CHAPTER I

INTRODUCTION

"Where shall I begin, please your Majesty?", he asked.
"Begin at the beginning" the King said gravely,
"and go until you come to the end: then stop."

C.L. Dodgson, 1865
(alias Lewis Carrol)
Alice's Adventures in Wonderland, Ch. 12.

What is this dissertation about? The short answer is that it is about the quest for "better decision trees." The long answer requires some context and quite a bit of elaboration. This work falls in the area of machine learning, a subfield of artificial intelligence (AI). We shall not concern ourselves with what machine learning exactly means since that question does not have a clear answer. Among the many attempts to address this issue is one by Simon [113]. In his attempt to define the process of learning, Simon states that

> *...learning denotes changes in the system that are adaptive in the sense that they enable the system to do the same task or tasks drawn from the same population more efficiently and more effectively the next time.* *(p. 28)*

Scott [107] points out that this is only a *functional* definition in the sense that it does not define learning by how learning takes place, rather, it defines learning by what learning achieves. Scott characterizes learning as

> *...the organization of experience... Learning is any process through which a system acquires synthetic a posteriori knowledge.* *(p. 360)*

Any definition of a general purpose learning algorithm is bound to have problems since our notion of "general purpose" is roughly equivalent to "human-like." As fascinating as the latter may be, it is not (yet) a well-understood entity.

Since this dissertation does not aim to discuss "general purpose" learning machines, we adopt a simple definition that describes a narrower class of "learning" machines. For the modest purposes of this dissertation, the following statement captures the behaviour of what we would describe as a *learning machine*:

> *A learning machine is a machine that is capable of improving its performance on a specific task over a finitely bounded period of time as it interacts with its environment.*

Thus, the "learning," whatever it is, is in the eye of the beholder. Note that we are narrowing down the definition by fixing the task of the learning machine and bounding the time it takes it to learn. For now, we leave open what "interaction with the environment" means. We shall quantify the notion of "improvement in performance," and restrict the time scale needed to achieve the improvement later in the chapter. Although lacking in preciseness, the definition is, nevertheless, sufficient to convey what we mean in an intuitive, informal sense. All the informal notions will be made precise as we define our learning task exactly. Whatever changes take place inside the learning machine that enable it to "improve" its performance are referred to as "knowledge about the task within the context of the environment." In this sense, learning is basically the acquisition of "knowledge."

There are two fundamental reasons for studying the process of learning. The first targets understanding the process of learning as manifested in humans. This relates to the age-old question regarding the nature of human knowledge: what is knowledge, and how is it acquired? The question is deeply rooted in philosophy, starting with Plato, and is a major theme of today's psychology. As part of their quest to gain insight and understanding into human learning, some researchers attempt to endow a machine with the ability to learn. This places special constraints on the approaches they may adopt since the learning mechanism they adopt must be physically and psychologically plausible in human beings. For example, we do not generally expect

that a human being stores a detailed database of all previous experience and performs a fast search for nearest matches each time a decision needs to be made. The data are simply too voluminous to be stored completely in "raw" format.

Another reason for studying learning is derived strictly from a mechanization perspective—the quest for endowing machines with the ability to make intelligent decisions autonomously. One goal of machine learning, then, is to enable computers to modify their own behaviour and acquire usable and useful knowledge autonomously as an alternative to being strictly programmed by humans. Usable knowledge refers to knowledge that is not too costly to store in terms of space, and not too costly to retrieve in terms of time. Useful knowledge is knowledge that leads to improvement in performance. In this case, the feasibility of the learning mechanism as a candidate human learning mechanism is not an issue. The goal is to enable a machine to learn since learning is considered an important aspect of intelligence. The quest is for a machine that can be considered "intelligent" by human observers (who are presumed to be "intelligent" themselves). In this case, the important factor is *computational feasibility*: human learning is of interest only to the extent that it may give insights into what is computationally feasible.

The approach taken in this dissertation falls under the latter category. Given a learning task that is performed by human beings, and that appears to be an important aspect of intelligence, can we devise an algorithm to perform that task on a computer? The *computational feasibility* issue adds one further restriction on the algorithm: it must have polynomial complexity. Our focus, then, is to precisely define a task that is generally considered "intelligent," to show that it can be performed by an algorithm of polynomial complexity, and to investigate possible ways to improve our algorithm's performance. The task we are concerned with is an instance of *induction*, generally referred to as *learning from examples*. Essentially, the goal is to discover compact patterns in large training data sets in order to summarize the latter in a useful form—a structure or a classification scheme. This structure, then, represents knowledge about the data. As such, the work described in this dissertation falls under the category of automated knowledge acquisition. We are concerned

with learning a scheme to classify future examples, a classifier, from training data consisting of previously classified examples.

In particular, this dissertation is concerned with studying the problem of inducing classifiers in the form of decision trees. The results derived in the later chapters are primarily intended to provide means for improving important aspects of top-down, nonbacktracking, decision tree generation algorithms. Some of the formal results derived are generally applicable to classification algorithms, others formally answer questions about decision tree classifiers; e.g., when can we claim that one decision tree is better than another?

The main importance of this venue of work is its potential for mechanizing "knowledge" extraction from data. Rather than relying on manually encoding knowledge in a program or an expert system, the machine learning approach attempts to circumvent the difficulties (to be elaborated in the next section) of the manual encoding approach by extracting classification rules from data directly. The training data consist of descriptions of certain situations or states along with a record of the (intelligent) action taken by a human expert in that situation. In general, the action or outcome associated with a situation can be any event that we are interested in predicting. The idea is to attempt to capture the general and appropriate conditions under which certain outcomes should occur. Rather than require that a domain expert provide domain knowledge, the learning algorithm attempts to discover, or induce, rules that emulate expert decisions in different circumstances by observing examples of expert tasks.

Let us begin by motivating the general topic addressed by this dissertation. Readers who are familiar with all the reasons that make a machine learning approach important and sometimes essential may proceed to Section 1.2 where we define the basic notions: learning problem, multiple concept learning, training data, decision trees, etc. Those who wish to bypass the basic definitions may proceed to the statement of research goals in Section 1.4.

1.1 Motivation

One of the goals of AI research is to provide mechanisms for emulating human decision-making and problem solving capabilities, using computer programs. The first "popular" AI attempts at such systems appeared as part of the technology known as "expert systems." Expert systems are intended to provide the means of encoding human knowledge about a specific task in terms of situation-action rules. The idea is that if such systems are endowed with sufficient knowledge of the task at hand, they may be able to emulate human expert behaviour in most, if not all, situations that arise during task execution. The traditional approach to constructing expert systems relies on interviewing domain experts. Since domain experts are not expected to be programmers, a programmer (usually called a *knowledge engineer*), sometimes with the aid of individuals that are skillful at conducting interviews, essentially asks the domain expert to describe how and why expert decisions are made under various circumstances. The knowledge engineer is supposed to encode the information obtained from interviewing in the form of executable programs of if-then rules. We refer to this process as: *manual knowledge acquisition.*

Even with such a constrained and narrow goal, serious difficulties arose that hindered the development of successful expert system applications [76]. Prominent among these difficulties is one referred to as the "knowledge acquisition bottleneck" [28]. Human experts find it difficult to express their knowledge, or explain their actions, in terms of concise situation-action rules. If pressed to do so (sometimes under intense interrogation), they often produce rules that are incorrect, or that have many exceptions. The articulation of specific intuitive knowledge into deterministic rules is a difficult, sometimes unrealistic, problem for human experts. Interviewing domain experts to extract such knowledge is also an expensive process demanding time from experts and knowledge engineers. In addition, it is a difficult and often frustrating process for the domain experts involved. Industrial diagnostic expert systems typically require a long development time. Though the extraction of knowledge from domain experts was initially identified as a bottleneck in the process of developing an

expert system, it is currently believed to be a "major hurdle" as well as an "onerous process" by most [17].

A second problem arises in a different situation: What if a task is not well-understood, even by the experts in that area? An example of this situation is manifested in our experience with the automation of the reactive ion etching (RIE) process in semiconductor manufacturing [51, 52]. In such domains, abundant data are available from the experiments conducted, or items produced. However, models that relate how output variables are affected by changes in the controlling variables are not available. Experts strongly rely on familiarity with the data and on intuitive knowledge of such a domain. It is an often accepted hypothesis in the "manual knowledge acquisition" literature that experts "work in shallow mode" [15]. That is, experts are able to make good decisions at the level of individual cases, but are unable to answer general questions about the data (i.e., provide useful rules) [15]. How would one go about constructing an expert system in such circumstances?

Two additional factors promote the case for the machine learning alternative to the manual construction of expert systems. The first is the growing number of large databases that store instances of diagnostic tasks. Such data are typically accessed by keyword or condition lookup. As the size of the database grows, the keyword lookup approach becomes ineffective. Suppose an expert needs to refer to previously diagnosed cases similar to a case currently being diagnosed. A query may easily return hundreds of matches, making it difficult for a human expert to make a decision based on the query. A method for automatically determining relevant conditions to make the "proper" query would be needed in this case. Databases containing large amounts of examples defy human analysis and induction capabilities. In addition, a high level of expertise is required to fully exploit such knowledge and infer important patterns from it. This creates an important, and as of yet unfulfilled, niche for machine learning techniques. Examples of earlier successes in this area, which also serve as motivation for us, are cited in the next section.

Another motivation is the evolution of complex systems that have an error detection capability. Telephone and computer communication networks are an example

[120]. Faults are detectable by the network hardware. Several thousand faults may occur during a day. To debug such a network, a human would need to sift through large amounts of data in search of an underlying cause. An automated capability of deriving conditions under which certain faults occur may be of great help to the engineer in discovering underlying problems in the hardware.

Both of the above situations indicate a need for a method to summarize large amounts of data effectively. Machine classification learning is a potential answer. If successful, a machine learning approach provides a potential partial solution to the problem of extracting knowledge from domain experts:

> Do not attempt to get it from experts in the first place! Avoid interviewing experts whenever possible and get whatever knowledge you can directly from the data.

Of course, the knowledge extracted from the data may be incorrect since it is produced by induction. However, rules extracted from human experts do not come with a guarantee of correctness either. Another side benefit for using machine learning is the possibility of discovering patterns not known to experts. An example of this occurred when we applied our learning algorithms to data obtained from Hughes Microelectronics Center from a reactive ion etching (RIE) process for manufacturing semiconductor wafers. The experts strongly disagreed with the derived rules. Some of the rules were making predictions that were "clearly wrong" from the process engineer's point of view. Later, it was discovered that the wafer etcher being used had a leak in one of the gas supplies, and that caused it to behave abnormally [26].

1.1.1 Earlier Successes

To illustrate the potential for success of inductive learning, we cite two experiments in diagnostic knowledge base construction and an experiment in database compaction. An additional successful application in astronomy by Cheeseman's AUTOCLASS system [10] is discussed later in Section 2.2.

The first experiment was conducted by Michalski *et al* [69, 71]. A database of examples of soybean diseases was used by a learning program, AQ11, for generating

diagnostic rules. At the same time, a team of knowledge engineers consulted with a human expert in constructing classification rules for the same set of examples. The program-generated rules outperformed the expert's rules in diagnosing unseen data [69].

The second experiment followed the same approach using a different program and a different domain. In this case, Quinlan's ID3 [92] algorithm for generating decision trees, which forms the starting point for the work described in this dissertation, was applied to a diagnosis task in a medical hemotology domain. ID3 quickly produced a classifier that outperformed an expert system developed using the traditional knowledge acquisition approach of interviewing experts [84].

Finally, Mozetic [83] applied The NEWGEM system to a medical knowledge base consisting of over 5000 examples of ECG multiple heart disorder cases. He was able to shrink the knowledge base down to 3% of its original size, making its use in the field possible.

1.2 Learning from Examples

We focus on the problem of learning classification schemes from instances of examples with known classifications (training data). In classification learning, the goal is to induce a classifier, or classification scheme, that can be used to predict the classes to which future unseen examples belong. Classification is considered one of the intelligent activities performed by humans. It is a prerequisite of symbolic reasoning since it provides the building block for abstraction. Almost all human decision-making functions can be viewed as classification tasks of some sort. A decision is usually a choice of one of a fixed set of options or actions. The state of the environment at the time the decision was taken is said to satisfy the conditions under which the decision is considered appropriate or correct. The problem of classification is to discover these conditions (or a good approximation to them). We shall shortly provide a formal statement of the problem and the desired solution.

The state of the environment is typically encoded in terms of a predetermined

set of *attributes*. An attribute may be *continuous* (numerical) or *discrete* (nominal) valued. For example, a nominal attribute may be *shape* with values { *square, triangle, circle*}. An example of a continuous attribute is *pressure* or *temperature*. The range of such an attribute is the reals, rationals, integers, or any set on which a linear ordering can be imposed. Note that the term *continuous-valued* is used in the literature even if an attribute is integer-valued (e.g. [92]). We use the term in this overgeneral sense as well[1].

In classification, the classes are required to be discrete. If the classes are continuous, then the problem becomes a *regression problem* [5]. An example class may be an action taken during the control of a process, e.g. *raise temperature* or *lower pressure*, or some decision such as diagnosing a fault, etc.

A training example consists of a description of a situation and the action performed or decision taken in that situation. The situation is described by listing the values of all the attributes. The action associated with the situation, the *class* to which the example belongs, is a specification of one of a fixed set of pre-determined allowed actions. The class of each training example is typically determined by a human expert during normal execution of some task.

The goal of the learning algorithm is to derive conditions, expressed in terms of the given attributes, that are predictive of the classes. Such rules may then be used to classify future examples. Of course, the quality of the rules depends on the validity of the conditions chosen to predict each action.

Given a set of training examples, the training set, it would be desirable to induce a *minimal set* of *maximally general* rules that *correctly* classify the examples. The importance of generality of the rules lies in increasing their predictive power over sets of unseen examples. This has been the prevalent goal of research in machine learning and pattern recognition. The question of whether it is justified to search for compact classifiers will be dealt with in Chapter III. In fact, we show that the more "compact" a classifier is, the more likely it is that the predictions it makes are "correct."

[1] A more appropriate name would be *ordered attributes* as used in [5].

Table 1.1: A Simple Training Set of Examples.

example	selectivity	Δ line width	class
e-1	normal	normal	*power is high*
e-2	normal	high	*power is low*
e-3	high	high	*power is low*
e-4	high	low	*power is high*
e-5	low	normal	*flow rate is low*
e-6	low	high	*flow rate is low*

To illustrate the discussion so far, we include an example training set.

Example 1.2.1

The simplified small example set shown in Table 1.1 consists of six examples e-1 through e-6. There are two attributes: *selectivity* and Δ *line width*. The attributes can take the values *low, normal,* and *high*. There are three classes: *flow rate is low, power is low,* and *power is high*. Note that this is only an illustrative simplification. Typically, the number of examples of a meaningful training set is at least in the hundreds, while the number of attributes is usually in the tens.

A classification rule for predicting a class consists of a specification of the values of one or more attributes on the left hand side and that class on the right hand side. A simple rule consistent with the training set of Example 1.2.1 may be:

IF (selectivity = low) THEN *flow rate is low*

1.2.1 Multiple Concept Learning

Concept learning refers to the mechanized induction of a classification scheme from a set of training examples. We refer to it as "induction" because the generated classifier is expected to capture generalized patterns from the data. Therefore, it should be capable of classifying future examples that have not appeared in the training data. Historically, the term concept learning has been used to describe learning a single concept [20, 45, 91, 128, 133]. In this case, the training data have only two classes. An example is either *positive*, meaning it is an example of the concept to be

learned, or it is not. A large body of the literature in machine learning and pattern recognition deals with the binary class problem. *Multiple concept learning* refers to concept learning when there are two or more classes in the data. We are concerned with the general problem where the number of classes is arbitrary (but of course finite).

Let A be the set of attributes. Assume there are m attributes; so $A = \{A_1, \ldots, A_m\}$. We denote the range of an attribute $A_j \in A$ by $Range(A_j)$. If A_j is discrete, then

$$Range(A_j) = \{a_{j1}, a_{j2}, \ldots, a_{jr_j}\}$$

and if A_j is continuous-valued, then, in general,

$$Range(A_j) \subseteq \Re.$$

Let C be the set of k classes, $C = \{C_1, \ldots, C_k\}$. An *example* is an m-tuple belonging to the space

$$\overset{m}{\underset{i=1}{\times}} Range(A_i)$$

where $\times$ denotes the cross product operation on sets. We denote the set of all possible examples by E. A *training example* is an example with a class label. Thus, a training example is an $m+1$-tuple of the form

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

where each $v_i \in Range(A_i)$ is one of the values of the attribute A_i, and $C_j \in C$ is one of the k classes. In general, we use the term "example" to refer to examples or training examples. It is usually clear from the context which we mean. Also note that the class is simply one of the discrete attributes that has been distinguished a priori.

Each example is assumed to be a member of exactly one of the k classes. The class to which an example e belongs is denoted by $Class(e)$. Thus, another way of referring to a training example is by the notation $\langle e; Class(e) \rangle$, where e is an example (m-tuple). Given a set of N training examples $TE = \{e_1, e_2, \ldots, e_N\}$, it may be partitioned into the sets:

$$E_i = \{e \in E | Class(e) = C_i\} \qquad i = 1, \ldots, k$$

such that $E_1 \cup E_2 \cup \ldots \cup E_k = TE$, and for all i, j such that $i \neq j$

$$E_i \cap E_j = \emptyset$$

Definition 1.2.1 A *classifier* CL is a function that assigns a class to an example:

$$CL : E \longrightarrow C.$$

Hence, a classifier takes an m-tuple of attribute values, i.e., an (unclassified) example (or the first m components of a classified example), and assigns a class to it. Such classifiers are also known as *absolute classifiers* since they predict with absolute certainty as opposed to probabilistic classifiers:

Definition 1.2.2 A *Probabilistic Classifier* PCL is a function that predicts the classification of examples: $PCL : E \longrightarrow p$, where $p \subseteq [0,1]^k$

$$p = \{\langle p_1, p_2, \ldots, p_k \rangle | \sum_{i=1}^{k} p_i = 1.0\}$$

The k-tuple $\langle p_1, p_2, \ldots, p_k \rangle$, called the *prediction vector* of PCL for a given example, is interpreted as the confidence vector for the k classes. Thus $p_i \in [0,1]$ represents the confidence of the classifier that the example belongs to class $C_i \in C$. The class of absolute classifiers is a subclass of the class of probabilistic classifiers. An absolute classifier is a probabilistic classifier that for any example, the prediction vector output is such that for some $i, 1 \leq i \leq k$, $p_i = 1$.

If a classifier is probabilistic, then typically a scheme is employed to extract a single absolute classification from each prediction made. The simplest method is to take the class with the highest confidence as the class predicted. Thus a probabilistic classifier PCL along with a decision strategy that maps the prediction vector to a single class prediction are equivalent to an absolute classifier, CL. Since the goal of a classifier is to make a decision regarding the class of an example, we consider the decision strategy employed to extract an absolute classification from a probabilistic one to be part of the classifier itself. Henceforth, we shall use the term *classifier*, or

CL, to denote an absolute classifier or a probabilistic classifier coupled with some decision strategy.

The (absolute) classifier CL may take any desirable form: a set of diagnosis rules, a decision tree, a classification hierarchy or graph, etc. If for an example $e \in E$, the classifier CL classifies e in some class $CL(e)$, then we say that the classification is correct iff $CL(e) = Class(e)$. If the classification of e is not correct, we say that CL has *misclassified* example e.

Note that the function $Class$ defined above is an example of a classifier. $Class$ is a special classifier that always returns the "true" class of any example. This means that we are implicitly assuming that examples in the training set are *noise free*, i.e., the class label is the correct label given the attribute values (see definition of training set above).

1.2.2 The Multiple Concept Learning Problem

We are now ready to define exactly what we mean by a learning problem. We focus on two particular types of learning problems.

Learning Problem PE : Given the *learning task* $\langle A, C, TE \rangle$, where A is a set of attributes, C a set of classes, and TE a set of training examples drawn according to some fixed probability distribution on E, *find* a classifier CL such that for any $e \in E$, drawn according to the same probability distribution,

$$\text{Prob}\{CL(e) \neq Class(e)\}$$

is minimized.

This type of learning problem specifies that the goal is to find the classifier that minimizes the probability of classification error. If the joint probability distribution of E and C is completely known, then the theoretical solution to the problem is known. The solution has been known for a couple of centuries now and was originally derived by Thomas Bayes in the 1700's [2]. However, in real life, the distribution on the examples is typically unknown. In this case, no known method exists for finding

the "optimal" classifier. We discuss the PE learning problem further in Chapter II. A relaxed version of this learning problem was formulated by Valiant [126] and is known as Probably Approximately Correct (PAC) learning problem. We shall discuss the PAC learning problem further in Chapter II.

Learning Problem CX : Given the *learning task* $\langle A, C, TE \rangle$, where A is a set of attributes, C a set of classes, and TE a set of training examples drawn according to some fixed probability distribution on E, *find* a classifier CL such that for all $e \in TE$,

$$CL(e) = Class(e),$$

and such that the *size complexity* of CL is minimized.

The size complexity is measured by some syntactic measure of the size of the representation of CL. For example, how many bits are required to encode CL.

In the CX learning problem, the goal is to find a minimal size classifier that correctly classifies the training set. Finding this minimal classifier can be shown to be equivalent to the multiple-valued logic minimization problem—an NP-hard problem [36]. Thus, we know that the existence of a polynomial algorithm for producing the optimal classifier is unlikely. If we take the more courageous position of assuming that P$\neq$NP, then there exists no polynomial algorithm for solving the CX learning problem.

Given this rather difficult situation, we are faced with important questions to answer: How does one go about producing a classifier from training data? How does one go about improving an existing method for inducing classifiers? This dissertation attempts to provide answers to these questions for the special case when the classifier takes the form of a decision tree. We discuss decision trees in Section 2.3.

1.2.3 Zero-Order Classification Rules

To demonstrate the difficulty of the learning problem, let us examine the possibility of finding a classification scheme using a very restricted class of classification

rules. We would like to find a set of rules, where each rule predicts one class. The preconditions of the rules consist of a single conjunction. Each conjunct tests whether an attribute has a particular value[2]. Thus each conjunct is of the form

$$(A_i = v_j), \quad \text{where} \quad v_j \in Range(A_i).$$

Such a conjunct is often referred to as an *attribute-value pair*. Since the conditions of these rules do not contain quantifiers, we say they are zero-order logic formulas.

One extreme approach that can be adopted to generate a set of classification rules of this type is to construct a rule for each example. Thus the example e,

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

would cause the following rule to be created:

IF $(A_1 = v_1) \wedge (A_2 = v_2) \wedge \ldots \wedge (A_m = v_m)$
THEN class is C_j.

This approach is tantamount to *rote learning*, or pure memorization, and results in the generation of a large number of very specific rules that have very low predictive value for examples not included in the training set TE. The reason for this is that many attribute-value pairs may actually be irrelevant to the determination of an example's class. Their inclusion in the preconditions serves only to limit the predictive power of the rule. The problem is to discover which attribute-value pairs are irrelevant.

On the other extreme, one could generate every possible combination of attribute-value pairs, somehow evaluate the ability of each combination to predict each of the classes, and finally pick a minimal cover set of these to form the rules. This actually constitutes the ideal target for a learning program since nothing can do better, assuming that a high degree of induction is acceptable. How many such

[2]For the purposes of this discussion, assume that all attributes are discrete.

formulas are there? The formulas are conjuncts of attribute-value pairs, so each is of the form

$$\bigwedge_{i \in I} (A_i = Vi_{s_i})$$

for all possible I and s_i where $I \subseteq \{1, 2, \ldots, m\}$, Vi_{s_i} is the s_i-th value in the range of A_i, and for each i, s_i can be any value in $\{1, 2, \ldots, |Range(A_i)|\}$. To get an idea of how many such formulas there are, we make the following assumption: Each of the m attributes can take on the average one of r possible values. Assuming that

$$\forall i \;\; |Range(A_i)| = r$$

makes it convenient to represent the space of all possible formulas by an m-digit number expressed in base $r + 1$. Thus each attribute is assigned a digit which can take up r values plus the value 'blank' to indicate that the corresponding attribute does not appear in the formula. Excluding the all 'blank' formula, it is obvious that there are $(r + 1)^m - 1$ possible formulas. A formula would form the left-hand side of a rule while its right-hand side will consist of a class. Thus, we have to check the validity of

$$k \cdot [(r + 1)^m - 1]$$

formulas against N examples (recall that k is the number of classes). For example, if we have 10 attributes with 5 values each, and 10 classes, we get over 600 million possible rules. These should then be checked against the N training examples for consistency, yielding a potentially large subset of consistent rules that cover the examples. The problem of finding a minimal cover of the N examples can be mapped in a straightforward manner to a multiple-valued logic minimization problem. It is therefore NP-complete [42].

This discussion clearly illustrates that an exhaustive search approach to finding a set of classification rules is computationally infeasible. Further problems exist with using rules. One has to insure that the rules do not make conflicting predictions and so forth. For this, and for the ease and efficiency of classification using decision trees, we turn our attention to the generation of classifiers in the form of decision trees. In

Section 2.3.1 we provide more detailed motivation for our choice of the decision tree framework.

1.3 Inducing Decision Trees

Given a learning task $\langle A, C, TE \rangle$, a decision tree classifier is a tree structure having the following properties:

- Each nonterminal node is labelled with a test involving one of the attributes in A. The test has a finite number of disjoint outcomes.
- Each outgoing branch from a nonterminal node corresponds to one of the outcomes of the test at the node.
- Each node in the tree has a prediction k-vector associated with it.
- Each terminal (leaf) node is labelled with one of the k classes.

From this characterization of a decision tree, it should be obvious how the tree is used to classify examples. Starting at the root, the attribute test is applied to the example, and the branch corresponding to the resulting outcome is followed leading us to one of the child nodes. The process is repeated until a leaf node is reached. The tree then predicts that the example's class is the class used to label the leaf. If at any stage the outcome of a test does not have a corresponding branch, one of the following two options is chosen:

1. Stop attempting to classify the example and claim that the tree fails to classify the example.

2. Guess the example's class based on the outcome prediction vector at the node.

The outcome prediction vector at a node is derived from the subset of training examples that satisfy all the conditions appearing on the path from the root to that node. We refer to this subset as the *examples of the node*. The prediction (class) vector represents the relative frequency of appearance of the respective classes in the node's examples.

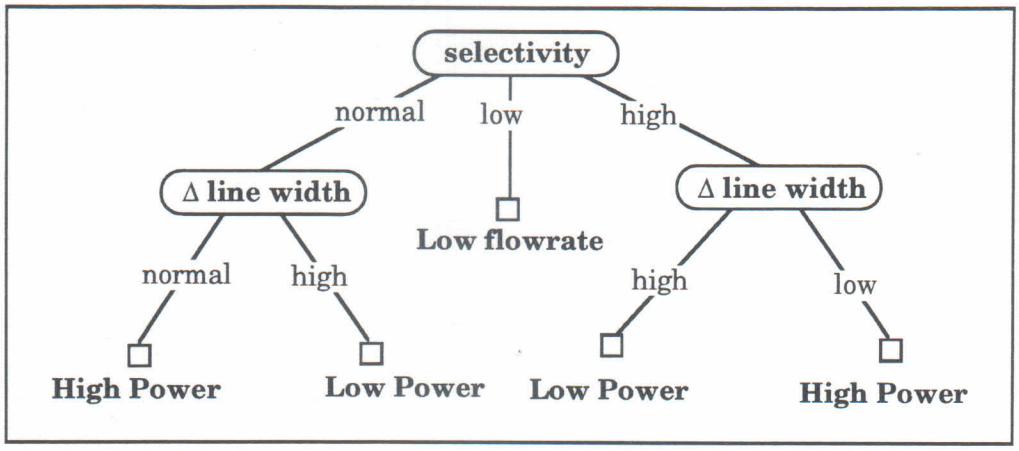

Figure 1.1: A Sample Decision Tree

Example 1.3.1

Figure 1.1 shows a decision tree for the simplified example set of Table 1.1. Note that this tree classifies all examples in the training set correctly.

Each leaf node in the decision tree corresponds to a classification rule. The preconditions of the rule consist of the conjunction of conditions on the branches from the root to that leaf. The action part of the rule is the class prediction of the leaf. Thus a decision tree represents a set of rules. This set of rules possesses the property that no more than one rule can ever match any given example.

We have so far introduced decision trees but we have not discussed *how* a decision tree is generated for a given training set. Many questions should be springing to mind at this point: How is an attribute selected at a node? What is the effect of selecting one attribute over another? How do we handle continuous-valued attributes? What is the effect of choice of training set on the resulting tree? We shall formally pose and answer some of these questions in the following chapters.

1.4 Research Goals

Between the two extremes of rote learning and optimal rule set determination, a wide range of heuristic methods exist for inducing classifiers from training data. The focus of this dissertation is to study the problem of inducing decision tree classifiers

with the goal of producing "better" decision tree generation algorithms. Specific questions that have to be answered in order to achieve the sought improvement are:

1. **What do we mean by one decision tree being "better" than another?** This requires that we formulate our performance measures for decision trees, and show that it is meaningful to claim that one tree is "better" than another.

2. **How do we formulate an algorithm that produces better decision trees?** The answer to this question is dependent on first understanding what makes a decision tree better and then designing the proper heuristic(s) used in searching for a good decision tree. In the decision tree paradigm the issues to be addressed concern decisions localized at a given node: selecting an attribute, determining test outcomes corresponding to outgoing branches, and extending the language used in representing test outcomes.

3. **How do we handle continuous-valued attributes?** In order to obtain a "symbolic" classifier, continuous-valued attributes need to be discretized so that a logical condition may be obtained for use in the decision tree.

4. **How do we verify that the claimed improvement has been attained?** This is done by empirically testing the new algorithms on many data sets from several synthetic and real-world domains and measuring their performance. This gives us the means to establish improvement over other existing algorithms.

The next section gives a brief outline of how the research goals were achieved and states where the detailed account of the respective results may be found in this dissertation.

1.5 Organization of this Document

Now that we have introduced the problem, provided enough motivation to establish its importance, and presented the research goals, we are ready to delve into the traditional detailed account of how the research goals were satisfied.

Chapter II includes a discussion of the basic classification learning problem (learning problem PE) along with the background covering basic approaches to solving the learning problem both in the classification and machine learning literature. The focus of Chapter II is to set the context for the decision tree approach to solving the learning problem. A more detailed literature survey of relevant work on concept learning in the machine learning literature is provided at the end of this dissertation in Chapter IX. The latter chapter is intended to give a brief review of other work in machine learning that generally constitutes extensions, modifications, or alternatives to the basic approach addressed in this dissertation. Therefore, Chapter IX is by no means necessary reading and is included for readers who are interested in a general description of other available approaches.

Chapter III deals with issues of performance evaluation for decision trees. We define several performance measures and show what we mean by one decision tree being "better" than another. We show that the number of leaves in a tree is the most important of the performance measure. Notably, we show that reducing a tree with a smaller number of leaves is expected to have a lower error rate. The results derived in this chapter serve as guiding principles for work in later chapters where we discuss various means for improving decision tree generation.

In Chapter IV we first discuss the details of Quinlan's ID3 [92] algorithm for decision tree generation which plays the role of the starting point for us. We introduce some of the problems inherent in the ID3 algorithm, and we present a simple generalization of the algorithm that overcomes some of the problems. We develop an algorithm, GID3, which has a user-specified parameter (TL) that controls its branching at a node. We conduct simple experiments whose purpose is to verify that, indeed, for some "proper" TL setting, GID3 overcomes some of the problems of ID3. We then introduce the algorithm GID3*, which is not dependent on any user-specified parameter. We show via empirical testing that GID3* generates a tree that is superior to any member of the family of trees generated by GID3 for various settings of its TL parameter. The empirical results also serve to support the formal results as well as the informal claims presented in Chapter III.

Chapters V and VI deal with the discretization of continuous-valued attributes. We derive formal results that serve to support our method for the binary discretization of continuous-valued attributes in Chapter V. The results also serve to improve the algorithm's efficiency. In Chapter VI, we extend the algorithm to extract multiple intervals rather than just two. We provide a formal argument to justify our extension, and we show empirically that the new capability indeed results in improved decision trees.

Chapter VII deals with the selection measure used by the tree generation algorithm. We study the selection measures currently used in the decision tree generation literature. Typical systems use evaluation functions that are *class impurity measures.* The selection measures used by ID3, GID3, and GID3*, for example, are based on a local information entropy minimization heuristic. We argue that the family of impurity measures is not particularly appropriate for use in classification problems. We define a family of measures, C-SEP, that we believe is better suited for attribute selection in the context of top-down decision tree generation. We formulate a new measure based on these arguments and implement it in the algorithm O-BTREE. We then empirically verify that the new measure is indeed more appropriate for use in classification problems.

We finally summarize the results, try to place matters in perspective, and outline future research directions in Chapter VIII.