance to us. Another question to be answered, is what do we mean by an algorithm generating "better" decision trees? We address this question in Chapter III.

As mentioned above, the ID3 algorithm employs the information entropy mini-

[2]A tree is said to be *consistent* with the training set if it classifies the training set without error.

mization heuristic to achieve DRule 1. The heuristic is discussed in detail in chapter IV. For DRule2, ID3 simply creates a separate branch for each individual value of the selected attribute . We discuss the consequences of this in Chapter IV. To make the description so far more concrete, we give an example of a decision tree generated by the ID3 algorithm.

Example 2.3.1

The decision tree depicted in Figure 1.1 is the tree that ID3 generates for the example set given in Example 1.2.1. Note that a branch is created for every value of the selected attribute. The conditions on the branches are therefore simple attribute-value pairs of the form (attribute = value). The decision tree corresponds to a set of five zero-order classification rules.

2.3.3 Decision Trees Versus Classification Rules

As we mentioned earlier, one of the advantages of using the decision tree framework for representing sets of rules is that the control problem is solved. Namely, when using the rules to classify examples, at most one rule will match any single example. The matching stage becomes very efficient since only the conditions of the matching rule will be tested. In addition, because they strictly form a partition on the example space, with each rule being a block of the partition, rules are easy to understand: they never overlap and they can be considered in isolation. All this convenience comes at a price, however.

Although every decision tree corresponds to a set of rules, the converse is not true. A set of rules may not be representable by a single decision tree. This can be illustrated by the following simple example.

Example 2.3.2

Consider the following two rules for the two classes C1 and C2:

IF (A=a) Then Class is C1

IF (B=b) Then Class is C2

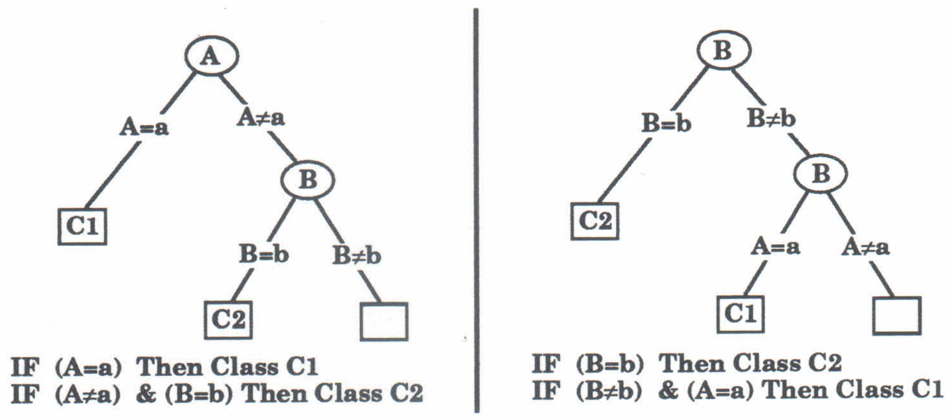

Figure 2.2: Two Approximating Decision Trees.

These two rules are not representable in a single decision tree. The closest one could come to this is by the approximating trees of Figure 2.2. Note that for either tree, one of the two rules will have an excessive condition involving another attribute. Specifically, a rule set that comes from one tree must share at least one test: the test at the root node.

What are the implications of this? The space of trees is more constrained than the space of rules for the same learning task. Both spaces are large and an exhaustive search approach is intractable. Finding optimal decision trees (with respect to size) is also NP-hard. It is not clear whether the limitation arising from the fact that the space of trees is smaller is a serious one. Since we do not yet have the final word on decision trees, we take the position that decision trees merit further study. We will live with this limitation, see how well we can do with decision trees, and attempt to derive "better" decision trees when possible. This will allow us to focus purely on the induction problem without having to deal with the additional control problem associated with using rules for classification.

2.3.4 Limitations of the Attribute-Value Pair Representation Scheme

We mentioned that the examples are expressed in the attribute-value pair notation. This language for expressing examples produces certain restrictions on the concepts that a learning algorithm can express. In the type of supervised learning that this thesis is concerned with, we do not pay attention to the *feature discovery problem.* Thus our learning algorithm is not intended to define (invent) new attributes in terms of existing ones. We assume that all the desired attributes have been defined *a priori.* The task of the learning algorithm is to discover which subset of attribute-value pairs is relevant to each class.

If new attributes need to be formulated, then that can always be done as a preprocessing stage. For example, it may be desirable that the learning algorithm have access to facts such as

$$A_1 > (A_2 + A_3)$$

where A_1, A_2, and A_3 are attributes in the problem. In general, the learning algorithm is not supposed to have access to "domain knowledge" that assures the meaningfulness of comparing the value of A_1 to the sum of the values of A_2 and A_3. If such knowledge is to be made available to the learning program, then this may be done by defining such a relation as a zero-order predicate, say A_0, which takes on the value TRUE for each example for which the relation

$$A_1 > (A_2 + A_3)$$

holds, and the value FALSE otherwise. We may thus encode arbitrary relations between the attributes in terms of new binary-valued discrete attributes. Hence, in principle, at the cost of increasing the storage needed for the training data, we may encode arbitrarily many relations between attributes. However, such encoding must be defined ahead of time by the users or by another algorithm that possesses the necessary knowledge.

Another limitation of the attribute-value pair notation is that it does not encode any ordering information between examples. For example, the training data may be a set of recorded time-series measurements. In this case, we may capture such ordering information either by adding new attributes or transforming the problem entirely. For example, time-dependent trends can be encoded in new attributes that measure entities such as: time averages, slopes, local minima and maxima, and other properties derived from multiple ordered examples. Thus, in principle, given sufficient preprocessing the "simple" learning algorithms that deal strictly with attribute-value pair notation can be made very useful.

In general, without the proper preprocessing of the data to encode inter- and intra-example relations in terms of new attributes, an attribute-value pair representation scheme cannot represent such information. If such knowledge is essential to the classification task then without the proper preprocessing, the task becomes very difficult, probably impossible, for the type of learning algorithm we are interested in.

2.4 Perceptrons and Neural Networks

Perceptrons and Neural Networks represent another type of classifier. Such classifiers may be considered as strictly "black box" classifiers. Statistical approaches are typically model-based: they attempt to model the data distribution and then make an optimal decision based on the fit. KNN-based approaches call for retaining the data and comparing new examples with the "nearest" K training examples. Decision tree/symbolic rule classifier approaches produce classifiers in a symbolic logical format that is intended to be meaningful to humans. However, perceptrons and neural networks are essentially "curve fitting" techniques where the target language in which the fit (classifier) is expressed is intended to be strictly an internal representation for which no "understandable" interpretation need be found. We discuss this issue of "comprehensibility" in Section 2.4.2 below.

A perceptron is a linear thresholding unit. It computes a linear combination of its inputs and compares the result with a fixed threshold. The input values are multiplied by their respective weights and then summed. The perceptron "learns" by adjusting the weights based on a learning algorithm that compares the perceptron's output with the desired output for each training example. The process of classifying training examples and then adjusting weights is iterated until the perceptron classifies the training set correctly.

Minsky and Papert [78], in a critique of Rosenblatt's [100] claims that perceptrons are powerful learning systems and that they hold great promise, showed that a single layer perceptron network is capable of learning only a very restricted class of concepts. This caused a widespread loss of interest in perceptrons without regard to the fact that the limitations did not necessarily apply to multi-layered networks. The perceptron made its comeback in the generalized form of neural networks [66] after a two-decade long hiatus. The units in such networks are generalized perceptron units: they have a nonlinear thresholding function (a sigmoid function) into which the linear weighted sums are fed. In this form, and with at least one inner layer of units, neural networks have a dramatically enhanced representational power. Many learning algorithms exist for training these nets. The most popular is based on propagating backward the error in the network's prediction to the inner layer units and adjusting the weights to minimize this error [66]. It has been shown that if the weight adjustments are "small enough," the back propagation learning algorithm will eventually converge on a stable set of weights. This solution, however, is not necessarily optimal (in terms of minimizing the expected error of the network). In addition, if the weight adjustments are small, the network typically requires a huge number of iterations through the training set to converge.

The main attraction of perceptron/neural network type approaches is due to two factors: simplicity of hardware implementation and potential physiological "similarity" to human brain mechanisms. The simplicity of the hardware implementation comes from the hope that each unit in the network can be implemented with very simple, inexpensive, uniform circuits. The physiological plausibility facet of the attraction was summarized well by Minsky and Papert [78]

> *The popularity of the perceptron as a model for an intelligent, general purpose learning algorithm has its roots, we think, in an image of the brain itself as a rather loosely organized, randomly interconnected network of relatively simple devices.* *(p. 18)*

2.4.1 Problems with the Neural Network Approach

An advantage of a decision tree based approach over a neural network or Bayesian classification based approach is in the interpretability or comprehensibility of the