WRAPPERS FOR PERFORMANCE ENHANCEMENT AND OBLIVIOUS DECISION GRAPHS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

By
Ron Kohavi
September 1995

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Yoav Shoham
(Principal Adviser)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Jerry Friedman
(Statistics)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

Nils Nilsson

Approved for the University Committee on Graduate Studies:

Abstract

In this doctoral dissertation, we study three basic problems in machine learning and two new hypothesis spaces with corresponding learning algorithms. The problems we investigate are: accuracy estimation, feature subset selection, and parameter tuning. The latter two problems are related and are studied under the wrapper approach. The hypothesis spaces we investigate are: decision tables with a default majority rule (DTMs) and oblivious read-once decision graphs (OODGs).

For accuracy estimation, we investigate cross-validation and the .632 bootstrap. We show examples where they fail and conduct a large scale study comparing them. We conclude that repeated runs of five-fold cross-validation give a good tradeoff between bias and variance for the problem of model selection used in later chapters.

We define the *wrapper approach* and use it for feature subset selection and parameter tuning. We relate definitions of feature relevancy to the set of optimal features, which is defined with respect to both a concept and an induction algorithm. The wrapper approach requires a search space, operators, a search engine, and an evaluation function. We investigate all of them in detail and introduce *compound operators* for feature subset selection. Finally, we abstract the search problem into search with probabilistic estimates.

We introduce decision tables with a default majority rule (DTMs) to test the conjecture that feature subset selection is a very powerful bias. The accuracy of induced DTMs is surprisingly powerful, and we concluded that this bias is extremely important for many real-world datasets. We show that the resulting decision tables are *very* small and can be succinctly displayed.

We study properties of oblivious read-once decision graphs (OODGs) and show that they do not suffer from some inherent limitations of decision trees. We describe a a general framework for constructing OODGs bottom-up and specialize it using the wrapper approach. We show that the graphs produced are use less features than C4.5, the state-of-the-art decision tree induction algorithm, and are usually easier for humans to comprehend.

To My Parents

Acknowledgments

> Nobody *can carry anyone else on his shoulders to the final goal. At most, with love and compassion one can say, "Well, this is the path, and this is how I have walked on it. You also work, you also walk, and you will reach the final goal." But each person has to walk himself, has to take every step on the path himself.*
> *—S. N. Goenka, The Art of Living*

Marie desJardins, in her guide to graduate students and advisors (1994), wrote:

> A good advisor will serve as a mentor as well as a source of technical assistance. A mentor should provide, or help you find, the resources you need (financial, equipment, and psychological support); introduce you and promote your work to important people in the field; encourage your own interests, rather than promoting their own; be available to give you advice on the direction of your thesis and your career; and help you to find a job when you finish.

I was lucky to get all of the above; however, these desiderata were fulfilled not by a single advisor, but by three advisors: Yoav Shoham, Jerry Friedman, and Nils Nilsson.

I would like to thank my official advisor, Yoav Shoham. Although Yoav's interests in machine learning were limited, he gave me the freedom to pursue my interests and provided me with many of the resources I needed, and beyond. Yoav was also my mentor for American culture; he tried to teach me some good manners, but with only partial success.

Jerry Friedman was my technical advisor. He helped me understand machine learning from a different perspective, and his first invited talk at Nils' group meeting (Bots) convinced me to start looking into statistics more seriously. Jerry has helped me see many things in a different light; his intuition and comments were always helpful, and they kept me thinking for many hours after each meetings.

Without any doubt, the person who has had the most significant impact on my life at Stanford was Nils Nilsson. Nils is the perfect example of an altruistic person; he will go

out of his way to help others, and he has helped me many times in different ways. As one example of many, he sent the following e-mail to me in 4 June 1993:

> It occurred to me in an oxygen-deprived state while running early this a.m. that you would probably benefit immensely from attending this [Machine Learning conference], meeting people, *etc.* ...Additional oxygen later in the day didn't change my mind.

The ML-93 conference was the first machine learning conference I attended, and Nils funded the trip.

Nils supported the $\mathcal{MLC}$++ project (a Machine Learning library in C++), which was developed at Stanford (Kohavi, John, Long, Manley & Pfleger 1994, Kohavi & Sommerfield 1995*b*). He provided the original funding for the project and "loaned" the project money to keep it going when grant money was slow to arrive (basically always). He also gave me complete freedom to lead the project, including managing the grant money and paying students who worked on it, something that is almost unheard of. While Nils gave me complete autonomy, he was always there to help in writing the grant proposals and in decoding ambiguous messages from funding agencies.

The idea of writing a class library that would help researchers in machine learning started around May 1993, with the first official meeting on June 8, 1993. The original contributors were Wray Buntine, George John, Pat Langley, Ofer Matan, Nils Nilsson, Karl Pfleger, Scott Roy, Mehran Sahami, Anton Schwartz, and Yoav Shoham. Yoav kept warning me that this project would be a time sink and that I would never finish my dissertation. Early thoughts about making this my dissertation vanished when Tom Dietterich wrote to Yoav on 3 August 1993 that "it seems to me that this is not a dissertation project, but rather an important engineering undertaking." Nonetheless, the project continued, and although it was a great time sink, it has definitely helped my dissertation and will hopefully help others. Our last release on 1 June 1995 was copied by 136 sites in four months.

Many students have worked on $\mathcal{MLC}$++, and I thank them all for their help in pushing this project forward. Richard Long started working with me on the project in the summer of 1993 and stayed on for a year, then graduated. Dan Sommerfield took the project class in machine learning and then replaced Rich for the second year of the project. The following students have also worked on the project and deserve my gratitude: Robert Allen, Brian Frasca, James Dougherty, George John, Chia-Hsin Li (Jamie), David Manley, Svetlozar

Nestorov, Mehran Sahami, Greg Snyder, and Yeogirl Yun (Yogo). Part of the $\mathcal{MLC}$++ funding was provided by ONR grants N00014-94-1-0448 and N00014-95-1-0669, and NSF grant IRI-9116399. The graphs drawn in this dissertation were done through the $\mathcal{MLC}$++ interface to *dot*, a program written by Eleftherios Koutsofios and Stephen C. North from Bell Labs (Koutsofios & North 1994). $\mathcal{MLC}$++ also benefited from the use of LEDA, a library of efficient data types and algorithms (Naeher 1995).

I owe a lot to George John with whom I have interacted the most. We bounced a lot of ideas back and forth, and learned many things together. I thank Ofer Matan and Karl Pfleger for our many interactions. The machine learning reading group (MLRG), which the four of us formed, forced us all to read some hard papers, which would have gone unread otherwise.

At the Robotics lab, I wish to thank Scott Benson, Ronen Brafman, Nir Friedman, Lydia Kavraki, James Kittock, Jutta McCormick, Avrami Tzur, and everyone else on the Nobotics and Bots groups. Out in MJH, I wish to thank Andrew Kosoresow, Tomas Uribe, the gang at MUGS led by Mike ("Black hole") Genesereth, and Sara Merryman. Farhad Shakeri, the system administrator, was always helpful in solving system problems, and Emma Peas was kind enough to recompile TEX for me when I exceeded its limits in writing this dissertation (I'm sure Don Knuth has *some* explanation why the number of strings allowed is only 3546).

I thank the many teachers who taught me courses in machine learning and statistics: Wray Buntine, Trevor Hastie, Pat Langley, and Rob Tibshirani. I also wish to thank David Aha, Jason Catlett, Tom Dietterich, Usama Fayyad, Matt Ginsberg, Rob Holte, Shaul Markovitch, Richard Olshen, Ross Quinlan, Eddie Schwalb, Moshe Tennenholtz, Paul Utgoff, Sholom Weiss, and David Wolpert for comments they gave throughout. George Light (GE) helped me to improve my English and carefully read many of my draft papers.

On the personal side, I wish to thank my parents, Tikva and Yitzhak, for all they have done for me; it takes many years to realize how much of what you are depends on your parents. I wish to thank my two sisters, Orit and Karen, for the fun we have had together growing up. Yael Kleefeld, my girlfriend, deserves many thanks, especially for her patience. She waited for me many nights while I was playing with my "mistress," as she called my dual-cpu Sun Sparc 10. Yossi and Hadasa Kleefeld deserve special thanks for those Fridays when I finally got to eat homemade food. I thank Beth Winkelstein for the many nights we folk-danced together.

Usama Fayyad wrote in his dissertation that he often wondered why individuals exaggerate by thanking “almost everyone on the planet.” I have thanked a small fraction of the planet and ask forgiveness from those I have omitted unintentionally. Thank you all!

Contents

List of Tables

List of Figures

Chapter 1

Introduction

O Lord, I could have stayed here all the night
To hear good counsel. O, what learning is!
—William Shakespeare in Romeo and Juliet

This chapter provides an introduction to supervised classification learning and an overview of the dissertation. This introduction will be mostly informal, and the formal definitions will follow in Chapter 2. In Section 1.1, we define supervised classification learning and relate it to the bigger field of machine learning. In Section 1.2, we motivate the problem with which we are dealing. In Section 1.3, we discuss the importance of bias, or priors, which make learning possible. In Section 1.4, we describe the organization of the dissertation and major contributions. We conclude with a summary in Section 1.5.

1.1 Supervised Classification Learning

Without learning, men grow as cows do
increasing only in flesh not wisdom.
—Sakyamuni Buddha

Simon (1983) defined **learning** as "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." This definition is overly broad for our needs; therefore, we will narrow it down to **inductive learning**, or empirical learning.

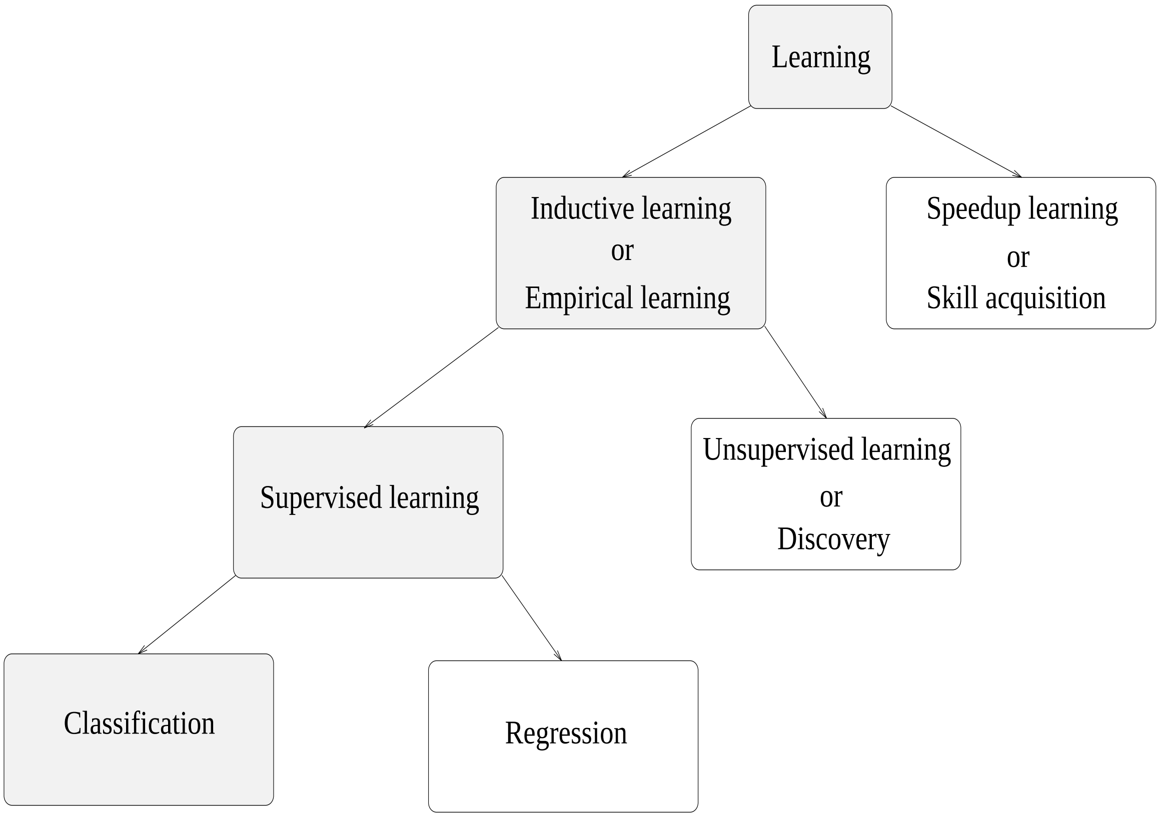

Figure 1.1: The learning hierarchy. Shaded nodes lead to supervised classification learning, the topic of this dissertation.

Empirical learning is "accomplished by reasoning from externally supplied examples to produce general rules" (Dietterich & Shavlik 1990, p. 1). In the inductive learning domain, we work on supervised classification learning problems. Figure 1.1 shows the learning hierarchy just described and related areas.

In **supervised classification learning**, the induction algorithm is given a set of examples, called a **training set**, in which each instance consists of a list of feature values and a discrete label. The list of feature values alone is called an **unlabeled instance**. The task of the algorithm is to learn a rule that correctly classifies new unlabelled instances. The term "supervised" suggests that some process, sometimes called the **teacher**, has labelled the instances in the training set. The term "classification" denotes the fact that the label is discrete, *i.e.*, consists of a few values.

Example 1.1 (A medical supervised learning problem)
Suppose we are trying to learn a rule for predicting whether a patient has a heart disease. We could find past records of patients, each record consisting of features such as age (integer), sex (male or female), cholesterol level (real number), presence of exercise induced angina (true or false).

Each record is presumed to be labelled by expert doctors, and the set of records is given to our induction algorithm as input. The output of the induction algorithm will be some rule to classify new patients. Figure 1.2 shows a portion of a decision tree that was induced from real-word data. The decision tree serves as our rule for classifying new patients: starting from the root, we repeatedly branch according to the feature tested until a leaf is reached, which labels the patient as sick or healthy. ∎

There are many classifiers that one can induce from data in the framework of supervised classification learning. In this dissertation, we will concentrate on induction algorithms that contribute to understanding the data as opposed to classifiers that aim only for high accuracy. For example, an induced decision tree might help doctors understand the data better, while a neural-network that has the same classification accuracy may be extremely hard for humans to understand. Induction algorithms that induce comprehensible structures aid in understanding the domain and may constitute new knowledge (Dietterich 1986, Newell 1982).

As shown in Figure 1.1, supervised classification learning is a small subset of the machine learning field. **Speedup learning,** exemplified by Explanation-Based Learning (Mitchell, Keller & Kedar-Cabelli 1986), deals with exploiting knowledge to speed up the efficiency of existing processes (*e.g.*, introduction of macro operators and metalevel control knowledge). **Unsupervised learning**, exemplified by clustering methods (Duda & Hart 1973, Krishnaiah & Kanal 1982, Cheeseman *et al.* 1988), deals with discovering structure in unlabelled instances. **Regression** problems deal with learning a function mapping from unlabelled instances to a real-valued label (Breiman, Friedman, Olshen & Stone 1984, Draper & Smith 1981).

1.2 Motivation

The three most important motivating factors for supervised classification learning are: data mining, overcoming the knowledge acquisition bottleneck, and improving upon expert performance.

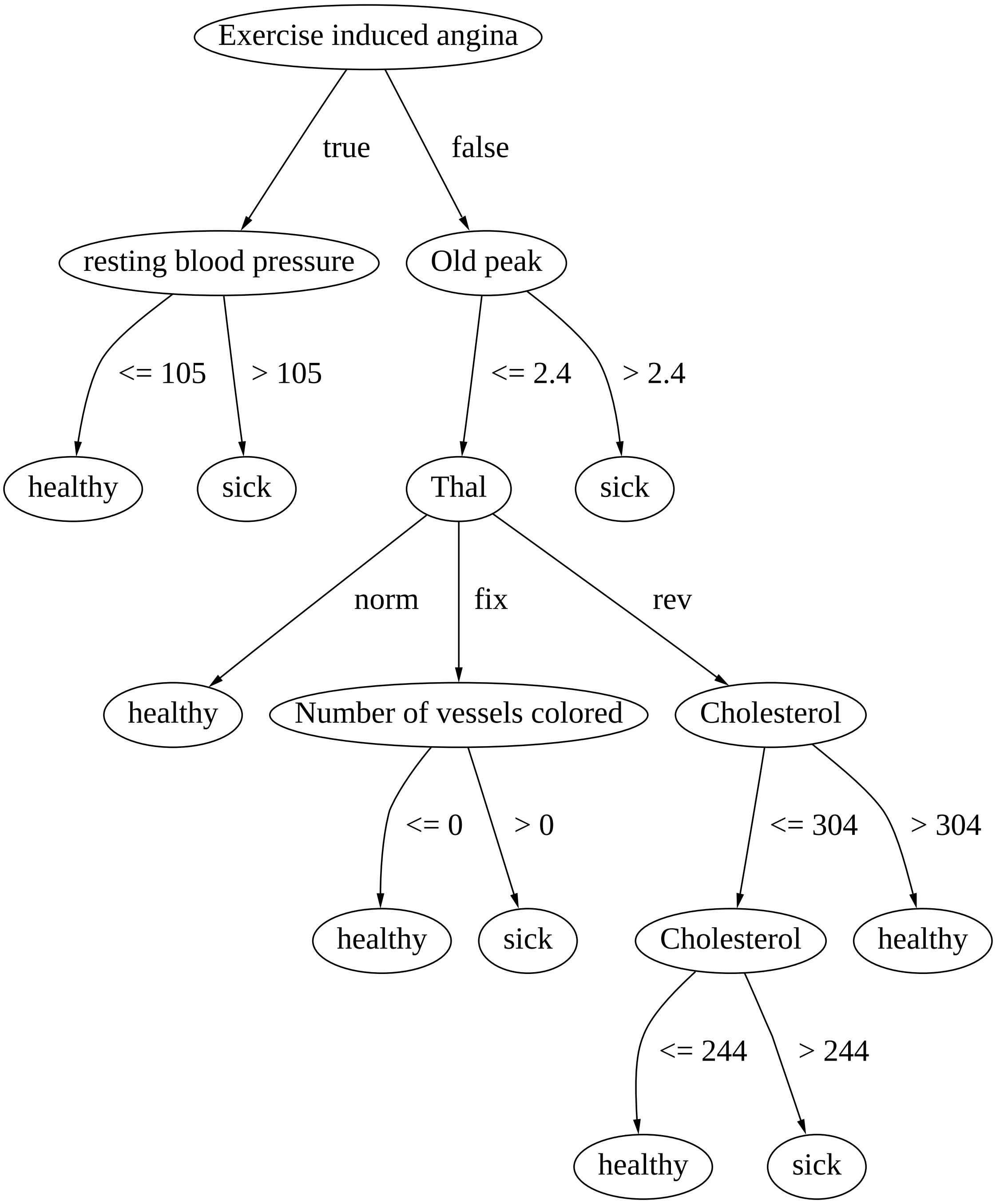

Figure 1.2: A portion of a decision tree induced by C4.5 for the Cleveland heart disease dataset (described later).

1.2.1 Data Mining

An ounce of knowledge is worth a ton of data.
—Brian R. Gaines, 1989

An ounce of gold may cost more to dig up than a ton of coal.
*—Australian mining industry metaphor, in Catlett (1991*a)

Data mining, or exploratory data analysis as it is sometimes called in Statistics, deals with the extraction of knowledge from data. Fayyad, Piatetsky-Shapiro & Smyth (to appear) define **data mining** as the "non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data."

The amount of data collected grows faster as storage technologies and data collection methods improve. Fayyad *et al.* (to appear) give some examples of massive datasets created recently:

1. Wal-Mart, a U.S. retailer, created a database that handles over 20 million transactions a day (Babcock 1994).

2. Mobil Oil Corporation, is developing a data warehouse capable of storing over 100 terabytes of data related to oil exploration (Harrison 1993).

3. The NASA Earth Observing System (EOS) of orbiting satellites and other spaceborne instruments is projected to generate on the order of 50 gigabytes of remotely sensed image data per *hour* when operational in the late 1990s and into the next century (Way & Smith 1991).

4. The sky catalog from the Palomar Observatory survey contains billions of entries with raw image data sizes measured in terabytes. Fayyad, Weir & Djorgovski (1993) describe the SKICAT project, which uses a decision tree to classify the objects from the sky survey into stars, galaxies, or instrumental artifacts.

The ability to extract interesting information and understand the data is of vital importance. The field of data mining is now growing rapidly with the increased need for smart data warehouses.

1.2.2 Knowledge Acquisition Bottleneck

For knowledge itself is power.
—Francis Bacon (1561-1626)

Expert systems (Feigenbaum, McCorduck & Nii 1988, Buchanan & Smith 1988) solve problems that are normally solved by human experts. To solve expert-level problems, expert systems need to build a large knowledge base, a task usually assigned to a **knowledge engineer.** Building the knowledge base is considered the bottleneck for building an expert system, and Feigenbaum (1977) described the knowledge engineer's task as to "uncover private knowledge by careful, painstaking analysis of a second party."

One motivation for research in machine learning, especially supervised classification learning, is to cut the knowledge acquisition time drastically. Michie (1987) wrote that in standard expert system construction a "knowledge engineer's object is to convert human know-how into say-how... [and by programming] into machine representation of the know-how." The way to cut the knowledge acquisition time is to bypass the narrow channel involving the knowledge engineer by using examples from the world or tutorial examples constructed by an expert (show-how).

Muggleton (1990) described some expert systems and the difference in man-years involved in constructing them. MYCIN (a medical diagnosis expert system) and XCON (a VAX computer configurator), had 400 and 8,000 rules, respectively, and were constructed by hand, *i.e.*, using knowledge engineers. GASOIL (an expert system for Hydrocarbon separation and configuration) and BMT (an expert system for configuring of fire-protection equipment in buildings) had 2,800 rules and 30,000 rules, respectively, and were constructed using ID3 (Quinlan 1986), a decision tree induction algorithm. The time to construct MYCIN and XCON was estimated to be an order of magnitude greater than BMT, which Muggleton claimed is the largest expert system in full-time commercial use.

Of course, induction cannot replace experts. Experts are important for constructing the appropriate set of features and for constraining the search space. Describing an expert, Michie (1987) wrote that "she has no trouble in knowing what low-level measurements [features] are relevant, although she commonly attributes relevance to additional measurements which later may be shown to be redundant." Gaines (1991) quantified the tradeoff in prior knowledge about relevant features. Clark & Matwin (1993) showed a similar use of background knowledge found in qualitative models.

An important requirement in expert systems, which is not necessary for many learning tasks, is the ability to explain the classifications. Michie (1987) wrote that "A system which gives good decisions but cannot explain itself in terms to which the human expert can relate may be a software product of great value, but it belongs to some other category: operations research, decision support systems, automatic control, and so on."

1.2.3 Improving Upon Expert Performance

Kononenko (1993) references 24 papers where inductive learning systems were actually applied in medical domains, such as oncology, liver pathology, prognosis of patient survival in hepatitis, urology, diagnosis of thyroid diseases, rheumatology, diagnosing craniostenosis syndrome, dermatoglyptic diagnosis, cardiology, neuropsychology, gynecology, and perinatology. He remarks that "typically, automatically generated diagnostic rules slightly outperformed the diagnostic accuracy of physicians specialists."

Fayyad *et al.* (1993) report that their decision tree outperformed astronomers in the sky survey and that for the majority of these objects, the astronomers were not able to determine the classes by examining the survey images.

1.3 The Need for Bias in Machine Learning

> ***The Knowledge Principle:*** *A system exhibits intelligent understanding and action at a high level of competence primarily because of the specific knowledge that it contains about its domain of endeavor... If a program is to perform well, it must know a great deal about the "world" in which it operates.*
> *—Feigenbaum (1988)*

In order to generalize one must make assumptions, which are called **biases** in machine learning (not to be confused with statistical bias, which is explained later). Mitchell (1982) wrote:

> Although removing all biases from a generalization system may seem to be a desirable goal, in fact the result is nearly useless. An unbiased learning system's ability to classify new instances is no better than if it simply stored all the training instances and performed a lookup when asked to classify a subsequent instance.... An unbiased system is one whose inferences logically follow from the training instances, whereas classifications of new instances do not logically follow from the classifications of the training instances.

Interesting conclusions that follow the general statement above were made by Schaffer (1994). Assuming all targets are equiprobable in a discrete domain with two possible classes, and measuring the **generalization accuracy**, which is the probability of making a correct prediction on *unseen* instances, the following observations were made:

1. A majority induction algorithm, that always guesses the prevalent class in the training set, will have a generalization accuracy of exactly 50%.

   The intuition that the majority induction algorithm will detect a difference when the prevalences are different is correct; however, when the classes are equally likely, the algorithm will perform slightly below 50% because a majority in the training set will imply a minority in the test set!

2. Learning curves using generalization accuracy (*i.e.*, a plot of the generalization accuracy as a function of the number of instances) must sometimes decrease with the number of instances.

3. The superiority (in terms of generalization accuracy) of one induction algorithm over another in some situations must be balanced exactly by the superiority of the latter algorithm over the former.

4. Cross validation, data diagnostic, and meta-level learning cannot overcome the above limitations.

One unrealistic assumption made by Schaffer is that all target concepts are equiprobable. Wolpert (1994*b*) presents the no-free-lunch theorems, which are more general than Schaffer's conservation law. The main theorem states that how well an induction algorithm does is determined by how "aligned" the learning algorithm is with the posterior probability of a target function given the data. More formally, an induction algorithm can be theoretically described as $\Pr(h \mid \mathcal{D})$, a probability distribution of hypotheses generated given training sets. If $\Pr(f \mid \mathcal{D})$ is the probability of a given target function f given a training set $\mathcal{D}$, then the generalization accuracy is

$$\sum_{h,f} u(h, f, \mathcal{D}) \Pr(h \mid \mathcal{D}) \Pr(f \mid \mathcal{D})$$

for a utility function u. If the probability assigned to a hypothesis h given a dataset $\mathcal{D}$ is high (or one for deterministic induction algorithms) and the probability that this function was indeed the generating function, then for zero/one-utility ($u(h, f, \mathcal{D}) = 1$ if $h = f$ and zero otherwise) the accuracy will be high.

The no-free-lunch theorems formalize the principle that knowledge facilitates learning, defined in (Lenat & Feigenbaum 1991) as "if you don't know very much to begin with, don't expect to learn a lot quickly." The question of interest to researchers in machine learning is how to define the biases of existing algorithms and how to find out when a given bias is appropriate, based on background knowledge. Biases can be divided into two types: restricted hypothesis space bias and preference bias.

Restricted hypothesis space bias This bias assumes that the target function belongs to some restricted space of hypotheses, typically defined in terms of their representation. For example, the perceptron algorithm searches only the space of linear threshold functions.

Preference bias This bias places a preference ordering on hypotheses. Many times the preference ordering is defined by how the search through the space of hypotheses is conducted. Most preference biases attempt to minimize some measure of syntactic complexity, following Occam's Razor principle of preferring simpler hypotheses (Blumer, Ehrenfeucht, Haussler & Warmuth 1987).

Most decision tree induction algorithms restrict the hypothesis space to the space of finite trees that conduct threshold splits on continuous features and equality or subset splits on discrete features. Decision tree induction algorithms generally employ a simplicity bias, preferring small decision trees over large ones.

Making threshold splits on continuous features is one example of a bias that is used by many induction algorithms. Our prior experience with the real world indicates that behavior is *smooth* in many cases. For example, if the number of times a woman was pregnant is relevant to whether she is likely to have diabetes, we would not expect the target to depend on whether the number is even or odd, but rather whether it is high or low. Similarly, if the temperature of a patient is relevant to whether he or she has some disease, we would expect smoothness in the sense that patients with very close temperatures should behave similarly if all other factors remain constant.

A bias that we investigate in a large part of this dissertation is the bias for a small number of features to base our classification on. Given an instance with many features, we will assume that it unlikely that all features are necessary. In many cases, our background knowledge indicates that this is a good bias. For example, most people would agree that not all fields (features) in a patient record are relevant for predicting heart disease; specifically, the patient number is most likely irrelevant.

1.4 Organization and Contributions

This dissertation can be divided into three main sections: accuracy estimation, wrappers, and OODGs. We now describe the organization of the dissertation and then describe the motivation for the work on this areas, which was almost in the opposite order to the way it is presented now.

1.4.1 Organization

No amount of organization can counteract determined laziness.
—Chris Hansen

In Chapter 2, we formalize supervised classification learning and the terms used throughout the dissertation. We describe common induction algorithms and explain the choice of datasets we made. We briefly describe the bias-variance tradeoff and conclude the chapter with a description of the experimental methodology used. Appendix F contains a table of symbols and notations.

In Chapter 3, we review accuracy estimation methods, and experimentally compare cross-validation and the .632 bootstrap. Accuracy estimation is a crucial part of the wrapper approach, and we attempt to stabilize the highly variable estimates of cross-validation.

In Chapter 4, we describe the wrapper approach, which is based on the simple idea of optimizing parameters based on the estimated accuracy of the classifier. We investigate the problem of feature subset selection, *i.e.*, hiding some features from the induction algorithm to improve its accuracy, and define our goal of an optimal feature subset. We discuss relevance and show that, while related to the optimal feature subset, one should not always select all relevant features. We test different search engines to search the space of feature subsets and introduce compound operators to speed the search. We conclude with an example of how the wrapper approach can be used to tune parameters for C4.5 (Quinlan 1993) and improve its performance on artificial and real-world problems.

In Chapter 5, we evaluate the power of feature subset selection as the only inductive process in an inducer. We use a structure called DTM, which is a decision table with a default majority rule, to test the hypothesis that there is a lot of generalization power in simply selecting a subset of features and matching their values in queries. The DTM structure can be updated easily, and we can *incrementally* cross-validate it, thereby reducing the running time of the accuracy estimation step of the wrapper approach. We conclude that much of the inductive power indeed comes from feature subset selection, and that for the datasets tested, a small subset of features has high predictive power. Appendix B shows General Logic Diagrams (Michalski 1978), which are two dimensional projections of the projected space, for the target concepts.

In Chapter 6, we describe OODGs, oblivious read-once decision graphs, and show that such graph structures have some advantages over decision-trees. We show a basic framework for learning OODGs bottom-up and discuss the strengths and limitations of our approach. We conclude by combining the basic algorithm with the wrapper approach and show that the resulting graphs are accurate for the datasets we tested, and in many cases very comprehensible. Appendix D shows trees generated by C4.5 and graphs generated by our HOODG algorithm.

Chapters 3 to 6 each include a section on future work and a summary section. We conclude with a summary in Chapter 7. Appendix E contains a global comparison of all induction algorithms used and many others. An index can be found at the end of the dissertation.

1.4.2 A Brief History

If at first you don't succeed, you are running about average.
—M. H. Alderson

Figure 1.3 shows the chronological development of the topics, with the chapters they are discussed in shown in small circles at the top of each node in the graph.

In May 1993, the idea of Oblivious Decision Graphs emerged with the name "Kite Pipeline: a Structure for Representing Discrete Functions" (Kohavi, draft paper). A few months later, after we understood the topic much better, it was clear that Lee (1959) had already thought of this idea and that there is a whole community—the OBDD community—that works with isomorphic structures. Learning the structures was still uninvestigated.

In Kohavi (1994*a*), the bottom-up algorithm was described. In Kohavi (1994*b*) the limitations began to emerge, with an understanding that feature subset selection must be done.

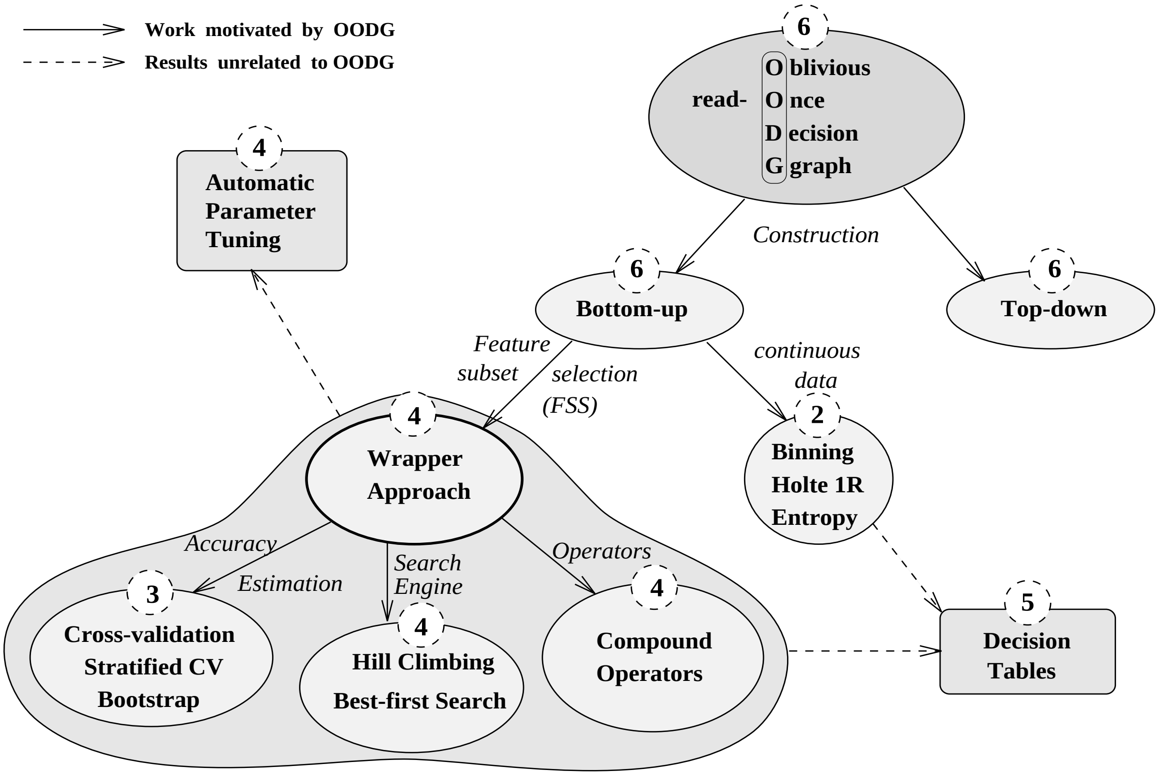

Figure 1.3: The chronological flow of topics. The small circles at the top of each node indicate the chapters of this dissertation in which they appear.

In John, Kohavi & Pfleger (1994) the wrapper approach was introduced as a method for feature subset selection that will work with any induction algorithm. For about a year most of the effort was concentrated on the problem of feature subset selection (Kohavi 1994*c*, Kohavi & Sommerfield 1995*a*). The results on feature subset selection using the wrapper approach were generalized into general optimization of parameters (Kohavi & John 1995) and specialized into subset selection for decision tables (Kohavi 1995*a*).

Rough sets were related to decision tables and to OODGs (Kohavi & Frasca 1994, Kohavi 1994*d*). The work with the wrapper approach led to a large experiment to see which accuracy estimation method should be used (Kohavi 1995*b*).

Recently, work on OODGs resumed with an experimental comparison of discretization methods to allow experiments on real datasets (Dougherty, Kohavi & Sahami 1995), and a top-down induction algorithm (Kohavi & Li 1995). Chapter 6 combines most of the work done by Kohavi in the last two years: an entropy discretization is used to discretize the data, the top-down algorithm finds a good initial subset, and the wrapper searches for a better node using the original HOODG algorithm for inducing OODGs as a black box.

1.4.3 Principal Contributions

Jones's First Law: Anyone who makes a significant contribution to any field of endeavor, and stays in that field long enough, becomes an obstruction to its progress—in direct proportion to the importance of their original contribution.
—Unix fortune

The following is a list of contributions to the field of machine learning made in this dissertation:

1. The use of OODGs as a hypothesis space and the introduction of the bottom-up algorithm.

2. The use of DTMs (decision tables with default majority) as a hypothesis space showing the surprising power of feature subset selection.

3. The wrapper approach and its usage in feature subset selection and parameter tuning.

4. Compound operators for feature subset selection.

5. The large experiment comparing cross-validation variants and bootstrap. The conclusion that leave-one-out is *not* the best cross-validation method for model selection. The realization that the .632 bootstrap is highly biased for real-world datasets and common induction algorithms (this motivated the introduction of the .632+ method by Efron and Tibshirani).

6. The definitions of relevance and irrelevance for features, and their relation to the optimal feature subset.

7. The properties of OODGs including the Kite Theorem, Neighbor Exchange Theorem, and the Adjacency Theorem.

8. The introduction of an initial feature subset node for OODGs based on conditional entropy.

9. The proof that a multi-level projection for OODGs is NP-hard.

1.5 Summary

I have been over into the future, and it works.
—Lincoln Steffens (1866-1936)

Learning without bias is impossible. To make machine learning work, we must investigate different structures that may be appropriate for different contexts and understand their strengths and limitations.

Machine learning works in many practical applications because the target concepts are not equiprobable as assumed in Schaffer (1994) and because our algorithms are somewhat "aligned" with real world phenomena: features are usually selected by experts and smoothness assumptions that algorithms assume do hold in many cases.

The more we understand about the underlying structures used in classifiers (*e.g.*, decision trees, graphs), the more we can modify them based on background knowledge. Similarly, the more we understand the induction algorithms and their assumptions, the easier it is to modify them.

Chapter 2

Supervised Classification Learning: Definitions and Methodology

If your thesis is utterly vacuous
Use first-order predicate calculus.
With sufficient formality
The sheerest banality
Will be hailed by the critics: "Miraculous!"
—Henry A. Kautz, in Brachman (1987)

We begin by formalizing the supervised classification learning problem and the terms used throughout the dissertation. We describe common learning algorithms that are used in this dissertation, and explain the choice of datasets we made. We briefly describe the bias-variance tradeoff and conclude the chapter with a description of the experimental methodology used. Readers not familiar with machine learning might wish to consult an introductory text on machine learning, such as Langley (1995) or Weiss & Kulikowski (1991). An excellent collection of papers in machine learning can be found in Dietterich & Shavlik (1990).

2.1 Definitions

The beginning of wisdom is the definition of terms.
—Socrates (469-399 B.C.)

Informally, the task of a supervised classification learning program is to generate a "good" classifier from a labelled set of examples. The classifier can then be used to classify unlabelled examples with the goal of correctly predicting the label of each unlabelled example. The classifier can be evaluated for accuracy, comprehensibility, compactness, and other desirable properties that determine how good and appropriate it is for the task at hand.

An **instance**, sometimes called an example, is a fixed list of **features** values. An instance describes the basic entity that we are dealing with, such as a person, a mushroom, a board position in a chess game, or a DNA sequence.

A **feature**, sometimes called an **attribute**, describes some characteristic of an instance. We use two features types: nominal (color $\in$\{red,green,blue\}) and continuous (height$\in \mathbb{R}$, a real number). Continuous features are used whenever there is a linear ordering on the values, even if they are not truly continuous (*e.g.*, height may be specified to the nearest inch or centimeter).

Every instance has a special nominal feature, the **label**, which describes the phenomenon of interest, *i.e.*, the phenomenon we would like to learn and make predictions about. An **unlabelled instance** is the part of the instance without the label, *i.e.*, the list of feature values. A **dataset** is a set of labelled instances. Table 2.1 shows a dataset with seven instances from a heart-disease domain. The last column, *sick*, is what we try to predict given the other features.

A **classifier** is a function that maps an unlabelled instance to a label. All classifiers use a stored data structures that is then interpreted as a mapping for an unlabelled instances to a label. For example, a decision tree classifier contains a stored decision tree that maps an unlabelled instance to a category by following the path from the root to a leaf (defined by the tests at the nodes) and returns the category at the leaf; a one nearest-neighbor (see below) classifier finds the nearest-neighbor in a set of internally stored labelled instances and returns its label. Throughout this dissertation, we will consider only deterministic classifiers, but the definition can be extended to nondeterministic classifiers as well; note also that in most cases, it is the induction algorithm that is nondeterministic, not the classifier.

Table 2.1: A dataset of seven instances from a heart-disease domain. The first line describes the feature names, the second line the feature types (continuous or nominal), and the other seven lines the instances, each described by a list of five feature values and a label value.

Age	Sex	cholesterol	resting ECG	max heart rate	sick
(cont)	$\{M, F\}$	(cont)	$\{norm, abn, hyp\}$	(cont)	$\{yes, no\}$
53	M	203	hyp	155	yes
60	M	185	hyp	155	yes
40	M	199	norm	178	no
46	F	243	norm	144	no
62	F	294	norm	162	no
43	M	177	hyp	120	yes
76	F	197	abn	116	no
62	M	267	norm	99	yes
57	M	274	norm	88	yes

An **inducer**, or an **induction algorithm**, builds a classifier from a given dataset. CART (Breiman *et al.* 1984) and C4.5 (Quinlan 1993) are decision tree inducers that build decision tree classifiers.

We now formally define the symbols used. Appendix F contains a summary of these symbols. Let the domain of feature X_i be $\mathrm{Dom}(X_i)$. Each unlabelled instance is an element of the unlabelled instance space $\mathcal{X} = \mathrm{Dom}(X_1) \times \mathrm{Dom}(X_2) \times \cdots \times \mathrm{Dom}(X_n)$, where n is the number of features. We denote an unlabelled instance by $\vec{X}$ and its value (feature vector) by $\vec{x}$. The value of a specific feature X_i is denoted by x_i.

Let $\mathcal{Y}$ be the set of possible label values. The label is denoted by Y and the label value is denoted by y. Let $\mathcal{X} \times \mathcal{Y}$ be the space of labelled instances and $\mathcal{D}$ be a dataset (possibly a multiset) consisting of m labelled instances, where each instance i is $\langle \vec{x}_i \in \mathcal{X}, y_i \in \mathcal{Y} \rangle$.

A classifier $\mathcal{C}$ maps an unlabelled instance $\vec{x} \in \mathcal{X}$ to a label $y \in \mathcal{Y}$ and an inducer $\mathcal{I}$ maps a given dataset $\mathcal{D}$ into a classifier $\mathcal{C}$. The notation $\mathcal{I}(\mathcal{D}, \vec{x})$ will denote the label assigned to an unlabelled instance $\vec{x}$ by the classifier built by inducer $\mathcal{I}$ on dataset $\mathcal{D}$, *i.e.*, $\mathcal{I}(\mathcal{D}, \vec{x}) = \mathcal{C}(\vec{x}) = (\mathcal{I}(\mathcal{D}))(\vec{x})$. We assume that there exists a distribution D on the set of labelled instances and that our dataset consists of i.i.d. (independently and identically distributed) instances.

The **accuracy** of a classifier $\mathcal{C}$ is the probability of correctly classifying a randomly selected instance, *i.e.*, $\mathtt{acc} = \mathtt{Pr}(\mathcal{C}(\vec{x}) = y)$ for a randomly selected instance $\langle \vec{x}, y \rangle \in \mathcal{X} \times \mathcal{Y}$, where the probability distribution over the instance space is the same as the distribution that was used to select instances for the inducer's training set. Note that the desired goal is to estimate the accuracy of the classifier induced from the *given* dataset, not the expected accuracy of a classifier induced from a dataset of the same size randomly drawn from the parent population. The former is termed **conditional accuracy**, while the latter is termed **expected accuracy**.

The task of an induction algorithm is to induce a classifier that has the following desired features:

1. It is accurate. This requirement is usually the most important feature, and it will be the main consideration in this dissertation.

2. It is comprehensible to humans. Given two classifiers with approximately equal accuracy, we might prefer the one that is also comprehensible. For some domains, such as medical domains, comprehensibility is crucial; for others, such as hand-written recognition, it is of no importance. In this dissertation, we give comprehensibility an important weight.

3. It is compact. While related to comprehensibility, one does not imply the other. A perceptron (see below) might be a compact classifier, yet given an instance, it may be hard to understand the labelling process. Alternatively, a decision table (Kohavi 1995*a*) (also see Chapter 5 on page 130) may be very large, yet labelling each instance is trivial: simply look it up in the table.

   Michie (1987) reported that when ID3's output on the chess domain was shown to a domain expert, *i.e.*, a chess master, it was completely opaque. Although it was very accurate, the tree was large, obscure, and the chess master was in a "total blackout." The phenomenon was confirmed for related chess material in Shapiro & Niblett (1982) and similar claims about other domains were made by Cendrowska (1987).

Some researchers, and most of the Statistics community, use error rates (one minus the accuracy) instead of accuracy.[1] We chose to use accuracy because it is the more common

[1] Jerry Friedman says that computer scientists are more optimistic than statisticians because we use accuracy and not error rates; Von Neuman's sampling method is called "acceptance sampling" in the Computer Science literature and "rejection sampling" in the Statistics literature. Although both measures are clearly equivalent in theory, the problem of which measure to use is ubiquitous. In the supermarkets, some companies write that their product is 96% fat free, yet others write that it has 4% fat.

measure in the machine learning community, but we do use error rates when the advantages are clear. Specifically, the **relative reduction in error** between algorithm A and algorithm B (which has higher accuracy) is $(\texttt{acc}(B) - \texttt{acc}(A))/(1 - \texttt{acc}(A))$. If algorithm A has an accuracy of 98% and algorithm B has an accuracy of 99%, then the relative reduction in error is 50%. Although this measure is sometimes more appropriate than absolute difference in accuracy, improving from 80% to 82% may be harder than from 96% to 98% because in the former case, the best possible accuracy might be 82%, while in the latter case it may be 100%. All artificial datasets used in this dissertation are deterministic concepts, and thus it is possible to achieve 100% accuracy on them all; for real datasets, the highest possible accuracy is unknown, but it is probably not 100% in most domains.

The above definitions can be extended in many ways, and most of the work presented in this dissertation can easily be generalized in several ways. For example, we consider equal misclassification costs using a 0/1 loss function, but the accuracy estimation techniques (Chapter 3 on page 35) and the wrapper approach (Chapter 4 on page 76) trivially extend to other loss functions. The features can be extended to other types, such as linear (but not continuous) and tree-structured (Haussler 1988), but this issue is orthogonal to this dissertation; algorithms that support these types could be used with methods described here. We assume a flat-file format, where the instances contain a fixed number of features. Inductive logic programming (ILP) techniques (Lavrac & Dzeroski 1994, Lavrac & Dzeroski 1994) can deal with variable formats, and techniques described here could conceivably be used in ILP, although they have not been tried.

2.2 Induction Algorithms

> *Following the middle ages, the origin of this word [algorithm] was in doubt, and early linguists attempted to guess at its derivation by making combinations like* algiros *(painful) + arithmos (number).*
> *—Knuth, The Art of Computer Programming (1973)*

Throughout this dissertation, we use two induction algorithms as a basis for comparisons. These are the C4.5 induction algorithms and the Naive-Bayes induction algorithm. Both are well known in the machine learning community and represent two completely different approaches to learning, hence we hope that our results are of a general nature and will generalize to other induction algorithms. Decision trees have been well documented in

Quinlan (1993), Breiman *et al.* (1984), Fayyad (1991), Buntine (1992), and Moret (1982); hence we will not describe them in detail. The Naive-Bayes algorithm is explained below. The specific details are not essential for the rest of the dissertation.

The **C4.5** algorithm (Quinlan 1993) is a descendent of ID3 (Quinlan 1986), which builds decision trees top-down and prunes them. The tree is constructed by finding the best single-feature test to conduct at the root node of the tree. After the test is chosen, the instances are split according to the test, and the subproblems are solved recursively. C4.5 uses gain ratio, a variant of mutual information, as the feature selection measure; other measures have been proposed, such as the Gini index (Breiman *et al.* 1984), C-separators (Fayyad & Irani 1992), distance-based measures (De Mántaras 1991), and Relief (Kononenko 1995*b*). C4.5 prunes by using the upper bound of a confidence interval on the resubstitution error as the error estimate; since nodes with fewer instances have a wider confidence interval, they are removed if the difference in error between them and their parents is not significant.

We reserve the term **ID3** to a run of C4.5 that does not execute the pruning step and builds the full tree (*i.e.*, nodes are split unless they are pure or it is impossible to further split the node due to conflicting instances). The ID3 induction algorithm we used is really C4.5 with the parameters `-m1 -c100` that cause a full tree to be grown and only prune if there is absolutely no increase in the resubstitution error rate. A relatively unknown post processing step in C4.5 replaces a node by one of its children if the accuracy of the child is considered better (Quinlan 1993, page 39). In one case (the corral database described below), this had a significant impact on the resulting tree: although the root split was incorrect, it was replaced by one of the children.

CART (Breiman *et al.* 1984) also builds decision trees top-down and prunes them, but the pruning mechanism is drastically different. Pruning in CART is done using ten-fold stratified cross-validation, and hence the algorithm runs an order of magnitude slower. Esposito, Malerba & Semeraro (1995*a*, 1995*b*) compared six different pruning methods on UC Irvine datasets and concluded that there is no significant difference in accuracy between C4.5's pruning and CART's pruning (in fact, C4.5 was slightly superior). Mehta, Rissanen & Agrawal (1995) have recently observed similar results, but noted that the trees generated by the CART-like algorithm were much smaller.

The **Naive-Bayesian** classifier (Langley, Iba & Thompson 1992, Duda & Hart 1973, Good 1965, Anderson & Matessa 1992, Taylor, Michie & Spiegalhalter 1994) uses Bayes rule to compute the probability of each class given the instance, assuming the features are conditionally independent given the label. Formally,

$$\begin{array}{ll}
\Pr(Y = y \mid \vec{X} = \vec{x}) & \text{by Bayes rule} \\
= \Pr(\vec{X} = \vec{x} \mid Y = y) \cdot \texttt{Pr}(Y = y) / \Pr(\vec{x}) & P(\vec{x}) \text{ is same for all label} \\
 & \text{values.} \\
\propto \Pr(X_1 = x_1, \ldots, X_n = x_n \mid Y = y) \cdot \texttt{Pr}(Y = y) & \text{by independence} \\
= \prod_{i=1}^{n} \Pr(X_i = x_i \mid Y = y) \cdot \texttt{Pr}(Y = y)\ . &
\end{array}$$

The version of Naive-Bayes we use in our experiments was implemented in $\mathcal{MLC}$++ (Kohavi *et al.* 1994). The probabilities for nominal features are estimated from data. The probabilities for continuous features are assumed to be coming from a Gaussian distribution, and we estimate the mean and standard deviation from the data. Unknown values in a test instance (an instance that needs to be labelled) are ignored, *i.e.*, they do not participate in the product. In case of zero occurrences for a label value and a feature value, we use the $.5/m$ as the probability, where m is the number of instances. Other approaches are possible, such as using Laplace's law of succession or using a beta prior (Good 1965, Cestnik 1990). In these approaches, the probability for n successes after N trials is estimated at $(n + a)/(N + a + b)$, where a and b are the parameters of the beta function. The most common choice is to set a and b to one, and estimating the probability as $(n + 1)/(N + 2)$, which is Laplace's law of succession.

The Gaussian assumption for continuous features is unrealistic in many cases. In some parts of the thesis, especially when making comparisons between algorithms, we prediscretize the data using a mutual information discretization procedure. Section 2.3 on the next page describes the discretization procedure in detail.

While decision trees are considered to be very comprehensible classifiers, Naive-Bayes is simple enough that people can easily understand its mapping, especially when log probabilities are used. Kononenko (1993) reports that physicians naturally interpreted the log probabilities of Naive-Bayes as an additive scoring function and found the explanation ability to be very natural. The overall impression was that the form of explanation typically replicates their way of diagnosing, *i.e.*, summation of evidence for/against the diagnosis. The physicians also considered certain paths in the decision tree as new information that is interesting and warrants further investigation.

Throughout the thesis, we will mention some other induction algorithms. A **Nearest-neighbor**, or an **instance-based** algorithm classifies an instance-based on a vote of the k nearest neighbors under some distance metric (Dasarathy 1990, Wettschereck 1994, Duda & Hart 1973, Devijver & Kittler 1982, Aha, Kibler & Albert 1991). A **perceptron**, sometimes called a neuron, is a classifier that is usually associated with the perceptron error correcting rule (Rosenblatt 1958, Nilsson 1990, Minsky & Papert 1988). Multi-layer network of perceptrons are called **neural networks** and are usually associated with the backpropagation rule for learning (Rumelhart, Hinton & Williams 1986, Hertz, Krogh & Palmer 1991).

A **Bayes rule**, or a Bayes classifier, is a rule that predicts the most probable class for a given instance, based on the full distribution D (assumed to be known). The accuracy of the Bayes rule is the highest possible accuracy, and it is mostly of theoretical interest, as in practice we do not know the distribution D.

2.3 Discretization

Statisticians do it continuously but discretely.
After all, it's only normal.
—Unix fortune

Some induction algorithms require pre-discretization of the data (*e.g.*, the algorithm for inducing oblivious decision graphs described in Chapter 6 on page 141). Other algorithms, such as Naive-Bayes described above and Decision Tables (see Chapter 5 on page 130) improve in practice when discretization is used. An experimental comparison between different discretization methods was done in Dougherty *et al.* (1995) and, for the datasets tested, discretization by mutual information turned out to be superior to uniform binning, a class-blind unsupervised discretization method, and to Holte's 1R discretization method, which is supervised (Holte 1993).

The discretization method used in this dissertation is based on minimizing mutual information, originally presented in Catlett (1991*b*) and later refined in Fayyad & Irani (1993). The method is implemented in $\mathcal{MLC}$++ (Kohavi *et al.* 1994). This supervised discretization algorithm uses the mutual information of the class partitions and the partitions formed by different thresholds to select the discretization boundaries. The notation below closely follows the notation of Fayyad & Irani (1993). If we are given a set of instances S, a feature X, and a partition boundary T, that splits S into S_1 and S_2, the class information entropy of the partition induced by T, denoted $E(X, T; S)$ is given by:

$$E(X, T; S) = \frac{|S_1|}{|S|}\mathrm{Ent}(S_1) + \frac{|S_2|}{|S|}\mathrm{Ent}(S_2) \ .$$

For a given feature X, the boundary T_{min} which minimizes the entropy function over all possible partition boundaries is selected as a binary discretization boundary. This method can then be applied recursively to both of the partitions induced by T_{min} until some stopping condition is achieved, thus creating multiple intervals on the feature X.

Fayyad and Irani make use of the *Minimum Description Length Principle* (Rissanen 1978, Rissanen 1986, Wallace & Freeman 1987) to determine a stopping criterion for their recursive discretization strategy. Recursive partitioning within a set of values S stops iff

$$\mathrm{Gain}(X, T; S) < \frac{\log_2(m_S - 1)}{m_S} + \frac{\Delta(X, T; S)}{m_S},$$

where m_S is the number of instances in the set S,

$$\mathrm{Gain}(X, T; S) = \mathrm{Ent}(S) - E(X, T; S),$$

$$\Delta(X, T; S) = \log_2(3^k - 2) - k \cdot \mathrm{Ent}(S) - k_1 \cdot \mathrm{Ent}(S_1) - k_2 \cdot \mathrm{Ent}(S_2),$$

and k_i is the number of class labels represented in the set S_i. Since the partitions along each branch of the recursive discretization are evaluated independently using this criterion, some areas in the continuous spaces will be partitioned very finely, whereas others, which have relatively low entropy, will be partitioned coarsely. We do not show how these formulas were derived and refer the reader to Fayyad & Irani (1993) for the details.

There are different ways of encoding trees, as shown in Quinlan & Rivest (1989) and later in Wallace & Patrick (1993). Fayyad and Irani have proposed one stopping criterion that seemed to work well in practice, and we believe minor variations will work similarly.

When instances are discretized, it is important to use only the training data for determining the discretization intervals and discretize the test data using the intervals determined from the training set. In k-fold cross-validation runs, the instances are discretized k times, once for each fold.

Theoretically, the discretization should be part of the induction algorithm, and it should be done over and over when using it in the wrapper approach (Chapter 4); however, the discretization process is rather slow, so we chose to discretize the training set once and then use the wrapper on the discretized data. This methodology leads to slightly optimistic internal estimates because the discretization intervals were determined using the whole training set and not using the internal folds of the training set. This bias is sufficiently small, especially for Naive-Bayes where the discretization is used, that it was worth the time saving. Note that this approximation only affects the internal estimates in the wrapper approach; results could possibly be improved slightly if discretization was used at every stage. In any case, this approximation is internal to the induction algorithm itself and does not affect the outer cross-validation that is used to test the induction algorithm. In no way does this approximation invalidate any of the final accuracy results reported.

2.4 The Datasets

Minds think with ideas, not information. No amount of data, bandwidth, or processing power can substitute for inspired thought.
—Clifford Stoll, Silicon Snake Oil, 1994

The performance of different algorithms in this dissertation was evaluated on both artificial and real-world domains. Artificial domains are useful because they allow us to vary parameters, understand the specific problems that algorithms exhibit, and test conjectures. Real-world domains are useful because they come from real-world problems that we do not always understand and are therefore actual problems on which we would like to improve performance. All real-world datasets used are from the UC Irvine repository (Murphy & Aha 1995), which contains over 100 datasets mostly contributed by researchers in the field of machine learning[2] The real-world datasets were chosen based on the following criteria: dataset size, reasonable encoding, comprehensibility, non-triviality, and age.

Dataset size We chose datasets that had at least 300 instances, so that the variance of the estimates would not be too large. Even at 300, the variance is rather large, but we could not find enough datasets with at least 500 instances. This restriction ruled out many datasets, including glass, breast cancer (Ljubljana), golf, hepatitis, iris, labor negotiations, lenses, lung cancer, and lymphography.

[2]Many comments have been made about the "real-world datasets" when referring to the UC Irvine repository. While many databases were stored there because some algorithm was successful at learning them, we have attempted to select the larger and the nontrivial datasets.

Reasonable/correct encodings Some datasets have data in encodings that make the learning task unreasonably hard for no good reason. For example, in the breast cancer database from Ljubljana (different than the one from Wisconsin), the feature values were discretized using uniform binning; tumor sizes are groups of five, ages are in intervals of ten, *etc.* With such encodings, it is extremely hard to learn useful concepts, and indeed there are very few algorithms that can significantly improve upon the majority prediction. In the anneal dataset, inapplicable values are incorrectly marked as unknowns.[3]

Comprehensibility Given the emphasis on comprehensibility in this dissertation, domains that cannot be interpreted were ruled out. For example, pixel maps from satellite images (satimage dataset) were ruled out because in pixel-like domains, all the features are equally important (the image may be translated) and there is no easy way to interpret the resulting classifier, say a decision tree. The following datasets were also eliminated for this reason: ionosphere, letter, net-talk, segment, tic-tac-toe, and vehicle. Classifiers such as nearest-neighbors and neural-nets are more suited for these parallel tasks (Quinlan 1994), unless the data are preprocessed (see Dietterich, Hild & Bakiri (1995) for an example where using block-encoding on a text-to-speech task drastically improved the performance of ID3 and made it indistinguishable from backpropagation).

Non-trivial datasets The accuracy should not be too high after seeing a small number of instances. When C4.5 is run on the mushroom dataset or the shuttle dataset, the accuracy is over 99% after as little as 100 instances. The "odor" feature for the mushroom dataset, for example, predicts the class with over 98% accuracy.

Age We avoided using old datasets, such as chess, hypothyroid, and vote because some algorithms (*e.g.*, C4.5) were developed over these datasets and there is a chance that the algorithms are well tuned to these datasets. Since these datasets were used over and over for years, researchers seeing the test-set accuracies are indirectly overfitting these datasets by tuning their parameters. This problem is one of the main concerns people raise with the use of UC Irvine datasets.

[3]This is the main reason neural-networks outperformed decision trees and rules in Schaffer (1993). For the neural-networks, Schaffer encoded unknowns as a separate value. If we correct the encoding, C4.5's accuracy is equivalent to or better than that of the neural-networks.

We chose to experiment with the following real-world domains:

Breast cancer Wisconsin There are 699 instances collected from Dr. Wolberg's clinical cases at the University of Wisconsin (Wolberg & Mangasarian 1990, Zhang 1992*a*). These were collected over a period of two and a half years, and the problem is to determine whether the tumors were benign or malignant based on data for each cancer patient. There are ten features, where one is the serial number, and nine are identifying characteristics: clump thickness, uniformity of cell size, uniformity of cell shape, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli, and mitosis. The features are all linearly ordered but discrete in the range (1-10) and hence defined as "continuous" (see Section 2.1 on page 16). There are 16 missing values, all in the bare nuclei feature.

Cleveland heart disease (cleve) There are 303 instances from Dr. Detrano. The task is to distinguish the presence or absence of heart disease in patients. There are seven nominal features and six continuous. The features include: age, sex, chest pain type, cholesterol, fasting blood sugar, resting ECG, max heart rate, *etc.*

Australian credit screening (crx) There are 690 instances from an Australian credit company. The task is to determine whether to give a credit card to an applicant. The features have been coded to preserve confidentiality. The dataset was first used in Quinlan (1987). There are six continuous features and nine nominal ones.

DNA There are 3,186 instances in the StatLog version of the DNA dataset we used (Taylor *et al.* 1994). The domain is drawn from the field of molecular biology. Splice junctions are points on a DNA sequence at which "superfluous" DNA is removed during protein creation. The task is to recognize exon/intron boundaries, referred to as EI sites; intron/exon boundaries, referred to as IE sites; or neither. The IE borders are referred to as "acceptors" and the EI borders are "donors." The instances were taken from GenBank 64.1 (genbank.bio.net). The features provide a window of 60 nucleotides, each represented as 3 binary indicator features that represent the value a,c,g, or t, thus giving 180 binary features. The classification is the middle point of the window, thus providing 30 nucleotides at each side of the junction.

Horse colic There are 368 instances in this dataset created by Mary McLeish & Matt Cecile from the University of Guelph. The task is to determine whether a lesion is surgical. There are 22 features, of which seven are continuous. The features describe whether the horse had surgery, whether it is young or old, rectal temperature, pulse, respiratory rate, temperature of extremities, mucous membrane color, capillary refill time, pain (subjective judgment), *etc.*

Pima Indian diabetes (pima) There are 768 instances from the National Institute of Diabetes and Digestive and Kidney Diseases. The task is to determine whether the patient shows signs of diabetes according to World Health Organization criteria. All patients are females who live near Phoenix, Arizona. They are at least 21 years old and of Pima Indian heritage. There are eight continuous features: number of times pregnant, plasma glucose concentration, diastolic blood pressure, triceps skin fold thickness, 2-hour serum insulin, body mass index, diabetes pedigree function, and age.

Sick euthyroid There are 3163 instances from the Garavan institute in Sydney, Australia. The task is to identify a patient as having thyroid disease. There are 18 nominal features and 7 continuous ones, including: age, sex, on-thyroxine, pregnant, sick, tumor, TSH-measured, TSH value, *etc.*

Soybean (large) There are 683 instances in this dataset. The task is to diagnose soybean diseases. There are 19 classes and 35 discrete features describing leaf properties and various abnormalities.

We chose to experiment with the following artificial domains:

Monk's problems The Monk's problems are three artificial problems that allow comparison of algorithms. In Thrun *et al.* (1991), 24 authors have compared 25 machine learning algorithms on three artificial problems. In the given domain, robots have six different nominal features as follows:

$$\begin{aligned}
\text{Head-shape} &\in \{\text{round, square, octagon}\},\\
\text{Body-shape} &\in \{\text{round, square, octagon}\},\\
\text{Is-smiling} &\in \{\text{yes, no}\},\\
\text{Holding} &\in \{\text{sword, balloon, flag}\},\\
\text{Jacket-color} &\in \{\text{red, yellow, green, blue}\},\\
\text{Has-tie} &\in \{\text{yes, no}\}.
\end{aligned}$$

We now describe the three problems:

Monk1 The standard training set contains 124 instances from the following target concept:

(head-shape = body-shape) or (jacket-color = red)

This problem is hard because of the interaction between the first two features (head shape and body shape) and because a multi-way split that is usually done by C4.5 fragments the data if a split is done on jacket-color. If the instances are projected on the three relevant features, 35 out of the 36 possible instances are given in the training set, making the problem easy. However, many induction algorithms do not do a good job at selecting the correct features.

Monk2, Monk2-local The standard training set contains 169 instances from the following target concept:

Exactly two of the features have their *first* value.

This second problem has no irrelevant features which makes it a very hard problem in the original encoding using six nominal features. However, if we encode it using a **local representation,** where each feature value is represented by one Boolean indicator feature, many features become irrelevant. Thrun and Fahlman used this encoding scheme when neural nets were tested, and this encoding will be termed **Monk2-local** in this dissertation. Under a local-representation encoding, there are 17 features, but 11 of them are irrelevant.

Monk3 The standard training set contains 122 instances from the following target concept:

(jacket-color = green and holding = sword) or
(jacket-color $\neq$ blue and body-shape $\neq$ octagon)

In the standard training set, 5% of the instances have their label reversed, and it is thus the only Monk problem that is not noise-free. It is also important to observe that 97.2% accuracy is achievable using only the second disjunction, *i.e.*, jacket-color $\neq$ blue and body-shape $\neq$ octagon.

Corral There are 32 instances in this Boolean dataset, which contains six features: $A0$, $A1$, $B0$, $B1$, "irrelevant," and "correlated." The target concept is

$$(A0 \wedge A1) \vee (B0 \wedge B1) .$$

The feature named "irrelevant" is uniformly random, and the feature named "correlated" matches the class label 75% of the time (for specific instances). Greedy strategies for building decision trees pick the "correlated" feature as it seems best by all known selection criteria. After the wrong root split, the instances are fragmented and there is not enough data at each subtree to describe the correct concept. Figure 2.1 shows the decision tree induced by C4.5. CART induces a similar decision trees with the "correlated" feature at the root. When this feature is removed, the correct tree is found as can be seen in Figure 2.2.

As mentioned in Section 2.2 on page 19, the version of ID3 that we use is basically C4.5 with pruning turned off. Because C4.5 executes a post-processing step that replaces a node by its child if it is considered more accurate, the root node in Corral is indeed replaced by the correct child when pruning is turned off, leading to a perfect tree. Even though ID3 does well on this dataset, it is a dataset where selecting the correct set of relevant features is hard: the "correlated" feature is selected by almost all greedy methods.

***m*-of-*n*-3-7-10** There are 300 training instances in this Boolean dataset. The target is that at least three bits of bits numbered three to nine are set to one (bits one, two, and ten are irrelevant). Such target concepts are common in medical domains where a patient needs to exhibit at least m of a set of n symptoms to be diagnosed with some disease (Spackman 1988). Towell & Shavlik (1993) showed the representation of neural networks closely resembles m-of-n concepts. This concept is extremely hard in terms of feature interactions. Seven features interact here, and most algorithms are unable to identify the correct features.

Table 2.2 provides a summary of the characteristics of the datasets. Small datasets were tested using ten-fold cross-validation; artificial datasets and large datasets were split into train and test sets (the artificial datasets have a well defined training set, as does the DNA dataset form StatLog). The baseline accuracy is the accuracy (on the whole dataset) when predicting the majority class.

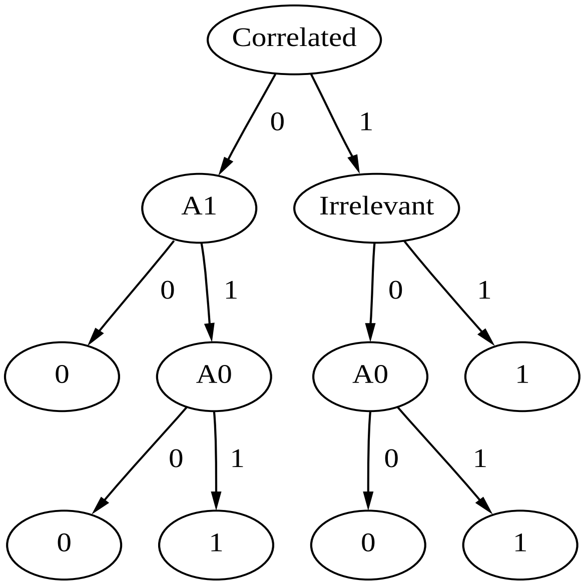

Figure 2.1: The "Corral" datasets fools top-down decision-tree algorithms into picking the "correlated" feature for the root, causing fragmentation, which in turns causes the irrelevant feature to be chosen. This tree was induced by C4.5 with the default parameters.

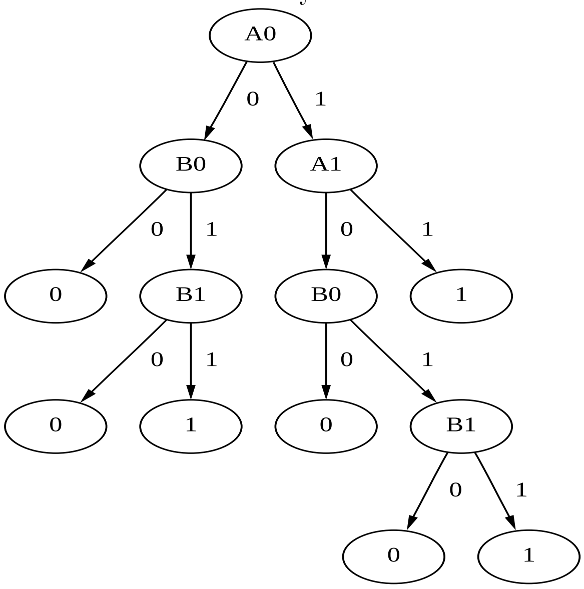

Figure 2.2: The correct tree induced by C4.5 when the "irrelevant" and "correlated" features are removed.

Table 2.2: Summary of datasets. Datasets above the horizontal line are "real" and those below are artificial. CV indicates ten-fold cross-validation.

no.	Dataset	Features			no.	Train	Test	baseline
		all	nominal	cont	classes	size	size	accuracy
1	breast cancer	10	0	10	2	699	CV	65.52
2	cleve	13	7	6	2	303	CV	54.46
3	crx	15	9	6	2	690	CV	55.51
4	DNA	180	180	0	3	2000	1186	51.91
5	horse-colic	22	15	7	2	368	CV	63.04
6	Pima	8	0	8	2	768	CV	65.10
7	sick-euthyroid	25	18	7	2	2108	1055	90.74
8	soybean-large	35	35	0	19	683	CV	13.47
9	corral	6	6	0	2	32	128	56.25
10	m-of-n-3-7-10	10	10	0	2	300	1024	77.34
11	Monk1	6	6	0	2	124	432	50.00
12	Monk2-local	17	17	0	2	169	432	67.13
13	Monk2	6	6	0	2	169	432	67.13
14	Monk3	6	6	0	2	122	432	52.78

2.5 The Bias-Variance Tradeoff

How do we tell a kid that not everything the computer says is right? Most of us know better, but we still have a long-standing trust in computers—they don't make mistakes, they don't have biases, they don't lie or cheat.
—Clifford Stoll, Silicon Snake Oil, 1994

The **bias** of a method that estimates a parameter θ is defined as the expected estimated value ($E[\hat{\theta}]$) minus the value of θ:

$$\text{Bias} = E[\hat{\theta}] - \theta \ .$$

An unbiased estimation method is a method that has zero bias.

The **variance** of a method is the statistical variance of the estimates:

$$\text{Var} = E\left[\left(\hat{\theta} - E[\hat{\theta}]\right)^2\right]$$

Given a regression problem, where the label y is real-valued, we can measure the effectiveness of a method $\mathcal{I}$ (an inducer) that predicts the label from an instance $\vec{x}$ as the squared difference between the prediction and the expected value of y at $\vec{x}$. If $\mathcal{D}$ is the input to the inducer (training set), then the effectiveness, or squared error, of the inducer at $\vec{x}$ is:

$$(\mathcal{I}(\mathcal{D},\vec{x}) - E[y \mid \vec{x}])^2$$

where the expectation is taken over D, the probability distribution over the labelled instance space. Taking expectation with respect to the training set $\mathcal{D}$ (*i.e.*, averaging over all possible training sets of the given size), yields

$$E_{\mathcal{D}}\left[(\mathcal{I}(\mathcal{D},\vec{x}) - E[y \mid \vec{x}])^2\right],$$

which can be decomposed as follows (Geman, Bienenstock & Doursat 1992):

$$\begin{aligned}
E_{\mathcal{D}} & \left[\left(\mathcal{I}(\mathcal{D},\vec{x}) - E[y \mid \vec{x}]\right)^2\right] \\
& \qquad\qquad\qquad\qquad \text{Subtract and add} E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] \\
&= E_{\mathcal{D}}\left[\left((\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right]) + (E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E[y \mid \vec{x}])\right)^2\right] \\
& \qquad\qquad\qquad\qquad (a+b)^2 = a^2 + b^2 + 2ab \\
&= E_{\mathcal{D}}\left[\left(\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right]\right)^2\right] + E_{\mathcal{D}}\left[\left(E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E[y \mid \vec{x}]\right)^2\right] + \\
& \quad 2E_{\mathcal{D}}\left[(\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right])\,(E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E[y \mid \vec{x}])\right] \\
& \qquad\qquad\qquad\qquad E[c] = c \text{ for constant c} \\
&= E_{\mathcal{D}}\left[\left(\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right]\right)^2\right] + \left(E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E[y \mid \vec{x}]\right)^2 + \\
& \quad 2E_{\mathcal{D}}\left[(\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right])\right] \cdot (E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E[y \mid \vec{x}]) \\
& \quad \text{now } E_{\mathcal{D}}\left[(\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right])\right] = E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] = 0 \\
&= E_{\mathcal{D}}\left[\left(\mathcal{I}(\mathcal{D},\vec{x}) - E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right]\right)^2\right] + \left(E_{\mathcal{D}}\left[\mathcal{I}(\mathcal{D},\vec{x})\right] - E[y \mid \vec{x}]\right)^2 \\
&= \text{“variance”} + \text{“bias”}
\end{aligned}$$

If, on average, $\mathcal{I}(\mathcal{D}, \vec{x})$ is different from $E[y \mid \vec{x}]$, then the inducer is a biased estimator of $E[y \mid \vec{x}]$. (An estimator may be biased for one class of target concepts and not for others.) It may be that an estimator is unbiased, yet it has high variance, *i.e.*, when the predictions of classifier $\mathcal{I}(\mathcal{D})$ may be very different from their expected value. In these cases, the inducer has high variance, and this may be the dominant term contributing to the overall squared error.

The main problem with the bias-variance decomposition, as shown, is that is applies only to the squared-error loss function. There are no known equivalent formulations for the zero-one loss function, which is more commonly used in experimental machine learning. Wolpert (1995) has recently proposed using log-loss for categorical outputs, and has shown a similar decomposition, where the bias is the Kullback-Leibler distance between the probabilities assigned to the labels by the target concept and the expected probabilities assigned by the inducer.

2.6 Experimental Methodology

> *If we take in our hand any volume; of divinity or school metaphysics, for instance; let us ask,* Does it contain any abstract reasoning concerning quantity or number? *No.* Does it contain any experimental reasoning, concerning matter of fact and existence? *No. Commit it then to the flames: for it can contain nothing but sophistry and illusion.*
> —*David Hume, An Enquiry Concerning Human Understanding, 1748*

Except for Chapter 3, where the experimental methodology is different (and explained there), the experiments in this dissertation compare runs of two or more algorithms on the datasets described above. For datasets that are large enough and for artificial data where there is a well defined training and test set, we report the accuracy on the test set, and the theoretical standard deviation as explained in Section 3.2.2 on page 39. For smaller datasets, we execute ten-fold cross-validation (described in Section 3.2.3 on page 41) and report the mean accuracy and the standard deviation of the folds.

To determine whether the difference between two algorithms is significant or not, we report the p-values, which indicate the probability that one algorithm is better than the other, where the variance of the test is the average variance of the two algorithms and a normal distribution is assumed. A more powerful test would have been to conduct a paired t-test for each instance tested, or for each fold, but the overall picture would not change much.

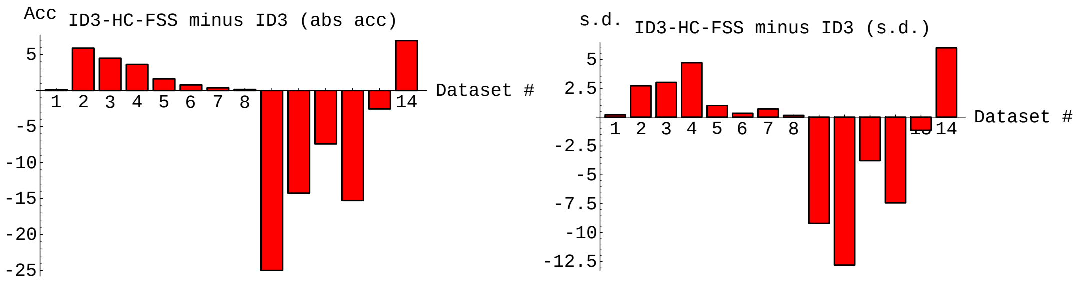

Figure 2.3: Absolute difference (ID3-HC-FSS minus ID3) in accuracy (left) and in std-devs (right).

Whenever we compare two or more algorithms, A_1 and A_2, we give the table of accuracies, and show two bar graphs. One bar graph shows the absolute difference, $A_2 - A_1$, in accuracies and the second bar graph shows the accuracy difference *divided by the standard deviation* of the accuracy, *i.e.*, $(A_2 - A_1)/\text{std-dev}$. The second bar graph shows the accuracy difference in different units—standard deviations—so we can easily tell which results are significant and by how much. For example, Figure 2.3 shows two bar graphs, one showing the absolute difference in accuracy, and one showing the difference in standard deviations. Corresponding bars (same number) will always be in the same direction from the origin, but their height might be different (*e.g.*, datasets 4 and 9).

Comparisons will generally be made such that A_2 is the algorithm proposed just prior to the comparison (the "new" algorithm) and A_1 is either a standard algorithm, such as C4.5, or the previous proposed algorithm. When the bar is above zero A_2, the proposed algorithm, outperforms A_1, which we are comparing with.

> When two bar graphs are shown, the left one measures the absolute difference in accuracy and the right one measures the accuracy difference in standard-deviations (s.d.). When the length of the bars on the standard-deviation chart are higher than two, the results are significant at the 95% confidence level.

When we mention the amount of time it took to run the algorithms, these are reported in CPU units (hours, minutes, seconds) on a Sun Sparc 10 for a single train-test sequence.

Chapter 3

Accuracy Estimation

FORECAST: A prediction of the future, based on the past, for which the forecaster demands payment in the present.
—Unix fortune

Estimating the accuracy of a classifier induced by a supervised learning algorithm is important not only in order to predict its future performance, but also in order to choose a classifier from a given set (model selection), or in order to combine classifiers. Although the study of accuracy estimation was originally motivated by the fact that the wrapper approach (Chapter 4) requires estimating the accuracy of hundreds of classifiers, the results and observations are not limited to use in wrappers and could be of general use.

We review accuracy estimation methods, compare cross-validation and the .632 bootstrap, and look at methods to stabilize the estimate of cross-validation. We report on a large-scale experiment—over a million runs of C4.5 and a Naive-Bayes algorithm—to estimate the effects of different variants of cross-validation on real-world datasets. For the .632 bootstrap, we vary the number of samples; for cross-validation, we vary the number of folds, the number of times the folds are generated, stratification, and trimming. The number of times cross-validation is executed has a large effect on the variance of k-fold cross-validation for k less than ten; in fact, the variance decreases as k decreases if repeated cross-validation is done; this result contrasts with the U-shape curved when a single cross-validation is done. We conclude that for model selection and datasets similar to ours, it is best to use k-fold cross-validation for low values of k (*i.e.*, $k \leq 10$), even if computation power allows using more folds (*e.g.*, leave-one-out).

3.1 Introduction

Today's data analyst can afford to expend more computation on a single problem than the world's yearly total of statistical computation in the 1920s.
—Efron & Tibshirani (1991)

Estimating the accuracy of a classifier induced by a supervised learning algorithm is important not only in order to predict its future performance, but also in order to choose a classifier from a given set (model selection) as in Schaffer (1993), or in order to combine classifiers (Wolpert 1992*b*, Breiman 1994*a*). For estimating the final accuracy of a classifier, we would like an estimation method with low bias and low variance. To choose a classifier or to combine classifiers, the absolute accuracies are less important, and we are willing to trade off bias for low variance, assuming the bias affects all classifiers similarly (*e.g.*, estimates are 5% pessimistic).

Computer power has grown to a point where computer intensive methods for accuracy estimation are used more often and on larger datasets. Methods such as cross-validation and bootstrap require parameters that determine the quality of the estimates. In this chapter, we explain some of the assumptions made by the different estimation methods and present concrete examples where each method fails. While it is known that no accuracy estimation can be correct all the time (Wolpert 1994*b*, Schaffer 1994), we are interested in identifying methods that are well suited for the biases and trends in typical real world datasets.

For years it was generally assumed that higher folds for cross-validation (up to leave-one-out) would yield better estimates, usually at the expense of longer computation time. For example, Weiss & Kulikowski (1991) write that "While leaving-one-out is a preferred technique, with large sample sizes it may be computationally quite expensive." Mosteller & Tukey (1968) wrote the following: "Suppose that we set aside one individual case, optimize for what is left, then test on the set-aside case. Repeating this for every case squeezes the data almost dry."

The use of incremental induction algorithms, however, allows cross-validation in time that is independent of the number of folds (Kohavi 1995*a*, Moore & Lee 1994, Utgoff 1994). With such algorithms, leave-one-out takes exactly the same time as ten-fold cross-validation; is it clear that leave-one-out should be preferred? While leave-one-out is almost unbiased (Lachenbruch 1967, Glick 1978, Efron 1983), it has high variance, leading to unreliable estimates. Recent results, both theoretical and experimental, have shown that it is not always the case that increasing the number of folds is beneficial, especially if the variance is more important than the bias, as is the case with model selection.

On the theoretical side, using leave-one-out cross-validation for model selection of linear models is asymptotically inconsistent: the probability of selecting the model with the best predictive power does not converge to one as the total number of observations approaches infinity (Zhang 1992*b*, Shao 1993). On the experimental side, Breiman & Spector (1992) and Kohavi (1995*a*) found that five-fold and ten-fold cross-validation are better for model selection of feature subsets than leave-one-out.

In this chapter, we assess the different effects that parameters have on the bias and variance of the two best-known accuracy estimators: the .632 bootstrap and cross-validation. For the .632 bootstrap, we vary the number of samples; for cross-validation, we vary the number of folds, the number of times cross-validation is executed, and whether or not a trimmed mean is used.

This chapter is organized as follows. In Section 3.2, we describe the common accuracy estimation methods and ways of computing confidence bounds. In Section 3.3, we discuss the methodology underlying our experiments. In Section 3.4, we evaluate the bias and variance of cross-validation and the .632 bootstrap and in Section 3.5, we evaluate cross-validation variants in an attempt to stabilize the accuracy estimates. In Section 3.6, we describe a technique to extrapolate the learning curve in order to reduce the bias. In Section 3.7, we discuss related work and we conclude with a discussion of future work and a summary in Sections 3.8 and 3.9.

3.2 Methods for Accuracy Estimation

The term assessment *is preferred to* validation *which has a ring of excessive confidence about it.*
—Stone (1974)

Given a finite dataset, we would like to *estimate* the future performance of a classifier induced by the given inducer and dataset. A single accuracy estimate is usually meaningless without a confidence interval; thus we will consider how to approximate such an interval when possible. An excellent review of accuracy estimation (not including bootstrap) in the context of nearest-neighbor algorithms can be found in Devijver & Kittler (1982, Chapter 10). McLachlan (1992, Chapter 10) provides a review of more recent methods from a statistical perspective.

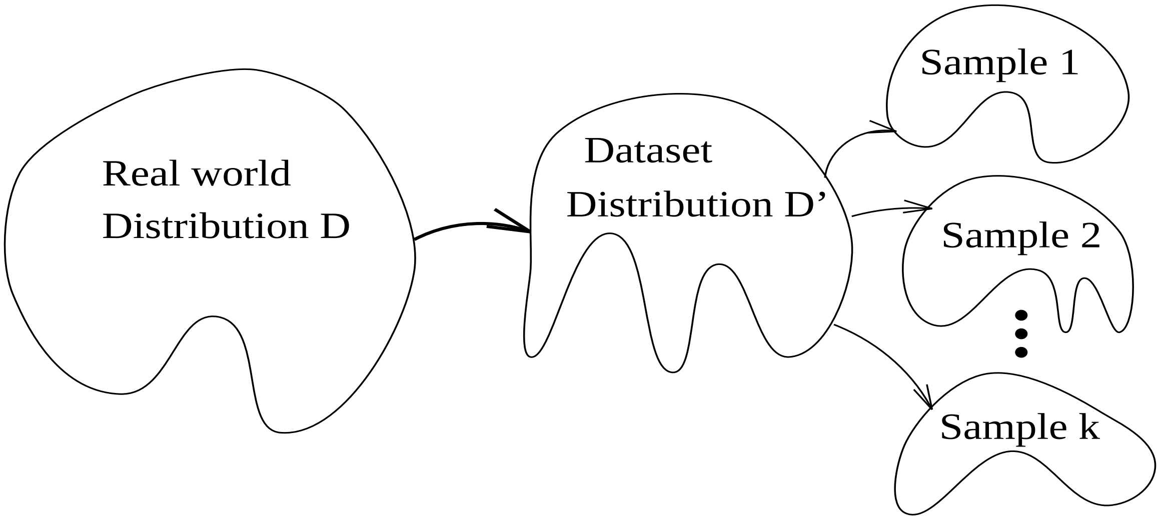

Figure 3.1: Accuracy estimation techniques, such as holdout, cross-validation, and bootstrap, are all based on the idea of resampling.

Except for the resubstitution estimate, all the non-parametric estimators that will be explained are based on the idea of *resampling*, depicted in Figure 3.1. The real world has some unknown distribution D. Our dataset has distribution D', which is assumed to be similar to the distribution D. In order to estimate the accuracy of inducers trained based on D', we create samples from D', train on these samples, and test on instances from D' (usually out-of-sample instances). We thus simulate the sampling process that occurred in the real world, assuming that D' *is* the real world.

3.2.1 Resubstitution Estimate

All the present recommendations are predicated on small-to-moderate training sample sizes, perhaps in the range of 10-50. ...for large samples, there is no need to look further than the resubstitution estimator when seeking a robust method.

—Knoke (1986)

The resubstitution estimate of the accuracy, sometimes called apparent accuracy, tests the classifier on the same data given to the inducer. Formally, we have

$$\mathtt{acc}_s = \frac{1}{m} \sum_{\langle \vec{x}_i, y_i \rangle \in \mathcal{D}} \delta(\mathcal{I}(\mathcal{D}, \vec{x}_i), y_i) \tag{3.1}$$

where $\delta(i,j) = 1$ if $i = j$ and 0 otherwise.

The resubstitution estimate is a highly optimistic estimate of accuracy because classification procedures attempt to minimize it. For restricted hypothesis spaces, *e.g.*, linear discrimination, where it is usually hard to fit large amounts of data, the resubstitution estimate is reasonable. For example, Knoke (1986) (quoted above) wrote that for large samples, the resubstitution estimate is the best accuracy estimation to use, but he assumes that the classification is based on linear discriminant functions. For many induction algorithms that perfectly fit the data, such as one nearest-neighbor or decision tree induction algorithms that do not prune, the resubstitution estimate is very optimistic; if there are no conflicting instances, the accuracy estimate will be 100%.

3.2.2 Holdout

The holdout method, sometimes called test sample estimation, randomly partitions the data into two mutually exclusive subsets called the training set and the test set, or holdout set. It is common to designate 2/3 of the data as the training set and the remaining 1/3 as the test set. The training set is given to the inducer, and the induced classifier is tested on the test set. Formally, let $\mathcal{D}_h$, the holdout set, be a subset of $\mathcal{D}$ of size h, and let $\mathcal{D}_t$ be $\mathcal{D} \setminus \mathcal{D}_h$. The holdout estimated accuracy is defined as

$$\texttt{acc}_h = \frac{1}{h} \sum_{\langle \vec{x}_i, y_i \rangle \in \mathcal{D}_h} \delta(\mathcal{I}(\mathcal{D}_t, \vec{x}_i), y_i) \ , \tag{3.2}$$

where $\delta(i,j) = 1$ if $i = j$ and 0 otherwise. Assuming that the inducer's accuracy increases as more instances are seen, the holdout method is a pessimistic estimator because only a portion of the sample is given to the inducer for training. The more instances we leave for the test set, the higher the bias of our estimate; however, fewer test set instances will cause the confidence interval for the accuracy to be wide, as shown below.

The classification of each test instance can be viewed as a Bernoulli trial: either correct or incorrect labelling. Let S be the number of correct classifications on the test set, then S is distributed binomially (sum of Bernoulli trials). For reasonably large holdout sets, the distribution of S/h is approximately normal with mean $\texttt{acc}$ (the true accuracy of the classifier) and a variance of $\texttt{acc} * (1 - \texttt{acc})/h$. Thus, by applying the De Moivre-Laplace limit theorem, we have

$$\Pr\left\{-z < \frac{\mathsf{acc}_h - \mathsf{acc}}{\sqrt{\mathsf{acc}(1-\mathsf{acc})/h}} < z\right\} \approx \gamma \;, \tag{3.3}$$

where z is the $(1+\gamma)/2$-th quantile point of the standard normal distribution. To get a 100γ percent confidence interval, one determines z and inverts the inequalities.

Inversion of the inequalities leads to a quadratic equation in acc, the roots of which are the low and high confidence points:

$$\frac{2h \cdot \mathsf{acc}_h + z^2 \pm z \cdot \sqrt{4h \cdot \mathsf{acc}_h + z^2 - 4h \cdot \mathsf{acc}_h^2}}{2(h+z^2)} \;. \tag{3.4}$$

The above equation is not conditioned on the dataset $\mathcal{D}$; if background information is available about the probability of the given dataset, it must be taken into account.

A simpler expression can be derived by the plug-in-estimate acc_h for acc in the denominator of Equation 3.3, yielding the following low and high confidence points:

$$\mathsf{acc}_h \pm z \cdot \sqrt{\frac{\mathsf{acc}_h(1-\mathsf{acc}_h)}{h}} \;. \tag{3.5}$$

This simpler approximation is inaccurate for small values of h and for extreme accuracies. For example, if the estimated accuracy is 100%, a zero width confidence interval is obtained for any value of z when using this approximation.

The holdout estimate is a random number that depends on the division into a training set and a test set. In **random subsampling**, the holdout method is repeated k times, for different random partitions, and the estimated accuracy is derived by averaging the estimated holdout accuracies. The standard deviation of the accuracy can be estimated as the standard deviation of the accuracy estimations from each holdout run. Note, however, that one cannot compute the standard deviation of the mean by dividing the population mean by $\sqrt{k}$ because the estimates are neither independent nor approximately so (the same test instances are used multiple times). Regrettably, many researchers in the field claim the significance of their results by increasing k until the difference, as small as it it, is made significant. To compute the variance, we recommend using the percentile method (Efron & Tibshirani 1993, pp. 168-176). A $1-2\alpha$ confidence interval is defined by taking the α and $1-\alpha$ quantiles of the estimates. This procedure requires that k be large, say at least 50, so that there is enough data to estimate the 5% mass in the tails if a 95% confidence interval is sought.

The main assumption that is violated in random subsampling is the independence of instances in the test set from those in the training set. Under repeated samplings, the union of the training set and test set is constrained to be equal to the given dataset. If the training set and test set are formed by a split of an original dataset, then an over-represented class in one subset will be under-represented in the other. To demonstrate the issue, we simulated a 2/3, 1/3 split of Fisher's famous iris dataset (Fisher 1936) and used a majority inducer that builds a classifier predicting the prevalent class in the training set. The iris dataset describes iris plants using four continuous features, and the task is to classify each instance (an iris) as Iris Setosa, Iris Versicolour, or Iris Virginica. For each class label, there are exactly one third of the instances with that label (50 instances of each class from a total of 150 instances); thus we expect 33.3% prediction accuracy. However, because the test set will always contain less than 1/3 of the instances of the class that was prevalent in the training set, the accuracy predicted by the holdout method is 27.68% with a standard deviation of 0.13% (estimated by averaging 500 holdouts).

In practice, the dataset size is always finite, and usually smaller than we would like it to be. The holdout method makes inefficient use of the data: a third of the dataset is not used for training the inducer. The next method mitigates the problem of hiding a large portion of the dataset from the algorithm.

3.2.3 Cross-Validation, Leave-one-out, and Stratification

Since the basic philosophy of cross-validation is non-probabilistic and non-parametric, it is perhaps not surprising that supporting theory is rather meagre.
—*Stone (1978)*

In k-fold cross-validation, sometimes called rotation estimation, the dataset $\mathcal{D}$ is randomly split into k mutually exclusive subsets (the folds) $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_k$ of approximately equal size. The inducer is trained and tested k times; each time $t \in \{1, 2, \ldots, k\}$, it is trained on $\mathcal{D} \setminus \mathcal{D}_t$ and tested on $\mathcal{D}_t$.[1] The cross-validation estimate of accuracy is the overall number of correct classifications, divided by the number of instances in the dataset. Formally, let $\mathcal{D}_{(i)}$ be the test set that includes instance $\langle \vec{x}_i, y_i \rangle$, then the cross-validation estimate of accuracy

[1]The notation $A \setminus B$ indicates set A minus set B.

is

$$\mathtt{acc}_{\mathrm{CV}} = \frac{1}{m} \sum_{\langle \vec{x}_i, y_i \rangle \in \mathcal{D}} \delta(\mathcal{I}(\mathcal{D} \setminus \mathcal{D}_{(i)}, \vec{x}_i), y_i) \ . \tag{3.6}$$

The cross-validation estimate is a random number that depends on the division into folds. **Complete cross-validation** is the average of all $\binom{m}{m/k}$ possibilities for choosing m/k instances out of m, but it is usually too expensive (Geisser 1975). Except for leave-one-out (m-fold cross-validation), which is always complete, k-fold cross-validation is estimating complete k-fold cross-validation using a single split of the data into the folds. Repeating cross-validation multiple times using different splits into folds provides a better Monte-Carlo estimate to the complete cross-validation at an added cost.

Stone (1974) is credited with the first formal description of cross-validation and its uses for choosing statistical predictors; however, he credited Mosteller, Wallace, and Lachenbruch for the idea. Lachenbruch (1967) described leave-one-out as a method for obtaining confidence intervals. Linhart & Zucchini (1986) claimed that the idea was explicitly stated in psychometric literature in the 1930s.

In **stratified cross-validation**, the folds are stratified so that they contain approximately the same proportions of labels as the original dataset. Breiman *et al.* (1984, Section 8.7) claimed that, especially for regression, stratification is the preferred method for selecting the right-sized tree; the cross-validation used in CART and in the C4.5 cross-validation utility, xval.sh, is stratified; and Weiss, who has done much work in accuracy estimation, uses stratified cross-validation in his experiments (Weiss 1991, Weiss & Indurkhya 1994*a*, Weiss & Indurkhya 1994*b*). An asymptotic ($m \to \infty$) theoretical analysis done by Olshen, Gilpin, Henning, LeWinter, Collins & Ross (1985) showed that if the label is independent of the features (*i.e.*, random), then for building decision trees by recursive-partitioning, stratification helps a single-fold of cross-validation, in the sense that the mean-squared error is smaller. A later theoretical analysis done by Bai (1988) showed that under some assumptions (finite instance space, unequal class probabilities), stratification neither helps nor hurts two-fold cross-validation in terms of mean squared error because it increases the covariance between the folds.

The cross-validation accuracy is sometimes defined as the average of the estimated accuracies from the k runs, and not as in Equation 3.6. This estimation method is accurate for all but very small samples, and it is the version we use in our experiments because it allows us to test the effect of trimming and the normality assumption (see Section 3.5).

The estimate is not a good approximator for small samples because if one does ten-fold cross-validation for 15 instances, ten test sets will be created, five having one instance and five having two instances. While Equation 3.6 gives equal weight to each test instance, an average of the folds will double the weight given to the instances in the five folds containing a single instance.

We now attempt to understand when the cross-validation estimate is justified. A **perturbation** to a dataset is the deletion or addition of an instance to the dataset. An inducer is **stable** for a given dataset and a set of perturbations, if it induces classifiers that make the same predictions when it is given the perturbed datasets. This definition of stability does *not* require that the instances have to be smooth under some metric, nor is it required that the inducer be stable for all datasets; what is important is that the combination of the inducer and the dataset is stable under the perturbations we conduct. An inducer may be biased to find parity-like concepts in Boolean domains, and will induce the correct classifiers, even though neither the data, nor the concept is smooth. A majority inducer is a very stable inducer, *except* when it is given data where the differences in prevalences are small.

Proposition 1 (Variance in k-fold CV)
Given a dataset and an inducer. If the inducer is stable under the perturbations that delete the instances for the folds in k-fold cross-validation, the cross-validation estimate will be unbiased and the variance of the estimated accuracy will be approximately $acc_{cv} \cdot (1 - acc_{cv})/m$*, where m is the number of instances in the dataset.*

Proof: If we assume that the k classifiers produced make the same predictions, then the estimated accuracy has a binomial distribution with m trials and a probability of success equal to the accuracy of the classifier. ∎

A confidence interval may be computed using either Equation 3.4 or Equation 3.5 on page 40 with h equal to m, the number of instances. Breiman *et al.* (1984, Chapter 3) used this method to derive the standard deviation of the accuracy estimates, which was then used in the "1 SE rule" for inducing decision trees.

Note that the inducer must not have been chosen using an accuracy estimation method. Such a choice violates the assumption that the inducer was determined in advance and given access only to a training set. By choosing an inducer based on the accuracy estimation, we are in effect giving the inducer indirect access to the (internal) test set.

In reality, a complex inducer is unlikely to be stable for large perturbations, unless it has reached its maximal learning capacity. We expect the perturbations induced by leave-one-out to be small, and therefore, the classifier should be very stable. As we increase the size of the perturbations, stability is less likely to hold: we expect stability to hold more in 20-fold cross-validation than in ten-fold cross-validation, and both should be more stable than a holdout of 1/3. The proposition does not apply to the resubstitution estimate because it requires the inducer to be stable when no instances are given in the dataset.

Proposition 1 helps us understand one possible assumption that is made when using cross-validation, namely stability. If an inducer is unstable for a particular dataset under a set of perturbations introduced by cross-validation, the accuracy estimate is likely to be unreliable. If the inducer is almost stable on a given dataset, we should expect a reliable estimate. The next corollary takes the idea slightly further and shows a result that we have observed empirically: there is very little change in the variance of the cross-validation estimate when the number of folds is varied.

Corollary 2 (Variance in cross-validation)
Given a dataset and an inducer. If the inducer is stable under the perturbations caused by deleting the test instances for the folds in k-fold cross-validation for various values of k, then the variance of the estimates will be the same.

Proof: The variance of k-fold cross-validation in Proposition 1 does not depend on k. ∎

Breiman (1994*b*) discusses the instability of some induction algorithms. While some inducers are likely to be inherently more stable, the following example shows that one must also take into account the dataset and the actual perturbations.

Example 3.1 (Failure of leave-one-out)
Fisher's iris dataset (Fisher 1936) contains 50 instances of each class, leading one to expect that a majority inducer should have an accuracy of about 33%. However, the combination of this dataset with a majority inducer is unstable for the small perturbations performed by leave-one-out. When an instance is deleted from the dataset, its label is a minority in the training set; thus the majority inducer predicts one of the other two classes and always errs in classifying the test instance. The leave-one-out estimated accuracy for a majority inducer on the iris dataset is therefore 0%. Moreover, all folds have this estimated accuracy; thus the standard deviation of the folds is again 0%, giving the unjustified assurance that the

estimate is stable. Doing k-fold cross-validation for $k < m$ yield better estimates, but still not the 33.3% one expects. Averaging 100 runs of ten-fold cross-validation, an estimated accuracy of 21.57% with a standard deviation of 0.21% was obtained. See Section 3.2.5 on page 47 for more discussion on this problem. ∎

The example shows an inherent problem with cross-validation that applies to more than just a majority inducer. In a no-information dataset, where the label values are completely random, the best an induction algorithm can do is predict majority. Leave-one-out on such a dataset with 50% of the labels for each class and a majority inducer (the best possible inducer) would still predict 0% accuracy. ∎

3.2.4 Bootstrap

The bootstrap family was introduced by (Efron 1979) and is fully described in Efron & Tibshirani (1993). Mammen (1992) studies bootstrap for non-parametric curve estimation and linear models.

Given a dataset of size m, a **bootstrap sample** is created by sampling m instances uniformly from the data (with replacement). Since the dataset is sampled with replacement, the probability of any given instance not being chosen after m samples is $(1 - 1/m)^m \approx e^{-1} \approx 0.368$; the expected number of distinct instances from the original dataset appearing in the test set is thus $0.632m$.

The $\epsilon 0_i$ accuracy estimate is derived by using bootstrap sample i for training and the rest of the instances for testing. Given a number b, the number of bootstrap samples, let $\epsilon 0_i$ be the accuracy estimate for bootstrap sample i. The .632 bootstrap estimate is defined as

$$\mathsf{acc}_{\text{boot}} = \frac{1}{b} \sum_{i=1}^{b} (0.632 \cdot \epsilon 0_i + .368 \cdot \mathsf{acc}_s) \tag{3.7}$$

where acc_s is the resubstitution accuracy estimate on the full dataset (*i.e.*, the accuracy on the training set). Bootstrap has many variants not discussed here, including bias-corrected bootstrap where the bias of the bootstrap samples is assessed and added to the estimates, and parametric bootstrap that can take advantage of the background knowledge about the underlying distribution that generated the data. For non-parametric estimation, the .632 bootstrap was claimed to be the best (Efron 1983).

The variance of the estimate can be determined by computing the variance of the estimates for the samples. One of the main questions in using bootstrap is how many bootstrap samples to use. Efron & Tibshirani (1993) suggest that 50 to 200 samples should be enough for computing the variance. Because only .632 of the instances in the dataset are used for training, the $\epsilon 0$ bootstrap estimate is closely related to two-fold cross-validation and Efron suggests that 2-CV might be similarly corrected with the resubstitution accuracy (Efron 1983).

The assumption made by bootstrap is basically the same as that of cross-validation, *i.e.*, stability of the algorithm on the dataset: the "bootstrap world" should closely approximate the real world. In many cases, it is obvious that the bootstrap worlds are inappropriate. For example, if you try to estimate the number of duplicate instances, the bootstrap worlds will definitely over-estimate the statistics because approximately .368 of the original instances will be duplicates. Some machine learning algorithms ignore duplicate instances directly or indirectly. Nearest-neighbor algorithms (1-NN), for example, are unaffected by duplicates (except for rare cases of breaking ties), and Jain, Dubes & Chen (1987) wrote that "[Bootstrap .632 is] not appropriate for nearest-neighbor classifiers."

The .632 factor in Equation 3.7 on the page before is somewhat ad-hoc and can be justified only if the accuracy varies between the resubstitution and the $\epsilon 0_i$ accuracy linearly with the distance of the test instance from the closest instance in the training set (under some distance function!). The .632 factor is only a heuristic, as Efron (1983) writes that "The .632 estimator... has the weakest theoretical justification." This linearity assumption fails for inductions that are good memorizers, *i.e.*, inducers that produce classifiers, which are able to perfectly classify the training set.

Specifically, the .632 bootstrap fails to give the expected result when the classifier is a perfect memorizer (*e.g.*, an unpruned decision tree or a one nearest-neighbor classifier) and the label values are completely random, say with two classes. The resubstitution accuracy is 100%, and the $\epsilon 0$ accuracy is about 50%. Plugging these into the bootstrap formula, one gets an estimated accuracy of about 68.4%, far from the real accuracy of 50%.

Bootstrap can be shown to fail if we add a memorizer module to any given inducer and adjust its predictions. If the memorizer remembers the training set and makes the predictions when the test instance was a training instance, adjusting its predictions can make the resubstitution accuracy change from 0% to 100% and can thus bias the overall estimated accuracy in any direction we want.

3.2.5 Bias and Variance

The **bias** of a method that estimates a parameter θ is defined as the expected estimated value ($E[\hat{\theta}]$) minus the value of θ (see Section 2.5). As we have seen above, the .632 bootstrap estimate is biased for random data and for inducers that are perfect memorizers. In cross-validation, the induced classifier is tested on instances that are disjoint from the instances used for training, and it does not matter whether the training instances are remembered or not.

The following example shows that for random label values and a majority inducer, cross-validation is an unbiased estimator. While the example might seem obvious, the exact analysis is helpful in understanding the problems inherent in cross-validation. Specifically, the discussion that follows may help clarify the problem of large variance and how it is mitigated by using larger folds (k-fold cross-validation with $k \ll m$).

Example 3.2 (Leave-one-out is unbiased for random data and majority) Suppose the Boolean target concept is random, *i.e.*, the label is independent of the features and the probability of the label taking value 0 (or 1) is 0.5. We now analyze the leave-one-out estimated accuracy of a majority inducer (predicting the majority label on the training set independent of the input feature values). For simplicity, we will assume that our training set has m instances, where m is an even number. The mean estimated accuracy of leave-one-out in this example is:

$$E[\mathsf{acc}_{\mathrm{CV}}] = \sum_{\mathcal{D}} p(\mathcal{D})\mathsf{acc}_{\mathrm{CV}}(\mathcal{D}) \ .$$

Since the label is uniformly random, we can partition the sum according to X, the number of labels with value 1:

$$E[\mathsf{acc}_{\mathrm{CV}}] = \sum_{k=0}^{m} p(X = k)\mathsf{acc}_{\mathrm{CV}}(\mathcal{D}|X = k)$$

Because we assumed the training set has an even number of instances, the number of instances in each class will never be tied once we leave an instance out, and the leave-one-out estimated accuracy will be k/m if $k \geq m/2 + 1$ and $(m-k)/k$ if $k \leq m/2 - 1$ (k is the number of instances with label one). As shown in Example 3.1, the estimated accuracy for

$k = m/2$ will be zero. Our sum is hence:

$$
\begin{aligned}
E[\mathsf{acc}_{\mathrm{CV}}] &= \sum_{k=0}^{m/2-1} p(X=k)(m-k)/m + 0 + \sum_{k=m/2+1}^{m} p(X=k)k/m && (3.8)\\
&= 2\sum_{k=0}^{m/2-1} p(X=k)(m-k)/m \quad \text{Two terms are symmetric} && (3.9)\\
&= \frac{2}{m}\sum \binom{m}{k} 0.5^m (m-k) \quad \text{Binomial distribution} && (3.10)\\
&= \frac{2}{m}\sum \frac{m}{m-k}\binom{m-1}{m-k-1}(m-k)0.5^m && (3.11)\\
&= 2\sum \binom{m-1}{m-k-1} 0.5^m && (3.12)\\
&= 2\sum 0.5 \binom{m-1}{k} 0.5^{m-1} && (3.13)\\
&= \left(\sum_{k=0}^{m-1} \binom{m-1}{k} 0.5^{m-1}\right)/2 \quad \text{Add other half of binomial} && (3.14)\\
&= 0.5 \quad \blacksquare && (3.15)
\end{aligned}
$$

The example shows that cross-validation is unbiased, but it is important to analyze the estimates derived for a specific sample we might get as our training set. For any training set that is not exactly balanced, the leave-one-out estimate will be optimistically biased; the more skewed the sample is, the more biased the estimate will be. For exactly one type of training set—precisely balanced between the two classes—the estimate will be pessimistic and off by 50%.

In practice, one is given a single dataset and the accuracy of the induced classifier needs to be assessed. If the dataset were given from a random concept as above and a majority inducer is used, the estimate on a single dataset will be either slightly optimistic or terribly pessimistic, but never correct. More important, however, is the fact that the estimates have high variance: different datasets sampled from this distribution will give highly variable estimates.

If leave-one-out is replaced with k-fold cross-validation for $k \ll m$, then the internal folds might have majority label opposite to that of the complement, hence some estimates will be greater than 0.5 and others will be less than 0.5; the average accuracy is therefore much more likely to be closer to 0.5.

The folds, formed by k-fold cross-validation for a constant k (*e.g.*, ten-fold) and m much larger than k will have a sampling variance similar to the sampling variance for the whole dataset we have (Figure 3.1). When k is on the order of m, however, such as in the case of leave-one-out, we do not test the sensitivity of our results to sampling variations.

For model selection, the bias of the accuracy estimation is less crucial than the variance, assuming the bias is about the same for both models compared. For example, if the estimate is off by some 5%, then it will not affect the choice of model we make. As we will show later, the variance is much harder to deal with than the bias.

3.3 Methodology

The word "valid" would be better dropped from the statistical vocabulary. The only real validation of a statistical analysis, or of any scientific enquiry, is confirmation by independent observations.
—Anscombe (1967)

We now describe the experiments conducted in order to empirically compare the various accuracy estimators. We chose C4.5 and Naive-Bayes to conduct the experiments (see Section 2.2). Since we are not interested in the induction algorithms themselves, only in how well the accuracy estimators perform, we used a fast version of Naive-Bayes that assumes a Gaussian distribution for continuous features (as opposed to one that discretizes the continuous features). We hope that the different hypothesis spaces—decision trees for C4.5 and summary statistics for Naive-Bayes—are different enough so that our results should be of a general nature and apply to other induction algorithms. On real-world datasets, neither algorithm dominates the other; there are datasets for which C4.5 is more accurate and vice-versa.

The target concepts are unknown for real-world datasets, so we used the holdout method to estimate the quality of the cross-validation and bootstrap estimates. To choose a set of datasets, we looked at the learning curves for C4.5 and Naive-Bayes for most of the supervised classification datasets at the UC Irvine repository (Murphy & Aha 1995) that contained more than 500 instances (about 25 such datasets). To ensure little variance, we chose datasets with at least 500 instances for testing. Note that even with 500 instances for testing, a holdout estimate can have a standard deviation of $\sqrt{.5 * (1 - .5)/500} \approx 2.2\%$ if the accuracy is 50%. While the true accuracies of a real dataset cannot be computed

Dataset	no. of ftrs.	sample-size / total size	no. of categories	duplicate instances	C4.5	Naive-Bayes
breast cancer	10	50/699	2	8	91.37$\pm$0.10	94.22$\pm$0.10
chess	36	900/3196	2	0	98.19$\pm$0.03	86.80$\pm$0.07
hypothyroid	25	400/3163	2	77	98.52$\pm$0.03	97.63$\pm$0.02
mushroom	22	800/8124	2	0	99.36$\pm$0.02	94.54$\pm$0.03
soybean large	35	100/683	19	52	70.49$\pm$0.22	79.76$\pm$0.14
vehicle	18	100/846	4	0	60.11$\pm$0.16	46.80$\pm$0.16
rand	20	100/3000	2	9	49.96$\pm$0.04	49.90$\pm$0.04

Table 3.1: True accuracy estimates for the datasets using C4.5 and Naive-Bayes classifiers at the chosen sample sizes. The "Duplicate instances" column indicates the number of instances that are duplicates in the complete dataset.

because we do not know the target concept, we can estimate the true accuracies using the holdout method. The "true" accuracy estimates in Table 3.1 were computed by taking a random sample (without replacement) of the given size, computing the accuracy using the rest of the dataset as a test set, and repeating 500 times.[2]

We chose six datasets from a wide variety of domains, such that the learning curve for both algorithms did not flatten out too early, that is, before one hundred instances. We also added a *no information dataset*, Rand, with 20 Boolean features and a Boolean random label. Figure 3.2 shows the learning curves for the chosen datasets. Each data point is the mean accuracy of the classifier on the instances not used in the training set, averaged over 20 runs; the error bars indicate a 95% confidence interval for the mean.

On one dataset, vehicle, the generalization accuracy of the Naive-Bayes algorithm deteriorated by more than 4% as more instances were given. A similar phenomenon was observed on the shuttle dataset (not shown). Such a phenomenon was predicted by Schaffer and Wolpert (Schaffer 1994, Wolpert 1994*b*), but we were surprised that it was observed on two real-world datasets. Figure 3.3 shows a plot of the "circularity" feature versus label value two in the vehicle dataset. The distribution is approximately bimodal, and thus violates the normality assumption made by Naive-Bayes. As more instances are given to the inducer, the single mean—right between the two humps—is computed more accurately,

[2]An alternative view of this procedure is that the instances *define* the true distribution where each instance has probability $1/m$. In this case, our test sets represent the exact out-of-sample accuracy of each classifier.

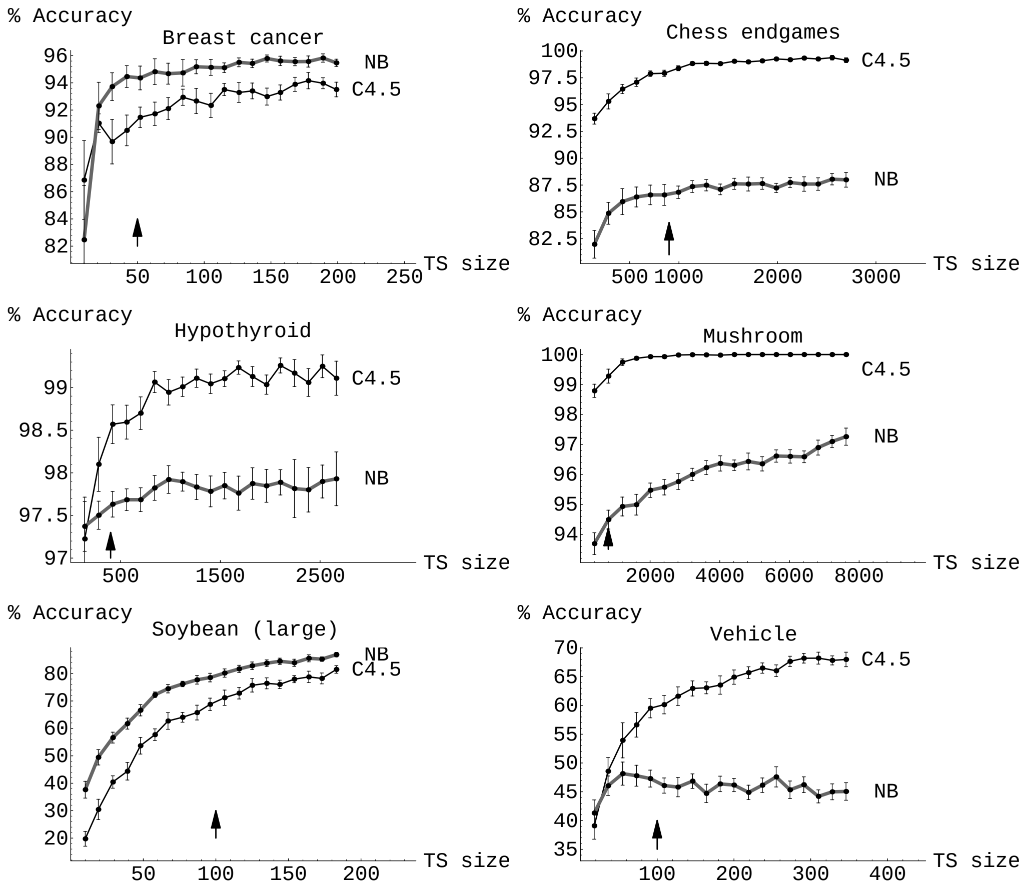

Figure 3.2: The learning curves for C4.5 and Naive-Bayes (NB). The x-axis shows the training set size and the y-axis shows the accuracy on the dataset instances not used during training. Each point is an average of 20 runs, with error bars indicating the 95% confidence intervals for the mean accuracy. Arrows show the points we chose for accuracy estimation.

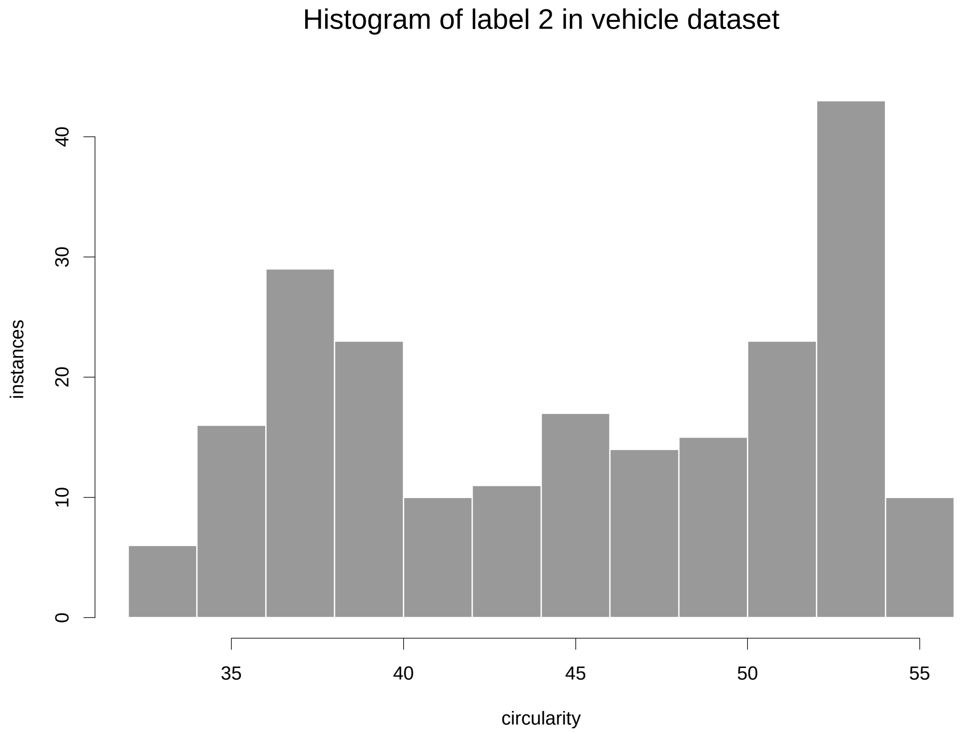

Figure 3.3: The distribution of the "circularity" feature and label value two. The normality assumption made by Naive-Bayes is wrong and becomes worse as enough instances are seen to estimate the mean at 45.5 accurately.

and the classifications then become worse.

Recently, Wolpert and Schaffer (Wolpert 1994*a*, Wolpert 1994*b*, Schaffer 1994) have advocated the use of off-training set error for testing. They suggested removing test-set instances that appeared in the training set, ignoring the label for those instances. (Our experiments allow for such test set instances if there are duplicate instances in the data.) Our experiments do not specifically remove such instances because we are interested in future expected accuracy, not generalization accuracy. Moreover, the datasets such as those tested here have very few duplicate instances and thus the results should be approximately the same. Table 3.1 shows the number of duplicate instances in each of the datasets used.

To see how well an accuracy estimation method performs, we sampled instances from the dataset (uniformly without replacement), and created a training set of the desired size.

We then ran the induction algorithm on the training set and tested the classifier on the rest of the instances in the dataset. This process was repeated 50 times at points where the *learning curve was not flat.* The motivation for these sampling points is that the differences in accuracy estimation should be amplified: if the learning curve is flat, inefficient use of the data might not show up. To allow comparisons between the two algorithms, the same folds in cross-validation and the same samples in bootstrap were used for both C4.5 and the Naive-Bayes.

3.4 The Bias and Variance of Cross-Validation and Bootstrap

It cannot be emphasized enough that no claim whatsoever is being made in this paper that all algorithms are equivalent in practice, *in the real world. In particular, no claim is being made that one should not use cross-validation in the real world.*
—*Wolpert (1994*a*)*

We now show our experimental results and discuss their significance. We begin with a discussion of the bias in the estimation methods and follow with a discussion of the variance. While an unbiased method is important to estimate future performance, the large variations in the results may deem an unbiased method inferior to a biased one with lower variance. When accuracy estimation is used for model selection, such as selecting the right amount of pruning for a decision tree (Breiman *et al.* 1984) or early stopping for neural nets (Finnoff, Hergert & Zimmermann 1993), we are interested in the *difference* between two classifiers, and the variance may be even more important because if the bias for the two models is equal, they cancel out.

3.4.1 The Bias

In this section, we investigate the bias of cross-validation and the .632 bootstrap. Since both methods estimate the accuracy by training the inducer on smaller samples from the dataset, we expect the estimates to be pessimistic (*i.e.*, lower accuracy, higher error).

The Bias of Cross-Validation

Figures 3.4 and 3.5 show the bias and variance of k-fold cross-validation on several datasets (the breast cancer dataset is not shown because it crosses the others and makes the graphs unreadable). Positive values on the x-axis indicate the number of folds; negative numbers indicate leave-k-out (*e.g.*, -2 stands for leave-two-out repeated $m/2$ times). The gray area indicates the 95% confidence interval for the true accuracy.

The diagrams clearly show that k-fold cross-validation is pessimistically biased, especially for two and five folds. For the learning curves that have a large derivative at the measurement point, the pessimism in k-fold cross-validation for small k's is apparent. For example, on the learning curve for C4.5, Soybean is about 3% lower in accuracy when going down from 100 training instances to 90, and the cross-validation estimate is therefore highly pessimistic even at ten folds. Most of the estimates are reasonably good at ten folds, and at twenty folds they are almost unbiased. In fact, almost no noticeable difference can be observed between twenty folds and leave-five-out for mushroom, hypothyroid, and chess (soybean, vehicle, and rand were tested at 100 instances, so 20-fold cross-validation is equivalent to leave-five-out).

Results for stratified cross-validation (Figures 3.6 and 3.7) are similar, except for lower pessimism. The estimated accuracy for soybean at two-fold was 7% higher and at five-fold, it was 4.7% higher; for vehicle at two-fold, the accuracy was 2.8% higher and at five-fold, 1.9% higher. Thus stratification seems to be a less biased estimation method.

An Alternative View of the Cross-Validation Bias

Cross-validation can be viewed as an unbiased estimator for the size of the internal training sets used; thus k-fold cross-validation is an unbiased estimator for datasets of size $m - m/k$. (This does not mean that cross-validation is unbiased for a specific given dataset, as Example 3.1 on page 44 shows, only that it is unbiased if we average all possible datasets.)

In stratified cross-validation, the test samples—and hence the training samples—are constrained to contain approximately the same proportion of the class labels as in the dataset. Stratification reduces the variance of the estimates because of the extra constraint on the class proportions (conditioning reduces variance). If background knowledge indicates that the dataset was stratified, *i.e.*, each class was sampled in proportion to its probability in the domain (these probabilities should be known based on background knowledge), then stratified cross-validation is clearly the correct choice, as we condition on something that is true. However, if no such knowledge exists, it is not clear that stratification is the correct thing to do.

The following experiments compare the cross-validation estimates with the accuracies

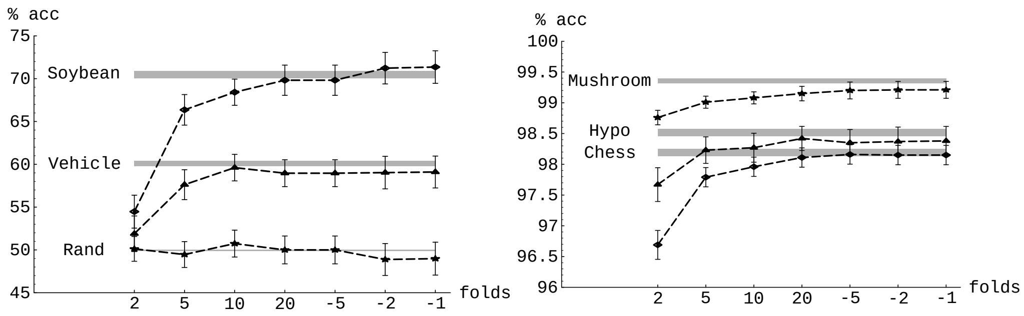

Figure 3.4: C4.5: The bias of cross-validation with varying folds. A negative k folds stands for leave-k-out. Error bars are 95% confidence intervals for the mean. The gray regions indicate 95% confidence intervals for the true accuracies. Note the different ranges for the accuracy axis and the difference in scale.

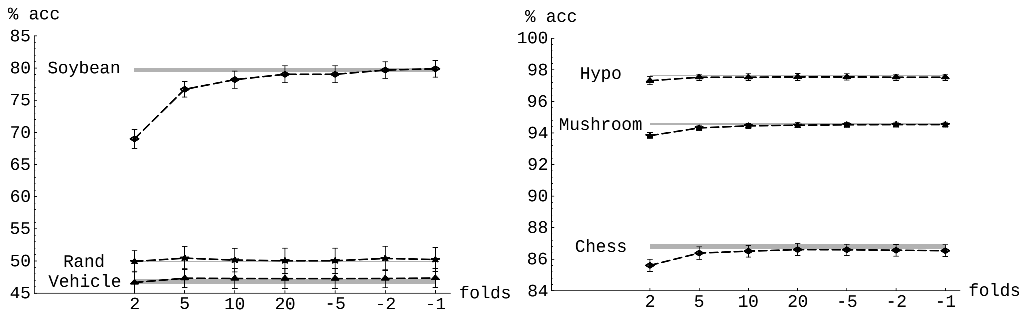

Figure 3.5: Naive-Bayes: The bias of cross-validation with varying folds.

of an inducer trained on datasets of size $m - m/k$. The "true" accuracies shown for samples of size $m - m/k$ are derived from sampling the original dataset (without replacement and without stratification). If cross-validation is an unbiased estimator for size $m - m/k$, then the results should be approximately equal. Figure 3.8 shows the difference in accuracies between the "true" accuracy and the cross-validation estimated accuracy. For each sample, we ran cross-validation and stratified cross-validation with the specified number of folds five times (running cross-validation five times is a better approximation to complete cross-validation; see Section 3.5 on page 59). Figures 3.8 and 3.9 show the differences in accuracy (note the different scale). Cross-validation has no significant bias, while stratified cross-validation is optimistically biased.

When stratified cross-validation is compared as an accuracy estimator for the given

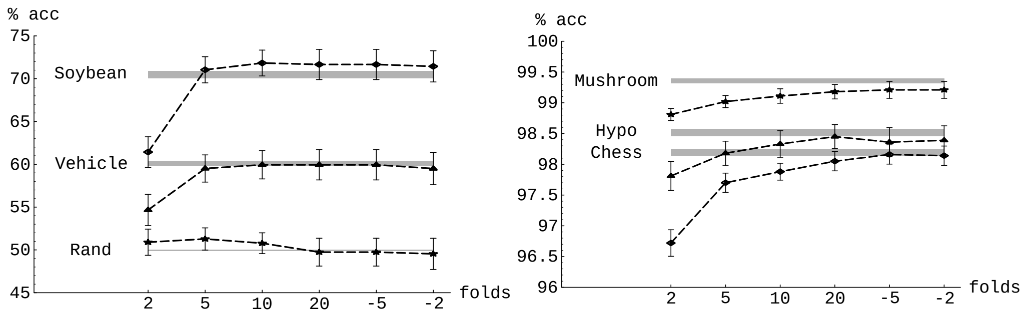

Figure 3.6: C4.5: The bias of stratified cross-validation with varying folds.

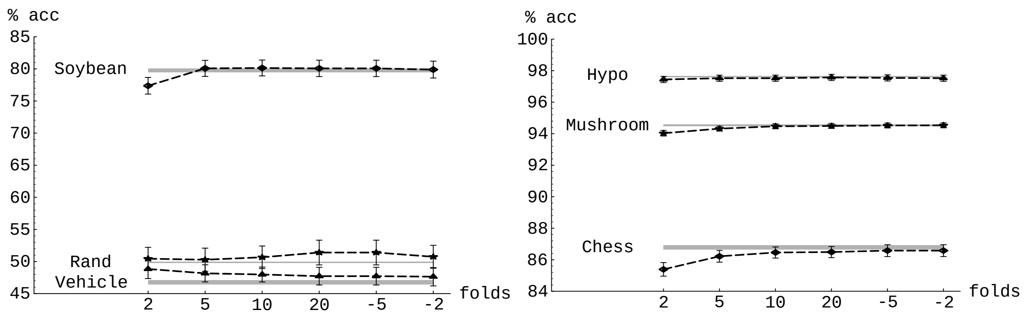

Figure 3.7: Naive-Bayes: The bias of stratified cross-validation with varying folds.

dataset (as opposed to a dataset of size $m - m/k$), then the pessimism of the small training set size is offset by the optimism of stratified cross-validation, leading to a less biased estimate. While there is no good theoretical justification for adding this level of optimism, it just seems to work well in practice.

The Bias of the .632 Bootstrap

Figures 3.10 and 3.11 show the bias and variance for the .632 bootstrap accuracy estimation method. Although the .632 bootstrap is almost unbiased for chess, hypothyroid, and mushroom for both inducers, it is highly biased for soybean with C4.5, vehicle with both inducers, and rand with both inducers. The bias with C4.5 and vehicle is 9.8%.

It is interesting to note that the bias of the .632 bootstrap is high for low-accuracy datasets. This observation suggests that an alternative bootstrap might change the weights of the $\epsilon 0$ and the resubstitution estimates (currently set to .632 and .318 correspondingly)

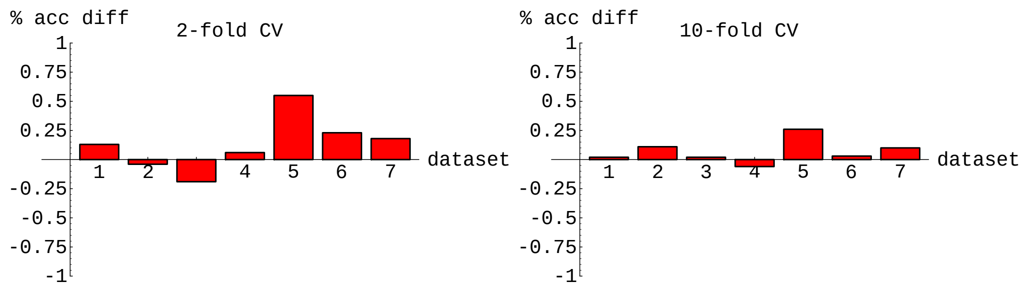

Figure 3.8: The bias of cross-validation for dataset of size $m - m/k$ is almost 0 (note the scale).

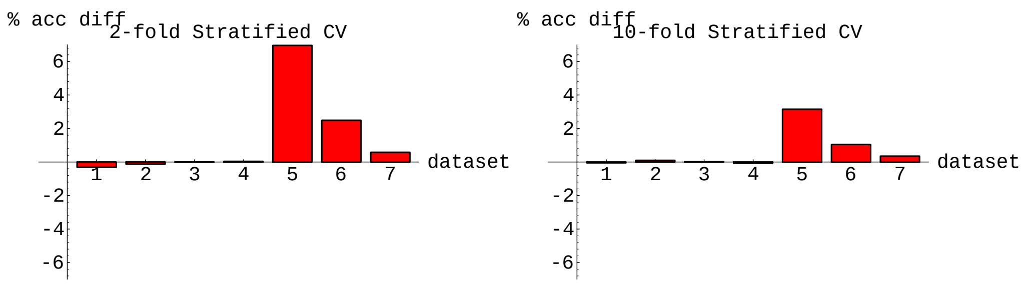

Figure 3.9: The bias of stratified cross-validation for dataset of size $m-m/k$. The optimistic bias is significant (note the scale).

based on the value of $\epsilon0$. See Section 3.8 on page 73 for more details.

Because the bias of the .632 bootstrap is so high, it is not a good estimator of accuracy for the datasets and inducers we tested. For model selection, the absolute accuracy is less important, but because the difference in the bias is very different between C4.5 and Naive-Bayes, it would not cancel out if the accuracy estimates are subtracted, and so we would not recommend the .632 bootstrap for model selection either.

3.4.2 The Variance

While a given method may have low bias, its performance (accuracy estimation in our case) may be poor due to high variance (see Section 2.5 for the bias-variance tradeoff). Researchers who do not have practical experience with accuracy estimation methods are often surprised by the large variance exhibited by these methods. In the experiments above,

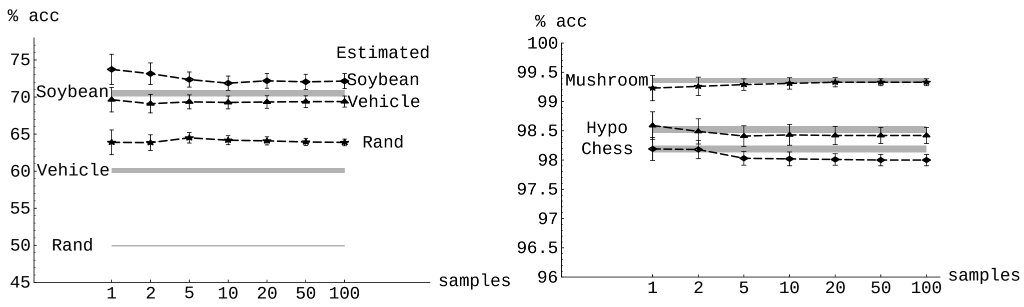

Figure 3.10: C4.5: The bias of bootstrap with varying samples. Estimates are good for mushroom, hypothyroid, and chess, but are extremely biased (optimistically) for vehicle and rand, and somewhat biased for soybean.

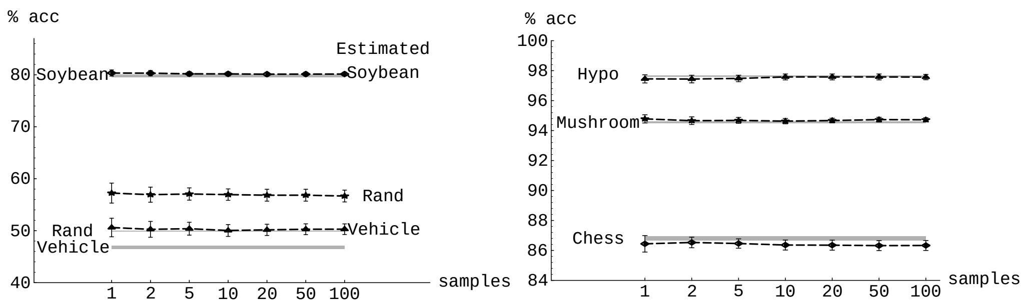

Figure 3.11: Naive-Bayes: The bias of bootstrap with varying samples. Estimates are good for mushroom, hypothyroid, and chess, but are extremely biased (optimistically) for vehicle and rand.

we have formed confidence intervals by using the standard deviation of the *mean accuracy.* We now switch to the standard deviation of the population, *i.e.*, the expected standard deviation of a single accuracy estimation run (the expectation is over the sample training-set and the cross-validation partition). In practice, if one does a single cross-validation run, the expected accuracy will be the mean reported above, but the standard deviation will be higher by a factor of $\sqrt{50}$, the number of runs we averaged in the experiments.

In what follows, all figures for standard deviation will be drawn with the same range for the standard deviation: 0 to 7.5%. Figure 3.12 shows the standard deviations for C4.5 and Naive Bayes using varying number of folds for cross-validation; Figure 3.13 shows the same information for stratified cross-validation (leave-one-out is not shown because it is exactly the same as for regular cross-validation); and Figure 3.14 shows the same information for the .632 bootstrap.

Cross-validation has high variance at two-folds with both C4.5 and Naive-Bayes. With C4.5, there is high variance at the high-ends too—at leave-one-out and leave-two-out—for three files out of the seven datasets. Stratification reduces the variance slightly, and thus seems to be uniformly better than cross-validation, both for bias and variance. The .632 bootstrap clearly has the smallest variance, but as we shall see in the next section, similar reductions in variance can be achieved for cross-validation by executing multiple runs.

3.5 Stabilizing Cross-Validation

It would be illogical to assume that all conditions remain stable.
—Spock, "The Enterprise Incident," stardate 5027.3

The bias of cross-validation can be reduced by increasing the number of folds, but increasing it too much may increase the variance. Even for the optimum number of folds chosen to reduce variance, the variance may be much too high to be useful. We now describe two variations of cross-validation aimed at reducing the variance: multiple runs and trimming.

3.5.1 Multiple Runs of Cross-Validation

Complete cross-validation is the average of all $\binom{m}{m/k}$ possibilities for choosing m/k instances out of m. The cross-validation estimate commonly used chooses a single split of the data into k folds, thus approximating complete k-fold cross-validation with k estimates having disjoint test sets. Executing cross-validation multiple times, each time with a different split into the k folds, can viewed as a Monte-Carlo estimation (Binder & Heerman 1988) to complete k-fold cross-validation, which is usually too expensive to run.[3] Repeating cross-validation multiple times will not change the bias inherent in the method but it might change the variance of the estimates.

[3]One reviewer asked if we ever tried running complete cross-validation to show that it is better for our datasets. For the chess dataset, one would need to run the induction algorithm $\binom{900}{90}$ times. If every one of the 10^{80} atoms in this universe were replaced by a machine a million times faster than the Sparc 10 we used, and assuming C4.5 were optimized to be a million times faster, one would still need 10^{25} years, which we believe would miss some important deadlines.

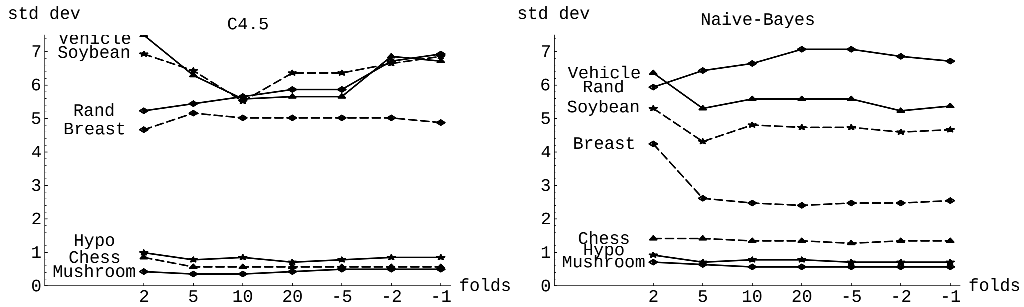

Figure 3.12: Cross-validation: standard deviation of accuracy (population). Different line styles are used to help differentiate between curves.

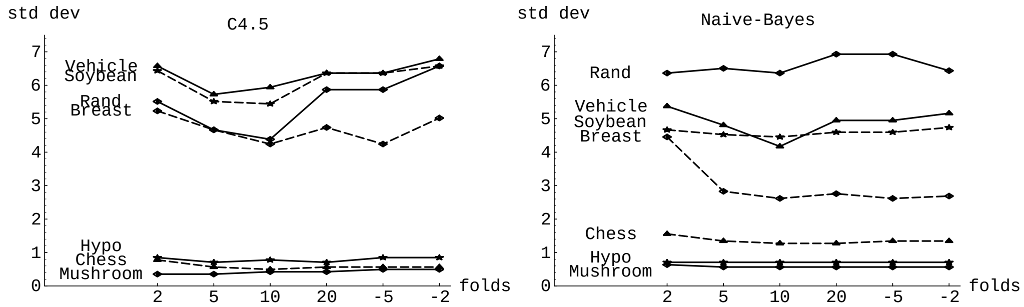

Figure 3.13: Stratified cross-validation: standard deviation of accuracy (population).

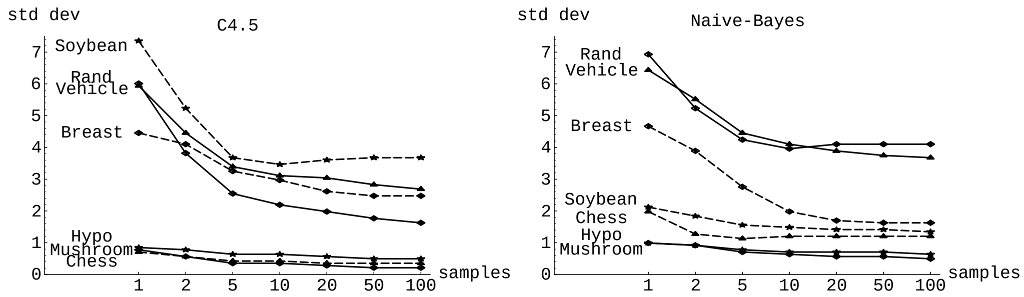

Figure 3.14: The .632 Bootstrap: standard deviation of accuracy (population).

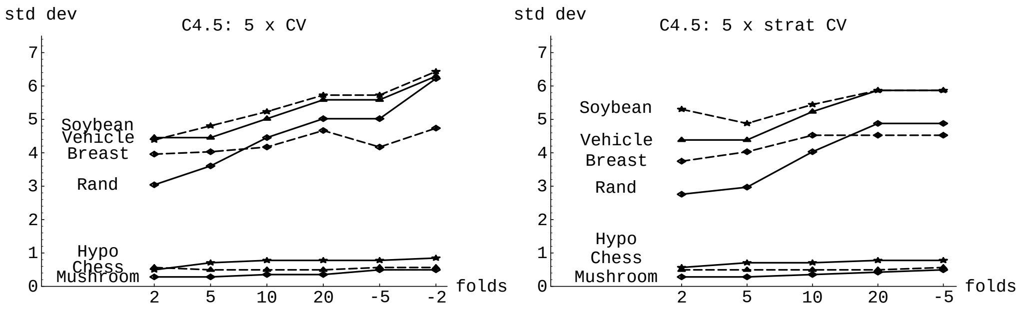

Figure 3.15: C4.5 cross-validation repeated five times: standard deviation of accuracy (population).

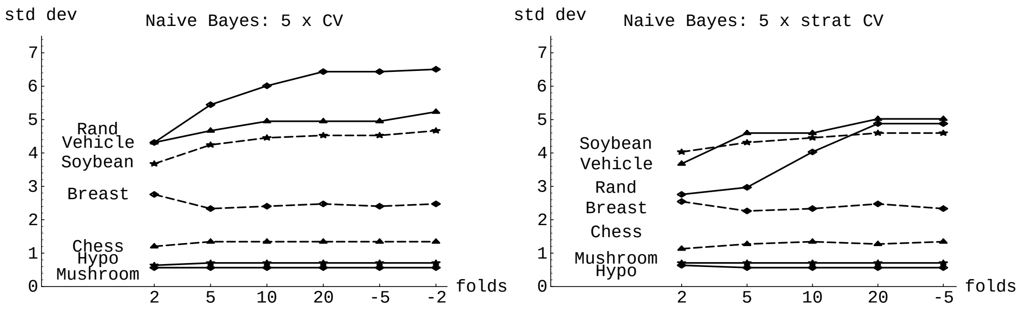

Figure 3.16: Naive-Bayes cross-validation repeated five times: standard deviation of accuracy (population).

Figures 3.15 and 3.16 show the variance of the cross-validation estimates when five cross-validation runs are made for the given number of folds. The emerging picture is quite different from that of running cross-validation a single time (Figures 3.12 and 3.13). Two-fold cross-validation has the lowest variance, and stratification does not seem to help as much.

Figures 3.17 to 3.20 show the variance of two-fold and ten-fold, with and without stratification, for varying times. One can see that significant decreases in variance can be achieved as the number of repetitions goes up to about 20. Comparing two-fold cross-validation with the .632 bootstrap shows that they have very similar variance. The advantage of cross-validation is that it allows choosing the number of folds to achieve a balance between the desired bias and variance.

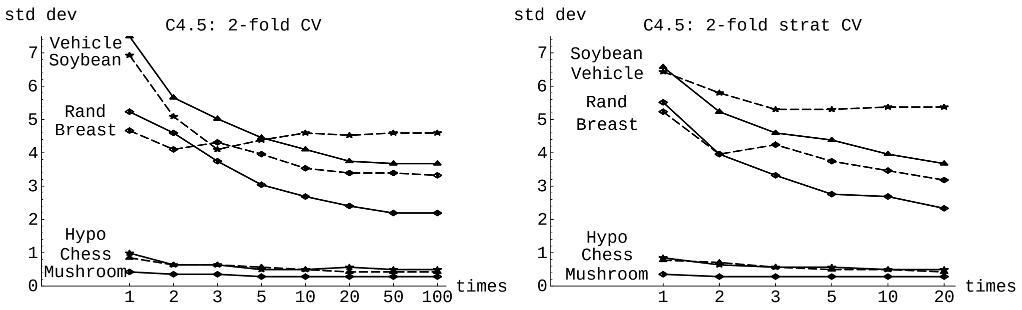

Figure 3.17: C4.5 with two-fold cross-validation : regular (left), stratified (right) with varying times.

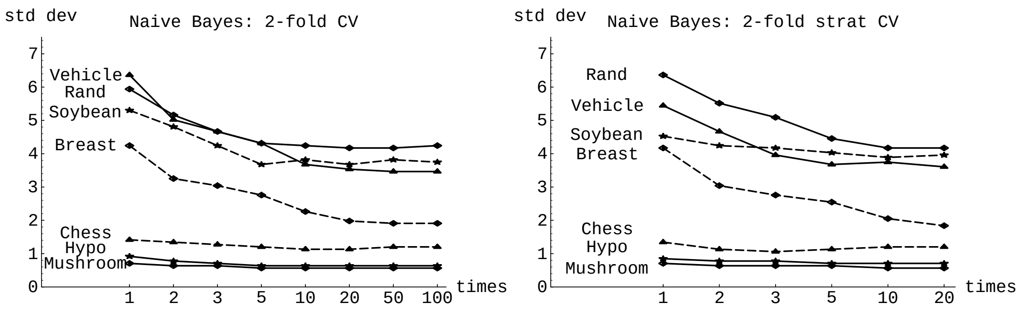

Figure 3.18: Naive-Bayes with two-fold cross-validation : regular (left), stratified (right) with varying times.

3.5.2 Cross-Validation: the Distribution and Trimming of the Folds

The property of robustness I believe to be even more important in practice than that the test should have maximum power and that the statistics employed should be fully efficient.
—G. E. P. Box, 1953

John (1994) and Walker (1992) suggested using a trimmed mean to reduce the variance of cross-validation. An **$\boldsymbol{\alpha}$-trimmed mean** ignores the 100α extreme data points (highest and lowest) when computing the mean (details below). The argument is that some cross-validation folds are unrepresentative because they contain only $1/k$ of the instances. Trimmed means are known to have good properties: they are robust to outliers up to $100\alpha\%$, their asymptotic efficiency relative to the untrimmed mean never drops below $(1-2\alpha)^2$, and the standard errors can be estimated easily (Staudte & Sheather 1990, Rice 1988).

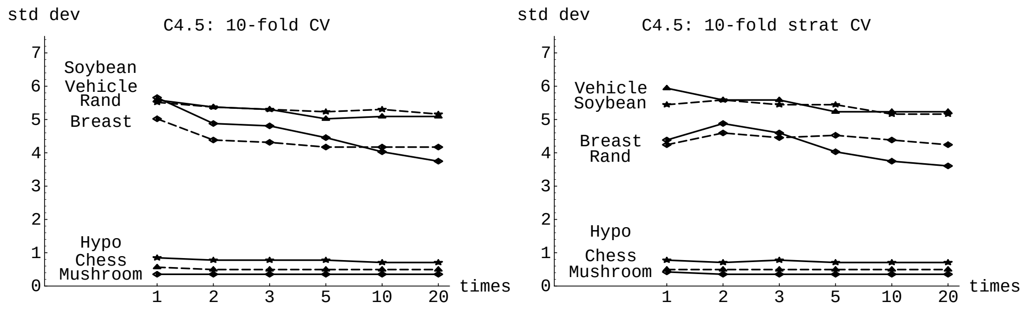

Figure 3.19: C4.5 with ten-fold cross-validation : regular (left), stratified (right) with varying times.

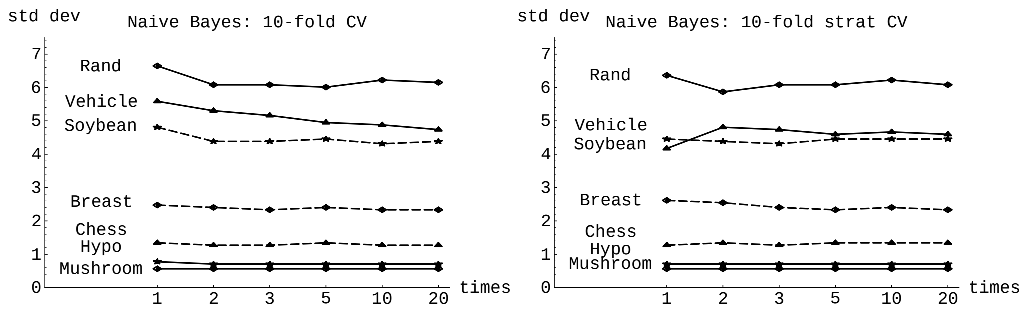

Figure 3.20: Naive-Bayes with ten-fold cross-validation : regular (left), stratified (right) with varying times.

In order to evaluate this suggestion and similar ones, it is useful to look at the distribution of cross-validation. Figure 3.21 shows the distribution of 1,000 cross-validation estimates with C4.5 (the figures for Naive-Bayes are similar in nature), accumulated by doing 20 times ten-fold cross-validation for each of the 50 samples from the original dataset. The histograms show that the cross-validation estimates are approximately normally distributed, but they are not very smooth.[4] Figure 3.22 shows the distribution of the 10,000 fold estimates (20 times ten-fold cross-validation, repeated for each of the 50 samples) and reveals one reason for the non-smoothness: few test instances in each fold. With only 50 instances for the breast cancer dataset, each fold contains only five instances, hence the

[4]The binning was determined automatically by S-plus (Spector 1994). While these histograms could be made to look more Gaussian by changing the bin sizes, the histograms shown are more informative because they indicate important gaps in the accuracy estimates.

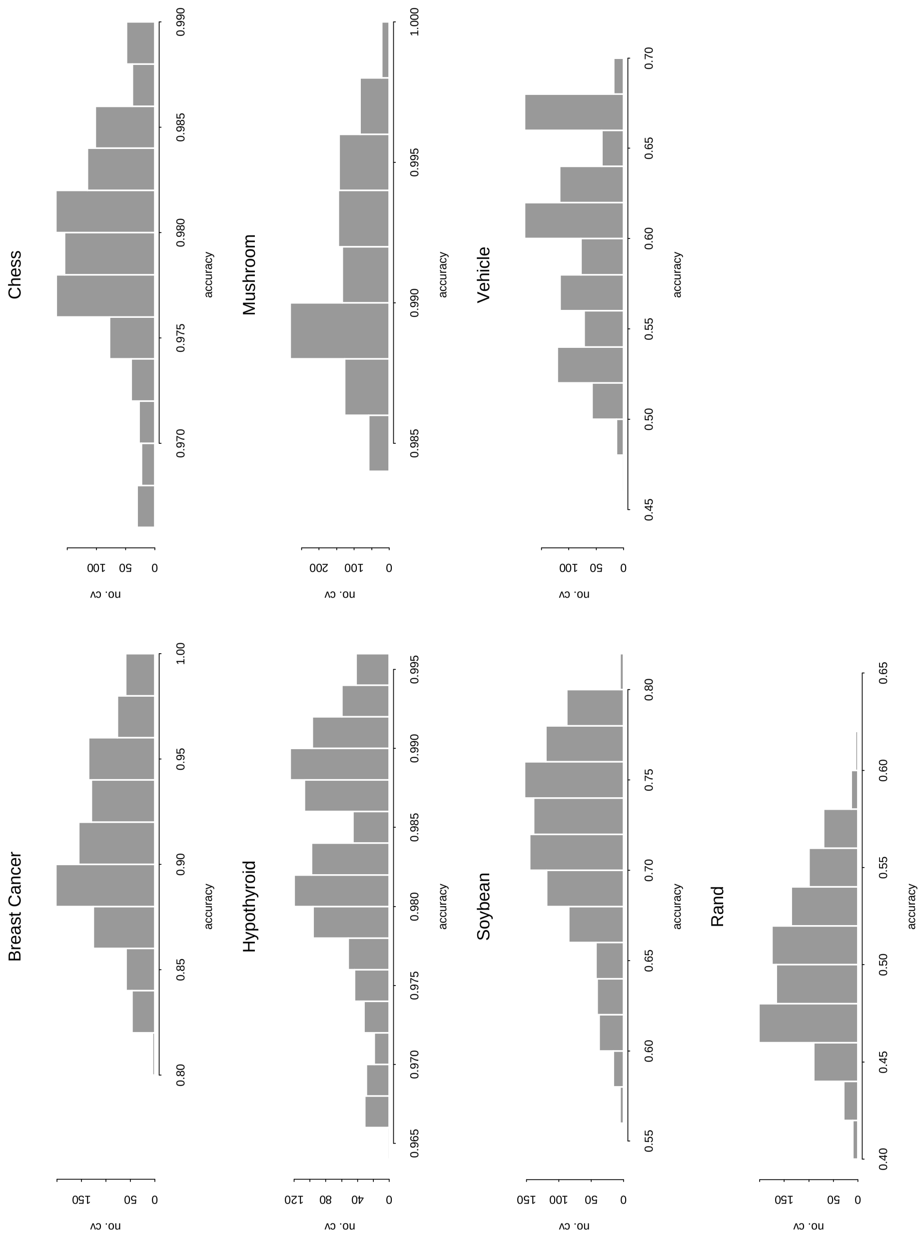

Figure 3.21: C4.5: Distribution for the cross-validation accuracies in 20 times ten-fold cross-validation (repeated 50 times)

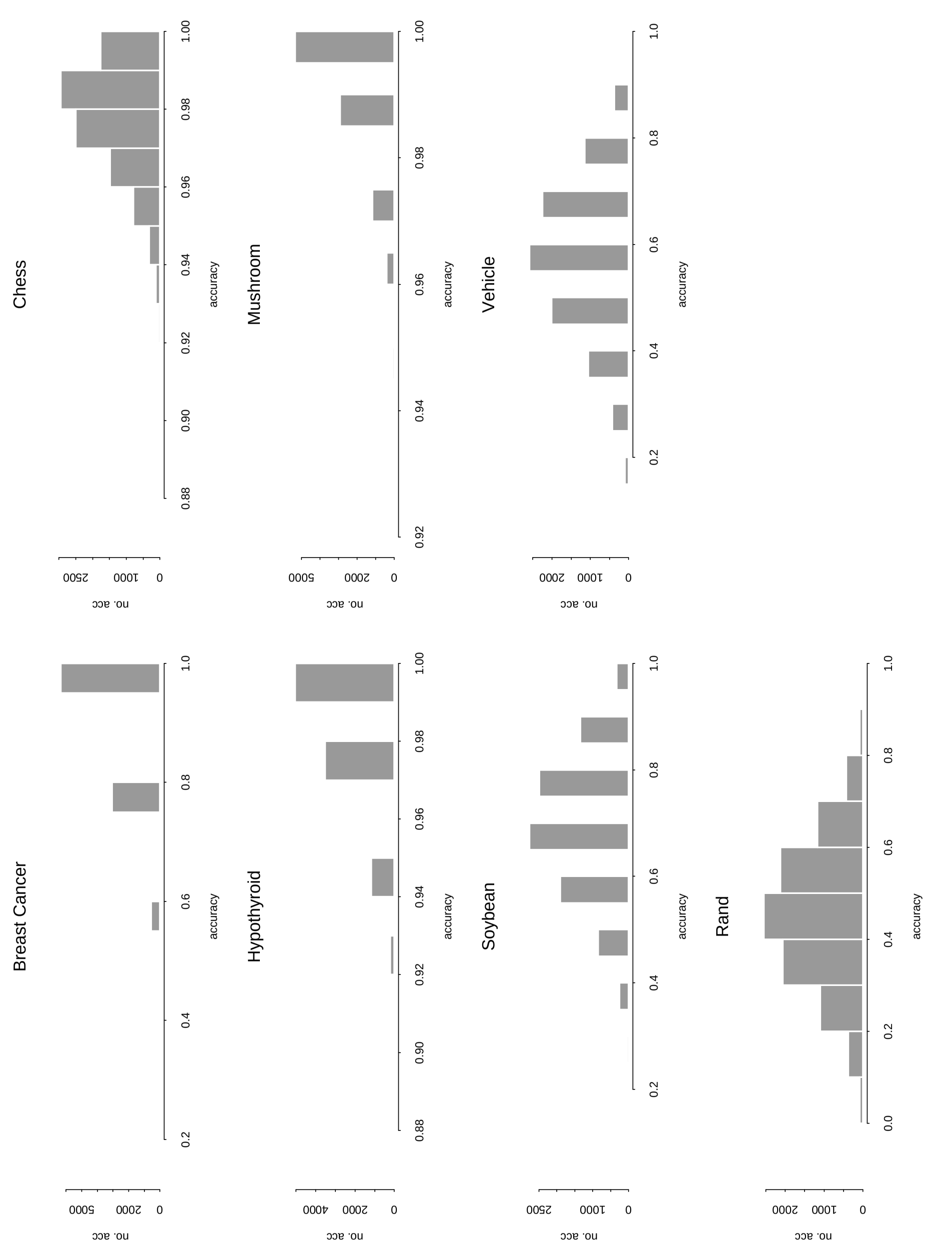

Figure 3.22: C4.5: The distribution for the fold accuracies in 20 times ten-fold cross-validation (repeated 50 times)

accuracy must be a multiple of 20%. Even for Hypothyroid, with 400 instances, the non-smoothness is apparent, although at a different scale. The accuracy can be made smoother if the predictions were probabilistic, *i.e.*, the classifier would return a probability distribution. This idea is discussed in Section 3.8 on page 73.

Apart from the problem of discrete classification just mentioned, the breast-cancer, the chess, the hypothyroid, and the mushroom datasets have a skewed distribution because an accuracy of 100% is an upper limit and the folds do not form a symmetric distribution around the mean. If the underlying distribution is Gaussian, then the sample mean is the minimizer of the sum of squared error from the true mean. Some studies show that for heavy tails, trimmed means are better, and the variance of the estimates is not much larger than the ordinary mean even if the underlying distribution is Gaussian (Andrews, Bickel, Hampel, Huber, Rogers & Tukey 1972). In the following experiment, we evaluate the use of a trimmed mean.

We experimented with two-fold and ten-fold cross-validation for both C4.5 and Naive-Bayes using an $\alpha = 10\%$ trimmed mean. Our trimmed mean removed the extreme $\lfloor \alpha n \rfloor$ points, so for two-fold cross-validation, trimming kicks in only when there are five repetitions (at least ten folds are needed). (Andrews *et al.* (1972) and Hoaglin, Mosteller & Tukey (1983) suggest weighting the remaining points with a weight of $1 + \lfloor \alpha n \rfloor - \alpha n$ and then taking a weighted mean, but we have not tried this procedure.) The formula in Rice (1988) (page 333) was used to compute the variance of a trimmed mean. The variance is not significantly different and the estimates are slightly optimistic compared to the untrimmed variants. For example, the accuracy differences between 20 times two-fold cross-validation with 10% trimmed mean and the untrimmed mean for the seven datasets were 0.64%, 0.07%, 0.09%, 0.01%, 0.21%, 0.15%, 0.01%. For 20 times ten-fold cross-validation the differences were 1.75%, 0.15%, 0.3%, 0.18%, 0.43%, 0.26%, -0.02%. Except for the random datasets, all results were still pessimistically biased. Figures 3.23 and 3.24 show the variance of cross-validation with 10% trimming for two and ten-fold cross-validation. Trimming slightly reduces the bias, but not the variance. The differences are generally very small.

3.6 Learning-curve Extrapolation

`NULL HYPOTHESIS`: *The type of hypothesis used by a pessimist.*
—Unix fortune

The pessimistic bias of cross-validation is the result of training the induction algorithm

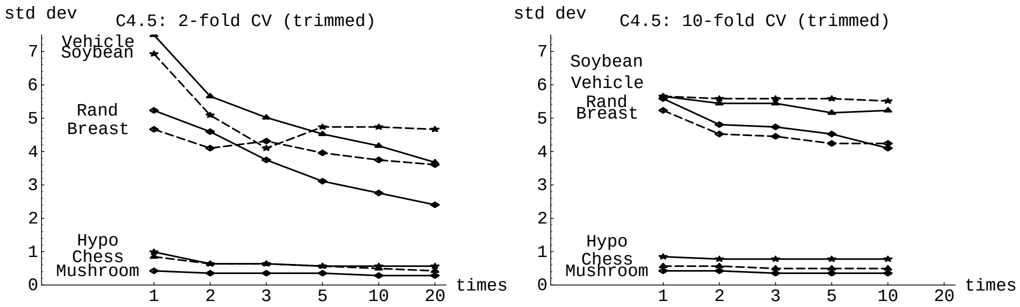

Figure 3.23: C4.5: cross-validation with 10% trimming: two-fold (left), ten-fold (right).

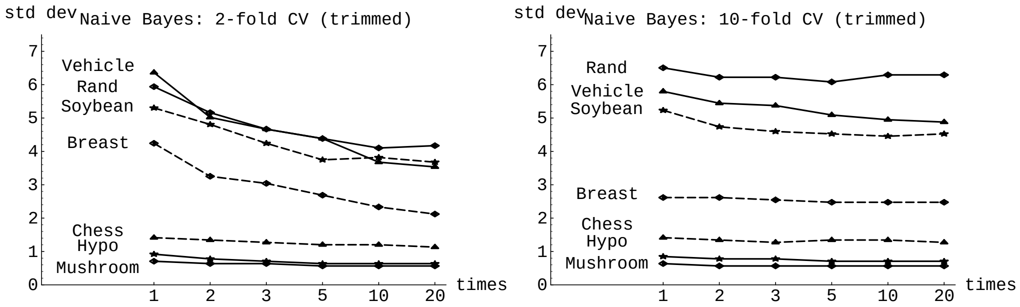

Figure 3.24: Naive-Bayes: cross-validation with 10% trimming: two-fold (left), ten-fold (right).

on fewer instances than there are in the entire dataset. To correct for this bias, we first assume that the pessimism is only a function of the induction algorithm and the number of training instances used. We attempt to approximate this function and extrapolate its value for the number of instances in the full dataset. While extrapolation in general is unstable, we extrapolate close to the sampled region and thus do not introduce much variance. The function we approximate is the learning curve, which relates the training set size to the true accuracy.

Cortes, Jackel, Solla, Vapnik & Denker (1994) have also extrapolated the learning curve (see Section 3.7 on page 69), but with a different motivation in mind. They sought to avoid training inducers on large amounts of data before inferring that one is superior. Training inducers on large amounts of data requires large amounts of resources that are unlikely to be available if many inducers are to be explored (see Kohavi & John (1995) for an example

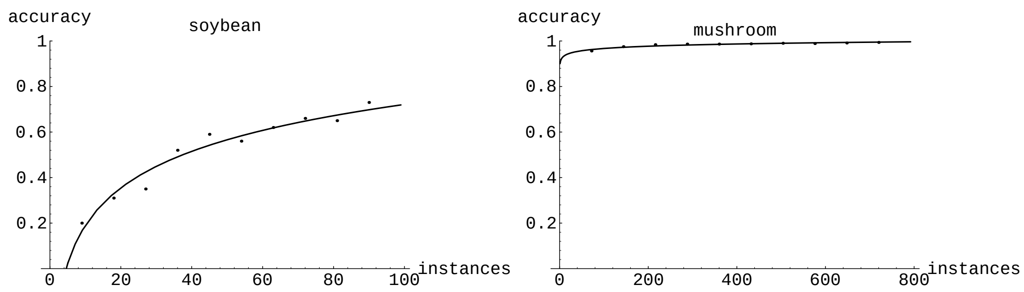

Figure 3.25: The bias corrected learning-curve fit to the data at ten points.

Dataset	sample-size total size	C4.5 True	C4.5 bias corrected estimate	Naive-Bayes True	NB bias corrected estimate
Breast cancer	50/699	91.37±0.10	91.54±0.53	94.22±0.10	94.33±0.41
Chess	900/3196	98.19±0.03	98.44±0.08	86.80±0.07	87.11±0.18
Hypothyroid	400/3163	98.52±0.03	98.45±0.10	97.63±0.02	97.70±0.10
Mushroom	800/8124	99.36±0.02	99.37±0.05	94.54±0.03	94.57±0.08
Soybean large	100/683	70.49±0.22	69.15±0.77	79.76±0.14	79.61±0.61
Vehicle	100/846	60.11±0.16	59.54±0.81	46.80±0.16	48.77±0.64
Rand	100/3000	49.96±0.04	50.40±0.66	49.90±0.04	50.29±0.93
Average		81.14	80.98	78.52	78.91

Table 3.2: True and bias-corrected ten-fold cross-validation estimates.

where this line of work can be extremely useful).

Stone (1982) showed that under appropriate regularity conditions, the learning curves of non-parametric regression algorithms behave as $1 - m^{-\alpha}$ where m is the number of instances and α is a constant that depends on the dataset and the learning algorithm. Haussler (1992) proved that samples of size $m = \frac{M^2}{2\epsilon^2}\left(\ln|\mathcal{H}| + \ln\frac{2}{\delta}\right)$ suffice to probably approximately correctly learn in the agnostic PAC model (where the target function is non-negative and M is an upper bound on the loss function), giving another justification for the $m^{-\alpha}$ model. We thus approximate the learning curve of an algorithm by a parametric curve of the form $a + b \cdot m^{-\alpha}$ where a, b, α are parameters to be determined by the dataset and the induction algorithm. Experimental evaluations on over twenty datasets from UC Irvine indicate that the family is quite general and gives a good fit to the learning curves of several algorithms on different datasets.

Our estimation process samples points from the learning curve using ten-fold cross-validation. A standard run of ten-fold cross-validation gives us the 90% sample point; other

Dataset	5x2-fold	5x5-fold	5x10-fold	5x20-fold	Bias-corrected
Breast cancer	4.14	4.05	4.14	4.61	3.69
Chess	0.55	0.70	0.55	0.53	0.64
Hypothyroid	0.75	0.75	0.77	0.78	0.72
Mushroom	0.41	0.44	0.41	0.42	0.32
Soybean large	5.45	6.38	5.45	5.68	5.57
Vehicle	5.18	4.99	5.11	5.57	5.69
Rand	4.01	3.60	4.38	4.96	4.65

Table 3.3: Comparison of root mean square error (RMSE) for five times cross-validation with the indicated number of folds and of the bias-corrected cross-validation.

points are computed by shrinking the training set sizes from the ten-fold cross-validation, keeping the test sets fixed. Specifically, we generate ten points, using 10% to 90% of the training sets determined by ten-fold cross-validation. The reason for keeping the test sets fixed is to avoid variance that arises from the shuffling in cross-validation.

The curve is fitted using Levenberg Marquardt non-linear least squares with starting values of $a = b = 0.5$ and $\alpha = 0.01$ (Marquardt 1963, Press, Teukolsky, Vetterling & Flannery 1992). These values were determined by looking at a few learning curves; we do not attribute any great significance to the starting point. After the curve is fitted, the accuracy performance is estimated by plugging in the full dataset size. Table 3.2 shows the estimated accuracies for the C4.5 and the Naive-Bayes induction algorithms after this extrapolation, which we call **bias correction.**

The root mean square error (the difference was taken between the estimate and the true error) is shown in Table 3.3. The bias correction run using ten samples of ten-fold cross-validation outperformed five times twenty-fold cross-validation in terms of root mean square (both require the same number of cross-validation folds). The bias-corrected method seems to be almost unbiased and not less stable than ten times ten-fold cross-validation. Figure 3.25 shows the fit to the ten estimated points for two datasets that generated distinctly different learning curves.

3.7 Related Work

I have in mind procedures such as AID, the automatic interactor detector, which guarantees to get significance out of any data whatsoever.
—G. A. Barnard (Stone 1974, Discussion)

Some experimental studies comparing different accuracy estimation methods have been previously reported, but most of them were on artificial or small datasets. We now describe some of these efforts.

Lachenbruch & Mickey (1968) conducted some experiments and compared leave-one-out with holdout estimation and resubstitution estimation. The concluded that "no one method is uniformly best for all situations," but that the resubstitution estimate is relatively poor.

Stone (1974) suggested the use of cross-validation both for choosing a predictor and assessing its quality. He looked at examples for the areas of univariate estimation, linear regression, and analysis of variance, and showed that prescriptions of parameters made by cross-validation agree with some analytical results on artificial distributions. He also demonstrated the use of cross-validation for estimating the accuracy of a weighted least-squares fit to a satellite dataset with 27 instances. A few years later (Stone 1977), he showed that estimating parameters by leave-one-out cross-validation leads to asymptotically inconsistent estimates in some cases, such as deciding whether to estimate a parameter by the median or the mean.

Geisser (1975) apparently discovered cross-validation independently of Stone and used the name "sample reuse." He noted that although k-fold cross-validation is pessimistic (conservative in his terms), some conservativeness may be offset due to repeated optimizations on much of the same data. The problem of overusing the accuracy estimation may be severe especially if the datasets are small (see Section 4.7 on page 114).

Efron (1983) conducted five sampling experiments and compared leave-one-out cross-validation, several variants of bootstrap, and several other methods. The purpose of the experiments was to "investigate some related estimators, which seem to offer considerably improved estimation in small samples." Four experiments were based on bivariate normal distributions with two to five features and 14 to 20 instances; the fifth experiment was based on a real world dataset described in Gong (1982) with 4 features and 20 instances. The results indicate that leave-one-out cross-validation gives nearly unbiased estimates of the accuracy but often with unacceptably high variability, particularly for small samples; the .632 bootstrap performed best.

Breiman *et al.* (1984) conducted experiments using cross-validation for decision tree pruning. They chose ten-fold cross-validation for the CART program and claimed it was satisfactory for choosing the correct tree. They also claimed that "the difference in the cross-validation estimates of the risks [accuracy here] of two rules tends to be much more accurate than the two estimates themselves."

Knoke (1986) provided a survey of error estimation of classification rules, but he assumed that classification is based on linear discriminant functions. Because the asymptotic bias of the resubstitution estimate is on the order of $1/m$ (McLachlan 1976) for linear discriminant functions, he concluded that "for large samples, there is no need to look further than the resubstitution estimator when seeking a robust method."

Jain *et al.* (1987) compared the performance of the $\epsilon 0$ bootstrap and leave-one-out cross-validation with nearest-neighbor classifiers using artificial data and claimed that the confidence interval of the bootstrap estimator is smaller than that of leave-one-out. Weiss (1991) followed similar lines and compared stratified cross-validation and two bootstrap methods with nearest-neighbor classifiers. His results indicated that stratified two-fold cross-validation has relatively low variance and is superior to leave-one-out. A corrected leave-one-out estimator was suggested that is equal to either the .632 bootstrap or two-fold cross-validation, depending on the relation between them and leave-one-out. The experiment was conducted on one real-world dataset (hypothyroid) and three artificial datasets with two features. Samples of sizes 20 and 60 were used, but the conclusions from samples of size 20 seemed less clear for those of size 60. Weiss wrote that "the variance of leaving-one-out is most evident with very small samples such as size 20, and gradually lessens as the sample size increases."

Weiss (1991) writes that stratified two-fold cross-validation is a relatively low variance estimator (the experiments show root mean square error from the true error rate), but then wrote that two-fold cross-validation was weaker than leaving-one-out for low true error rates. The variance was reduced sufficiently for the bias of two-fold cross-validation to be visible.

Crawford (1989) discusses the .632 bootstrap as an alternative method for pruning decision trees. His experiments were done on small artificial datasets and he claims that as the sample size is reduced to 20, the large variance in the cross-validation takes its toll, and the .632 bootstrap clearly outperforms cross-validation.

Breiman & Spector (1992) conducted a feature subset selection experiment for regression and compared leave-one-out cross-validation, k-fold cross-validation for various k, stratified k-fold cross-validation, bias-corrected bootstrap, and partial cross-validation (not discussed here). Tests were done on artificial datasets with 60 and 160 instances using multivariate mean-zero normal or lognormal distributions with 40 coefficients (features), but only 3 to 21 were non-zero. The behavior observed was:

1. Leave-one-out has uniformly low bias and RMS (root mean square) error. Five-fold cross-validation and two-fold cross-validation have larger bias and RMS error at the larger submodels, *i.e.*, models with many features.

2. For 60 instances, ten-fold cross-validation is biased pessimistically with larger RMS error at the higher dimensional submodels. This bias is considerably reduced for samples of size 160.

3. The bias-corrected bootstrap has a fairly low bias and RMS error for samples of size 160. With 40 features, bootstrap could not be run on the samples of size 60 because there were only about $.632 \cdot 60 = 37.92$ unique instances in the bootstrap samples, implying that the regression matrix $X^T X$ was singular most of the time. The differences between 20 and 50 bootstrap samples were small.

4. Stratification did not seem to have any effect on cross-validation.

5. For selecting the correct model, ten-fold cross-validation was uniformly better than leave-one-out. On samples of size 160, the bias-corrected bootstrap had a slight edge. Although five-fold cross-validation is not as good an estimator globally (*i.e.*, for accuracy estimation) as ten-fold cross-validation, it does as well on submodel selection and evaluation. Similarly, for submodel selection, the accuracy holds up even for as few as five bootstrap samples.

Bailey & Elkan (1993) compared leave-one-out cross-validation to the .632 bootstrap using the FOIL inducer and four synthetic datasets involving Boolean concepts in DNF using 50 Boolean features with four to eight relevant ones. The number of training instances was 100 with 50 instances of each class (stratified sample). They observed high variability and little bias in the leave-one-out estimates, and low variability but large bias in the .632 estimates.

Weiss and Indurkyha (Weiss & Indurkhya 1994*b*, Weiss & Indurkhya 1994*a*) conducted experiments on real-world data to determine the applicability of cross-validation to decision tree pruning. Their results were that for samples of size at least 200, using stratified ten-fold cross-validation to choose the amount of pruning yields unbiased trees (with respect to their optimal size). For smaller samples, they found that while ten-fold is nearly unbiased, a strategy based on two-fold cross-validation is generally more effective.

Zhang (1992*b*) and Shao (1993) showed that in order to select a linear model containing the best set of features, as m grows, the number of instances held for testing (*i.e.*, not given in the training set) should grow too. Zhang showed that the models chosen by any k-fold cross-validation for any k will overfit in the sense that too many features will be selected. However, for moderately-sized data sets, he claims that ten to fifteen folds are reasonable choices.

Cortes *et al.* (1994) also extrapolated the learning curve, but using a different model. Their model assumed that the shape of the error curve for the resubstitution error is $a - b/m^\alpha$ for some constants a, b, and α, and that the shape of the error curve for the true error (one minus the learning curve as defined here) is $a + b/m^\alpha$ (for the same a, b, and α as in the resubstitution error curve). This model was motivated by statistical mechanics and validated on Boolean classifiers with linear decision surfaces. It seems to work well for neural networks, but the model is bad for many known induction algorithms (*e.g.*, the resubstitution curve for one nearest-neighbor is a constant zero in noiseless domains).

Kadie (1995) and Kadie & Wilkins (to appear) have modeled the shape of learning curves in a system called Seer. They fit two models, called EDit and Burr, that are motivated by computational learning theory. The EDit model is based on the upper bound on the number of instances needed to learn a hypothesis with a given VC dimension (Shawe-Taylor, Anthony & Biggs 1993); the Burr model is similar in shape, but simpler. Both models are modified to account for the roughness of the learning curve by a noise term that has a beta distribution. Parameters for the model were found by fitting the curve using the Levenberg-Marquardt nonlinear optimization (Marquardt 1963, Press *et al.* 1992).

3.8 Future Work

> FUTURE, *n. That period of time in which our affairs prosper, our friends are true, and our happiness is assured.*
> —*The Cynic's Word Book, Bickersteth*

The work presented in Kohavi (1995*b*), which was the basis for this chapter, renewed the interest of Efron and Tibshirani in the subject of accuracy estimation using bootstrap. They have recently proposed a modified rule, called the 632+ rule (Efron & Tibshirani 1995), which attempts to correct for the bias exhibited in Figures 3.10 and 3.11 by changing the .632 factor based on the ϵ_0 estimated accuracy. The new rule also works for perfect memorizers and random data, where the .632 bootstrap fails. Since the new rule was somewhat motivated by the datasets reported here, it would be unfair to compare it for these datasets and a new set would need to be chosen.

We have addressed issues of bias and variance separately. These are two separate numbers that we feel are more important than a single report of the root mean squared error (RMSE). Specifically, for model selection the variance is more important if the bias is the same for the algorithms compared. More research should be done to compare the accuracy estimation methods estimates for the *difference* between models.

In the alternative view of cross-validation (Section 3.4.1), model selection using k-fold cross-validation will select the estimated best algorithm for which performance is best for $m - m/k$ instances. Unless the learning curves cross from $m - m/k$ instances to m, then the choice will be correct. There have been some recent results indicating the occurrence of phase transitions (*i.e.*, a sudden large increase in accuracy) in learning curves of specific learning algorithms (Haussler, Kearns, Seung & Tishby 1994, Hertz, Krogh & Thorbergsson 1989, O'Kane 1994). In practice, we have not seen a phase transition using the C4.5 or Naive-Bayes algorithms on any of the datasets in the UC Irvine repository (Murphy & Aha 1995); however, if they do occur between datasets of sizes $m - m/k$ and m, they are likely to cause the learning curves to cross, and hence the wrong model will be selected.

Devijver & Kittler (1982, Chapter 10) show that for a nearest-neighbor classifier, using the probability distribution over classes instead of the majority of the nearest neighbors provably reduces the expected variance of the estimated accuracy (note that this is a different estimate of the accuracy because the loss function is not 0/1). Glick (1978) makes similar claims and attempts to smooth counting estimators. Classifiers that can produce probability distributions for the classes can be used in this manner. To our knowledge, such second-generation accuracy estimators have not been explored experimentally, although they seem to be a promising approach (see also Section 3.5.2 on page 62).

3.9 Summary

The Guide is definitive. Reality is frequently inaccurate.
—Douglas Adams, The Hitchhiker's Guide to the Galaxy.

We reviewed common accuracy estimation methods: holdout, cross-validation, and the .632 bootstrap. We showed examples where each one fails to produce a good estimate and compared the latter two approaches on a variety of real-world datasets with differing characteristics. We used about 15 Sparc 20s to run this experiment, which included over a million runs of C4.5 and the Naive-Bayes algorithm. An experiment of this scale could not have been done a few years ago.

Proposition 1 on page 43 shows that if the induction algorithm is stable for a given dataset, the variance of the cross-validation estimates should be approximately the same, independent of the number of folds. Although the induction algorithms are not stable, they are approximately stable for a reasonable number of folds. k-fold cross-validation with moderate k values (10-20) reduces the variance while increasing the bias. As k decreases (2-5) and the sample sizes get smaller, there is variance due to the instability of the training sets themselves, leading to an increase in variance. This variance is most apparent for datasets with many categories, such as soybean. In these situations, stratification seems to help. Repeated cross-validation runs change the picture for $k \leq 10$. For five repetitions, it is no longer the case that the variance increases as k decreases, in fact, the variance decreases. We conclude that repeated cross-validation runs seem to be very useful when the number of folds is small.

Our results indicate that the .632 bootstrap has low variance, but extremely large bias on some problems. Stratified cross-validation is better than ordinary cross-validation both in terms of bias and variance; however, stratification yields an optimistic accuracy estimators for the amount of data used in training. Trimming seems to be of little help in reducing the variance, and the problem of discrete accuracy estimation resulting form the use of zero-one loss are apparent in the figures showing the distribution. Second-generation accuracy estimation methods that use the probability distributions may mitigate the problem.

Our recommendation is to use stratified cross-validation with ten to twenty folds for accuracy estimation, where the bias is usually more important. For model selection, where bias is not as crucial, we recommend multiple runs of five-fold cross-validation. One approach, which works well in practice, is to dynamically choose the number of runs based on the estimated variance of the accuracy: if the variance is high, execute another run of cross-validation.

Chapter 4

Wrappers

WRAP AROUND: VI. *To change* phase *gradually and continuously by maintaining a steady wake-sleep cycle somewhat longer than 24 hours, e.g., living six long (28-hour) days in a week (or, equivalently, sleeping at the rate of 10 microhertz).*
—*Jargon File, V2.9.9*

Users of machine learning algorithms must decide not only which algorithm to use on a particular dataset, but also what parameter values to use for the chosen algorithm. Some parameter settings may be set with the help of background knowledge (*e.g.*, prune more in noisy domains), but often the background knowledge does not translate well into actual settings. For example, there is no clear understanding as to when one splitting criterion is superior to another. While the problem of algorithm selection is recognized as an important issue in machine learning, the problem of finding the best parameter values has not been systematically studied. With many parameters possible, the problem cannot be simply posed as model selection from a few models, but requires an efficient search in the space of parameters. One important problem that falls under parameter tuning is that of feature subset selection: given a dataset with many features, how can we select a "good" subset.

This chapter introduces the wrapper approach to feature subset selection and parameter tuning. We use feature subset selection to study the strengths and weaknesses of the wrapper approach and find ways to improve the original design. We study the relation between feature subset selection and relevance, and introduce a new method to dynamically change the topology of the search space for feature subsets using compound operators. We show how the wrapper approach can be used to tune parameters for C4.5 and discuss some problems with the approach.

4.1 Introduction to Wrappers

`IT IS IN THE PROCESS`: so wrapped up in red tape that the situation is almost hopeless.
—Glossary of important business terms

Suppose you are given a dataset and the C4.5 induction algorithm. How do you tune its parameters? The usual approach is simply to ignore them, *i.e.*, simply run the default settings and pray that Ross Quinlan's default settings are good for your dataset. A better approach is to use some background knowledge and attempt to set them appropriately, but this is usually impossible in practice because background knowledge does not easily translate into parameters. A third approach is to run C4.5 with different settings and estimate the accuracy of different ones, as suggested in Quinlan (1993). Automating this last approach for a large number of parameter settings is the topic of this chapter.

In theory, every possible parameter setting creates a different model, so the problem can be viewed as that of model selection (Linhart & Zucchini 1986). If there are only a few models, as is the case when one chooses between three induction algorithms, one can estimate the accuracy of each one and select the one with the highest accuracy (Schaffer 1993) or perhaps even find some underlying theory to help predict the best one for a given dataset (see Brazdil, Gama & Henery (1994) for an attempt that was not very successful in finding regularities in the StatLog project). When we attempt to tune multiple parameters, the space of possible combinations is usually too big for brute-force enumeration of all possibilities, and we must resort to heuristic search.

The idea behind the wrapper approach, shown in Figure 4.1 is simple: the induction algorithm is considered as a black box with tunable parameters. The induction algorithm is run on the dataset, usually partitioned into internal training and test sets, with different settings of the parameters. The setting with the highest estimated value is chosen as the final parameter set on which to run the induction algorithm. There are two crucial components to the wrapper approach: a search component and an evaluation component. The search component repeatedly suggests parameter settings. The evaluation component evaluates these settings by running the induction algorithm several times and getting an estimate of our objective function, usually accuracy.

Different search algorithms (engines) are discussed in Section 4.4. The evaluation component can be any objective function that takes into account factors such as the accuracy,

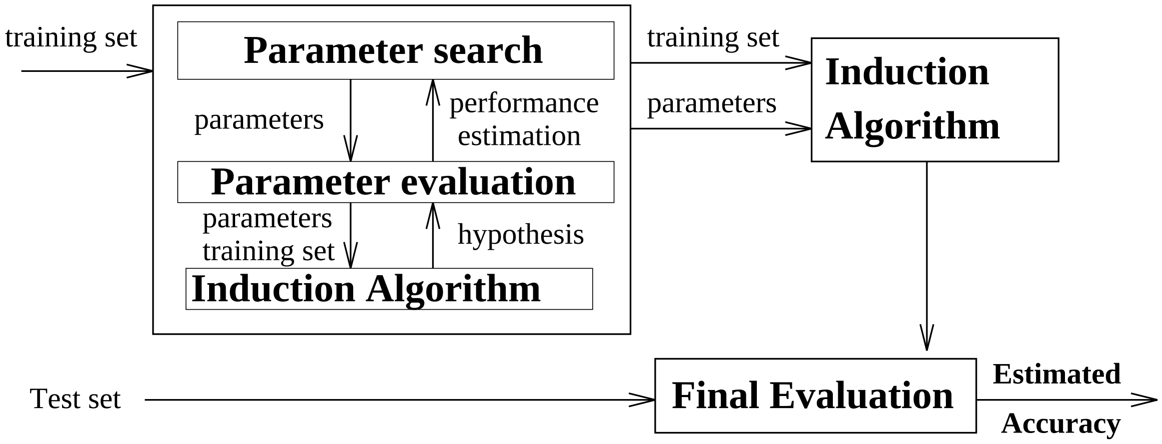

Figure 4.1: The wrapper approach to parameter tuning. The induction algorithm is used as a black box.

comprehensibility, and others. We assume that our goal is to improve accuracy, and whenever there are two models with approximately the same accuracy, we prefer the simpler one (*e.g.*, less features used). For accuracy estimation, any of the approaches described in Chapter 3 can be used; based on our experimental results, we use repeated runs of five-fold cross-validation.

Because datasets vary in size, large datasets tend to have good accuracy estimates while small ones are highly variable. A heuristic we use in all the experiments is to repeat cross-validation until the standard deviation of the mean of the folds (assuming independence between folds) is below 1%, or five runs of five-fold cross-validation have been executed. Since the folds are not independent, this is only a heuristic, but it seems to work well in practice.

This heuristic has the nice property that it forces the accuracy estimation to execute cross-validation more times on small datasets than on large datasets. Because induction algorithms typically run longer on large datasets, the overall accuracy estimation time, which is the product of the induction algorithm running time and the number of cross-validation runs, does not grow too fast. We thus have a conservation of "hardness" using this heuristic: small datasets will be cross-validated many times to overcome the high variance resulting from small amounts of data. For much larger datasets, one could switch to a holdout heuristic to save even more time (a factor of five), but we have not found this necessary for the datasets we used.

This chapter is organized as follows. In Section 4.2, we review the feature subset selection problem, investigate the notion of relevance, define the task of finding optimal features, and describe the filter approach. In Section 4.3, we describe the wrapper approach with its specific instantiation and the experimental setup used in later sections. In Section 4.4, we investigate the search engine used to search for feature subsets and show that greedy search (hill-climbing) is inferior to best-first search. In Section 4.5, we modify the organization of the search space to improve the running time. Section 4.6 contains a global comparison of the best methods found. In Section 4.7, we discuss one problem in the approach, overfitting, and suggest a theoretical model that generalizes the problem in Section 4.8. In Section 4.9, we apply the wrapper to automatically tune the parameters of C4.5. Related work is given in Section 4.10, future work is discussed in Section 4.11, and we conclude with a summary in Section 4.12.

4.2 Feature Subset Selection

It is always easier to argue for a feature than to argue that the advantage of the feature—which will be very plausible in all interesting cases— is outweighed by nebulous concerns of coherence, simplicity, stability, difficulties of transition, etc.
—Stroustroup (1994, page 148)

In this section, we look at a specific type of a parameter tuning problem: finding a good feature subset, which could be thought of as a zero-one parameter for each feature (use or ignore). While this type of parameter may not be explicitly available in all algorithms, it is applicable to all induction algorithms because we can simply hide features in the dataset before passing the dataset to the induction algorithm. The issue of optimal feature subsets is discussed and its relation to relevance of features. We show problems with existing definitions of relevance and demonstrate how partitioning relevant features into two families—weak and strong—helps us understand the issue better. We examine two general approaches to feature subset selection—the filter approach and the wrapper approach—and we then investigate the latter in detail.

4.2.1 The Problem

Practical machine learning algorithms, including top-down induction of decision tree algorithms (*e.g.*, ID3, C4.5, CART) and instance-based algorithms (*i.e.*, nearest-neighbor algorithms), are known to degrade in performance (prediction accuracy) when faced with many features that are not necessary for predicting the desired output. Algorithms such as Naive-Bayes are robust with respect to irrelevant features but may degrade in performance if the features are correlated (even if relevant).

For example, running C4.5 in default mode on the Monk1 problem, which has three irrelevant features, generates a tree with 15 interior nodes, five of which test irrelevant features. The generated tree has an error rate of 24.3%, which is reduced to 11.1% if only the three relevant features are given. Aha (1992) noted that "IB3's storage requirement increases exponentially with the number of irrelevant attributes" (IB3 is a nearest-neighbor algorithm that attempts to save only important prototypes). Performance likewise degrades rapidly with irrelevant features.

The problem of feature subset selection is that of finding a subset of the original features of a dataset, such that an induction algorithm that is run on data containing only these features generates a classifier with the highest possible accuracy. Note that feature subset selection chooses a set of features from existing features, and does not construct new ones; there is no feature extraction or construction (Kittler 1986, Rendell & Seshu 1990).

From a purely theoretical standpoint, the question is not of much interest. The optimal Bayes rule is monotonic, *i.e.*, adding features cannot decrease the accuracy, and hence restricting the induction algorithm to a subset of features is never advised. Practical algorithms, however, are not given access to the underlying distribution, and most practical algorithms attempt to fit to the data by solving NP-hard optimization problems. For example, decision tree induction algorithms usually attempt to find a small tree that fits the data well, yet finding the optimal binary decision tree is NP-hard (Hyafil & Rivest 1976, Hancock 1989). For neural-networks, the problem is even harder; the problem of loading a three-node neural network with a training set is NP-hard if the nodes compute linear threshold functions (Judd 1988, Blum & Rivest 1992). Because most induction problems are NP-hard and heuristics are used, we define an optimal feature subset with respect to the induction algorithm. The problem of feature subset selection is then reduced to the problem of finding an optimal subset.

Definition 4.1

Given an inducer $\mathcal{I}$, and a dataset $\mathcal{D}$ with features $X_1, X_2, \ldots, X_n$, from a distribution D over the labelled instance space, an ***optimal feature subset,*** $\vec{X}_{\mathbf{opt}}$, *is a subset of the features such that the accuracy of the classifier $\mathcal{C} = \mathcal{I}(\mathcal{D})$ is maximal.*

An optimal feature subset need not be unique because it may be possible to achieve the same accuracy using different sets of features (*e.g.*, when two features are perfectly correlated, than one can be replaced by the other). In most practical problems, we do not have access to the underlying distribution and must estimate the classifier's accuracy from the data.

4.2.2 Relevance of Features

One important question is the relation between optimal features and "relevance," which is a very loaded term. In this section, we present definitions of relevance that have been suggested in the literature. We then show a single example where the definitions give unexpected answers, and we suggest that two degrees of relevance are needed: weak and strong. We show that relevance of a feature does not imply that it is in the optimal feature subset but that irrelevance generally implies that it should not be in the optimal feature subset.

Existing Definitions

Almuallim & Dietterich (1991, p. 548) define relevance under the assumptions that all features and the label are Boolean and that there is no noise.

Definition 4.2
*A feature X_i is said to be **relevant**$_1$ to a concept C if X_i appears in every Boolean formula that represents C and **irrelevant**$_1$ otherwise.*

Gennari, Langley & Fisher (1989, Section 5.5) define relevance as[1]

Definition 4.3
*X_i is **relevant**$_2$ iff there exists some x_i and y for which $p(X_i = x_i) > 0$ such that*

$$p(Y = y \mid X_i = x_i) \neq p(Y = y) \ .$$

Under this definition, X_i is relevant if knowing its value can change the estimates for Y, or in other words, if Y is conditionally dependent on X_i. Note that this definition fails to

[1]The definition given is a formalization of their statement: "Features are relevant if their values vary systematically with category membership."

Definition	Relevant	Irrelevant
Definition 4.2	X_1	X_2, X_3, X_4, X_5
Definition 4.3	None	All
Definition 4.4	All	None
Definition 4.5	X_1	X_2, X_3, X_4, X_5

Table 4.1: Feature relevance for the Correlated XOR problem under the four definitions.

capture the relevance of features in the parity concept where all unlabelled instances are equiprobable, and it may therefore be changed as follows.

Let S_i be the set of all features except X_i, *i.e.*, $S_i = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m\}$. Denote by s_i a value-assignment to all features in S_i.

Definition 4.4
*X_i is **relevant**$_3$ iff there exists some x_i, y, and s_i for which $p(X_i = x_i) > 0$ such that*

$$p(Y = y, S_i = s_i \mid X_i = x_i) \neq p(Y = y, S_i = s_i) \; .$$

Under the following definition, X_i is relevant if the probability of the label (given all features) can change when we eliminate knowledge about the value of X_i.

Definition 4.5
*X_i is **relevant**$_4$ iff there exists some x_i, y, and s_i for which $p(X_i = x_i, S_i = s_i) > 0$ such that*

$$p(Y = y \mid X_i = x_i, S_i = s_i) \neq p(Y = y \mid S_i = s_i) \; .$$

The following example shows that all the definitions above give unexpected results.

Example 4.1 (Correlated XOR) Let features $X_1, \ldots, X_5$ be Boolean. The instance space is such that X_2 and X_3 are negatively correlated with X_4 and X_5, respectively, *i.e.*, $X_4 = \overline{X_2}$, $X_5 = \overline{X_3}$. There are only eight possible instances, and we assume they are equiprobable. The (deterministic) target concept is

$$Y = X_1 \oplus X_2 \qquad (\oplus \text{ denotes XOR}) \; .$$

Note that the target concept has an equivalent Boolean expression, namely, $Y = X_1 \oplus \overline{X_4}$. The features X_3 and X_5 are irrelevant in the strongest possible sense. X_1

is indispensable, and either but not both of $\{X_2, X_4\}$ can be disposed of. Table 4.1 shows for each definition, which features are relevant, and which are not.

According to Definition 4.2, X_3 and X_5 are clearly irrelevant; both X_2 and X_4 are irrelevant because each can be replaced by the negation of the other. By Definition 4.3, all features are irrelevant because for any output value y and feature value x, there are two instances that agree with the values. By Definition 4.4, every feature is relevant because knowing its value changes the probability of four of the eight possible instances from 1/8 to zero. By Definition 4.5, X_3 and X_5 are clearly irrelevant, and both X_2 and X_4 are irrelevant because they do not add any information to S_4 and S_2, respectively. ∎

Although such simple negative correlations are unlikely to occur, domain constraints create a similar effect. When a nominal feature such as color is encoded as input to a neural network, it is customary to use a *local encoding*, where each value is represented by an indicator feature. For example, the local encoding of a four-valued nominal $\{a, b, c, d\}$ would be $\{0001, 0010, 0100, 1000\}$ (see also Section 2.4). Under such an encoding, any single indicator feature is redundant and can be determined by the rest. Thus most definitions of relevance will declare all indicator features to be irrelevant.

Strong and Weak Relevance

We now claim that two degrees of relevance are required: weak and strong. Relevance should be defined in terms of a Bayes classifier—the optimal classifier for a given problem. A feature X is **strongly relevant** if removal of X alone will result in performance deterioration of an optimal Bayes classifier. A feature X is **weakly relevant** if it is not strongly relevant and there exists a subset of features, S, such that the performance of a Bayes classifier on S is worse than the performance on $S \cup \{f\}$. A feature is **irrelevant** if it is not strongly or weakly relevant.

Definition 4.5 repeated below defines strong relevance. Strong relevance implies that the feature is indispensable in the sense that it cannot be removed without loss of prediction accuracy. Weak relevance implies that the feature can sometimes contribute to prediction accuracy.

Definition 4.5 (Strong relevance)
*A feature X_i is **strongly relevant** iff there exists some x_i, y, and s_i for which $p(X_i =$*

$x_i, S_i = s_i) > 0$ *such that*

$$p(Y = y \mid X_i = x_i, S_i = s_i) \neq p(Y = y \mid S_i = s_i) \ .$$

Definition 4.6 (Weak relevance)
*A feature X_i is **weakly relevant** iff it is not strongly relevant, and there exists a subset of features S_i' of S_i for which there exists some x_i, y, and s_i' with $p(X_i = x_i, S_i' = s_i') > 0$ such that*

$$p(Y = y \mid X_i = x_i, S_i' = s_i') \neq p(Y = y \mid S_i' = s_i') \ .$$

A feature is **relevant** if it is either weakly relevant or strongly relevant; otherwise, it is **irrelevant**. Borrowing some terminology from rough sets (Pawlak 1991, Slowinski 1992), the set of strongly relevant features form the **core** and any set of features that allow a Bayes classifier to achieve the highest possible accuracy forms a **reduct.** A reduct can only contain strongly relevant and weakly relevant features. Pawlak (1991) shows that the core is the intersection of all the reducts and that every reduct consists only of the core features and weakly relevant features.

In Example 4.1, feature X_1 is strongly relevant; features X_2 and X_4 are weakly relevant; and X_3 and X_5 are irrelevant.

4.2.3 Relevance and Optimality of Features

A Bayes classifier must use all strongly relevant features (the core), and possibly some weakly relevant features. Classifiers induced from data, however, are not optimal, as they have no access to the underlying distribution; furthermore, they may be using restricted hypothesis spaces that cannot utilize all features (see example below). Practical induction algorithms that generate classifiers may benefit from the omission of features, including strongly relevant features.

Example 4.2 (Relevance and Optimality) Let the universe of possible instances be $\{0,1\}^3$, that is, three Boolean features, say X_1, X_2, X_3. Let the distribution over the universe be uniform, and assume the target concept is $f(X_1, X_2, X_3) = (X_1 \wedge X_2) \vee X_3$. Under any reasonable definition of relevance, all features are relevant to this target function.

Figure 4.2: The feature filter approach, in which the features are filtered independently of the induction algorithm.

If the hypothesis space is the space of monomials, *i.e.*, conjunctions of literals, the only optimal feature subset is $\{X_3\}$. The accuracy of the monomial X_3 is 87.5%, the highest accuracy achievable within this hypothesis space. ∎

The example above shows that relevance (even strong relevance) does not imply that a feature is in an optimal feature subset. Another question is whether an irrelevant feature can ever be in an optimal feature subset. The following trivial example shows that this may be true.

Example 4.3 (Buggy Inducer) Consider an inducer `Buggy-ind` that induces a classifier. Due to a bug, if feature number one is not given, the inducer produces a classifier that labels instances with label zero; otherwise, a more sensible classifier is produced. Even if feature one is totally irrelevant, it must appear in any optimal feature subset in order for the classifier to predict anything other than class zero. ∎

We hope that cases such as the example above are extremely rare in practice. In general, a totally irrelevant feature should not be in the optimal feature subset for an algorithm.

4.2.4 The Filter Approach

There are a number of different approaches to subset selection. In this section, we review existing approaches in machine learning. We refer the reader to Section 4.10 for related work in statistics and pattern recognition. The reviewed methods for feature subset selection follow the *filter approach* and attempt to assess the merits of features from the data, ignoring the induction algorithm.

The filter approach, shown in Figure 4.2, selects features using a preprocessing step. The main disadvantage of the filter approach is that it totally ignores the effects of the selected feature subset on the performance of the induction algorithm. We now review some existing algorithms that fall into the filter approach.

The FOCUS Algorithm

The FOCUS algorithm (Almuallim & Dietterich 1991, Almuallim & Dietterich 1994), originally defined for noise-free Boolean domains, exhaustively examines all subsets of features, selecting the minimal subset of features that is sufficient to determine the label value for all instances in the training set. This preference for a small set of features is referred to as the MIN-FEATURES bias.

This bias has severe implications when applied blindly without regard for the resulting induced concept. For example, in a medical diagnosis task, a set of features describing a patient might include the patient's social security number (SSN). (We assume that features other than SSN are sufficient to determine the correct diagnosis.) When FOCUS searches for the minimum set of features, it will pick the SSN as the only feature needed to uniquely determine the label[2]. Given only the SSN, any induction algorithm is expected to generalize very poorly.

The Relief Algorithm

The Relief algorithm (Kira & Rendell 1992*a*, Kira & Rendell 1992*b*, Kononenko 1994) assigns a "relevance" weight to each feature, which is meant to denote the relevance of the feature to the target concept. Relief is a randomized algorithm. It samples instances randomly from the training set and updates the relevance values based on the difference between the selected instance and the two nearest instances of the same and opposite class (the "near-hit" and "near-miss").

The Relief algorithm finds all weakly relevant features:

> Relief does not help with redundant features. If most of the given features are relevant to the concept, it would select most of them even though only a fraction are necessary for concept description (Kira & Rendell 1992*a*, page 133).

In real domains, many features have high correlations with the label, and thus many are (weakly) relevant, and will not be removed by Relief. In the simple parity example used

[2]This is true even if SSN is encoded in ℓ binary features as long as more than ℓ other features are required to determine the diagnosis.

in (Kira & Rendell 1992*a*, Kira & Rendell 1992*b*), there were only strongly relevant and irrelevant features, so Relief found the strongly relevant features most of the time. While nearest-neighbors are not hurt much by weakly relevant features, Naive-Bayes is affected. The Relief algorithm was motivated by nearest-neighbors and it is good specifically for similar types of induction algorithms.

Feature Filtering Using Decision Trees

Cardie (1993) used a decision tree algorithm to select a subset of features for a nearest-neighbor algorithm. Since a decision tree typically contains only a subset of the features, those that appeared in the final tree were selected for the nearest-neighbor. The decision tree thus serves as the filter for the nearest-neighbor algorithm.

Although the approach worked well for some datasets, it has some major shortcomings. Features that are good for decision trees are not necessarily useful for nearest-neighbor. As with Relief, one expects that the totally irrelevant features will be weeded, and this is probably the major effect that led to some improvements in the datasets experimented with. However, while a nearest-neighbor algorithm can take into account the effect of many relevant features, the current methods of building decision trees suffer from data fragmentation and not too many splits can be made before the number of instances is exhausted. If the tree is approximately balanced and the number of training instances that trickles down to each subtree is approximately the same, then a decision tree cannot test more than $O(\log_2 m)$ features in a path.

Summary of Filter Approaches

Figure 4.3 shows the set of features that FOCUS and Relief search for. While FOCUS is searching for a minimal set of features, Relief searches for all the relevant features (both weak and strong).

Filter approaches to feature subset selection do not take into account the biases of the induction algorithms and select feature subsets that are independent of the induction algorithms. In some cases, measures can be devised that are algorithm specific, and these may be computed efficiently. For example, in linear regression, measures such as Mallow's C_p (Mallows 1973) and PRESS (Prediction sum of squares) (Neter, Wasserman & Kutner 1990) have been devised so that they do not require running the regression many times, and thus avoid the cross-validation step used in the default wrapper setup. These measures

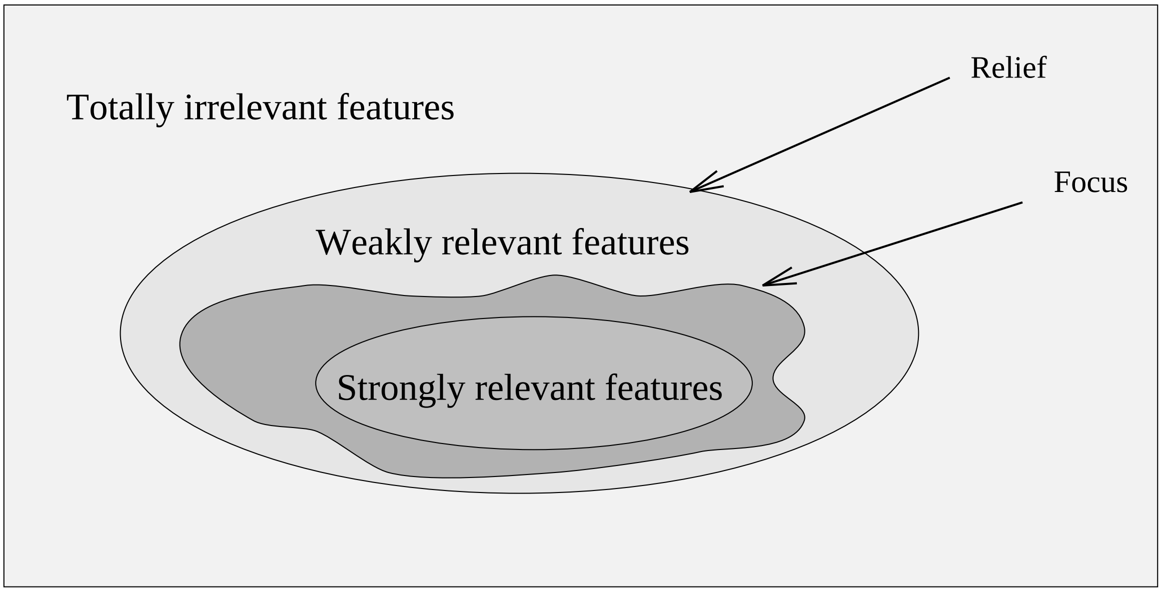

Figure 4.3: A view of feature set relevance.

and the Relief measure, for example, would not be appropriate as feature subset selectors for Naive-Bayes.

The corral dataset, which is an artificial dataset we invented, gives a possible scenario, where filter-approaches fail miserably. We repeat the description from Section 2.4. There are 32 instances in this Boolean domain. The target concept is

$$(A0 \wedge A1) \vee (B0 \wedge B1) .$$

The feature named "irrelevant" is uniformly random, and the feature "correlated" matches the class label 75% of the time (for specific instances). Greedy strategies for building decision trees pick the "correlated" feature as it seems best by all known selection criteria. After the wrong root split, the instances are fragmented and there are not enough instances at each subtree to describe the correct concept. Figure 4.4 shows the decision tree induced by C4.5. CART induces a similar decision tree with the "correlated" feature at the root. When this feature is removed, the correct tree is found.

Because the "correlated" feature is really correlated with the label (in fact, very highly correlated), then filter algorithms will generally select it. Wrapper approaches, on the other

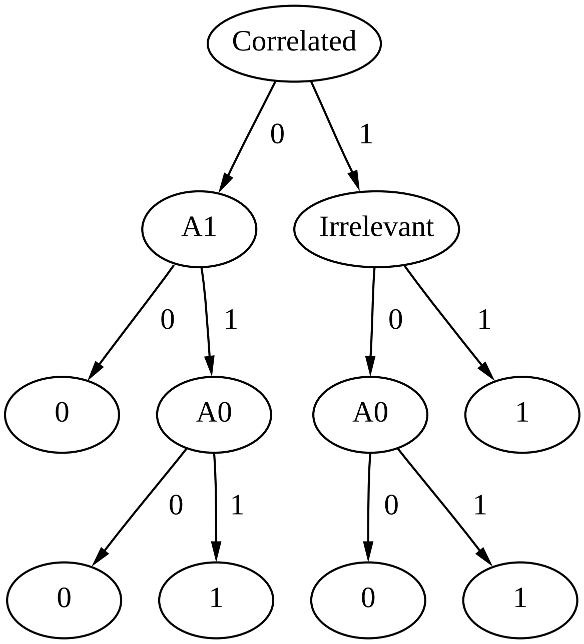

Figure 4.4: The "Corral" datasets fools top-down decision-tree algorithms into picking the "correlated" feature for the root, causing fragmentation, which in turns causes the irrelevant feature to be chosen.

hand, may discover that the feature is hurting performance and will avoid selecting it.

4.3 The Wrapper Approach to Feature Subset Selection

If variable elimination has not been sorted out after two decades of work assisted by high-speed computing, then perhaps the time has come to move on to other problems.
—R. L. Plackett, discussion in Miller (1984)

In the wrapper approach, shown in Figure 4.5, the feature subset selection is done using the induction algorithm as a black box (*i.e.*, no knowledge of the algorithm is needed, just the interface). The feature subset selection algorithm conducts a search for a good subset using the induction algorithm itself as part of the evaluation function. The accuracy of the induced classifiers is estimated using accuracy estimation techniques as described in Chapter 3. The problem we are investigating is that of state space search, and different search engines will be investigated in the next sections.

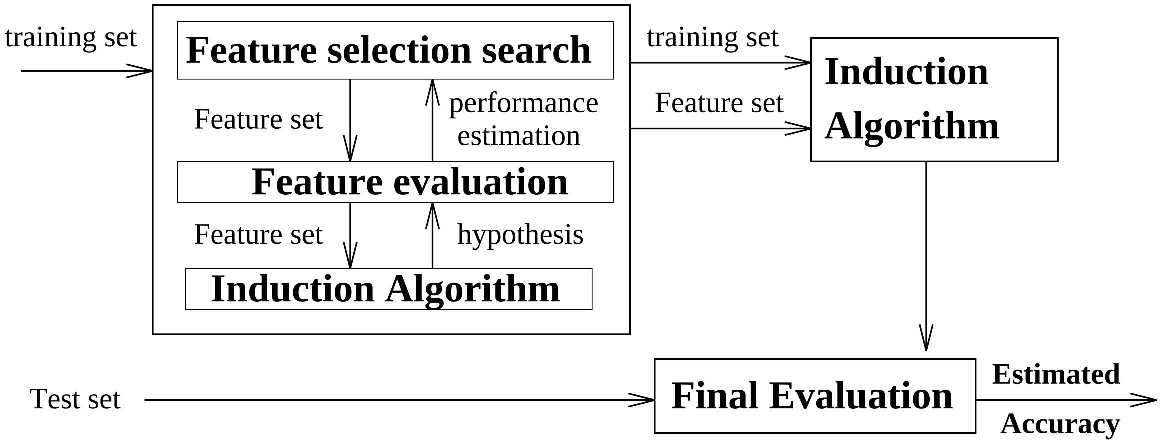

Figure 4.5: The wrapper approach to feature subset selection. The induction algorithm is used as a "black box" by the subset selection algorithm.

The wrapper approach conducts a search in the space of possible parameters. A search requires a state space, an initial state, a termination condition, and a search engine (Ginsberg 1993, Russell & Norvig 1995). The next section focuses on comparing search engines, that is, the way we search the space of possible parameters. The features are used as adjustable parameters, *i.e.*, whether a feature is shown to the algorithm or not. The feature subset selection problem will thus serve as our first testbed for comparing search algorithms.

The search space organization that we chose is such that each state represents a feature subset. For n features, there are n bits in each state, and each bit indicates whether a feature is present (1) or absent (0).

Operators determine the partial ordering between the states, and we have chosen to use operators that add or delete a single feature from a state, corresponding to the search space commonly used in the stepwise methods in statistics. Figure 4.6 shows such the state space and operators for a four-feature problem. The size of the search space for n features is $O(2^n)$, so it is impractical to search the whole space exhaustively, unless n is small. We will shortly describe the different search engines that we compared.

The goal of the search is to find the state with the highest evaluation, using a heuristic function to guide it. Since we do not know the actual accuracy of the induced classifier, we use accuracy estimation as both the heuristic function and the evaluation function (See Section 4.8 for more details on the abstract problem). The evaluation function we use

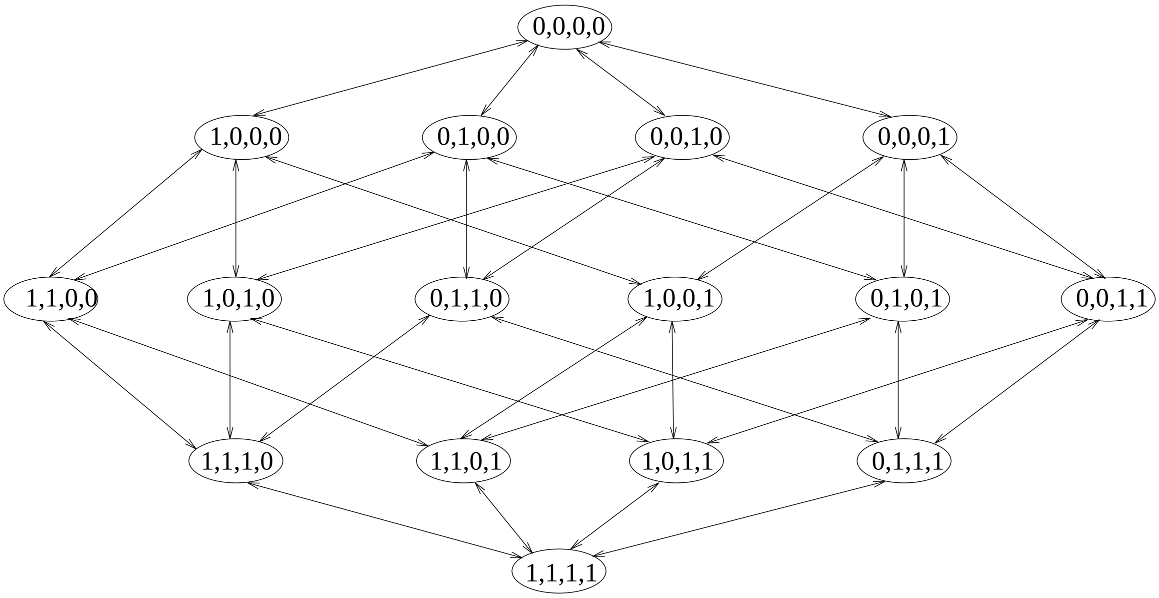

Figure 4.6: The state space search for feature subset selection. Each node is connected to nodes that have one feature deleted or added.

is five-fold cross-validation, which is repeated multiple times. The number of repetitions is determined on the fly by looking at the standard deviation of the accuracy estimate, assuming they are independent. If the standard deviation of the accuracy estimate is above 1% and five cross-validations have not been executed, we execute another cross-validation run. While this is only a heuristic, it seems to work well in practice and avoids multiple cross-validation runs for large datasets.

The term **forward selection** refers to a search that begins at the empty set of features; the term **backward elimination** refers to a search that begins at the full set of features. The initial state we use in most of our experiments is the empty set of features, hence we are using a forward selection approach. The main reason for this choice is computational: building classifiers when there are few features in the data is much faster. Although in theory, going backward from the full set of features may capture interacting features more easily, the method is extremely expensive with only the add feature and delete feature operators. In Section 4.5, we will introduce compound operators that will make the backward elimination approach practical. The following summary shows the instantiation of the search instance:

State	A Boolean vector, one bit per feature
Initial state	The empty set of features (0,0,0..., 0)
Heuristic/evaluation	Five-fold cross-validation repeated multiple times with a small penalty (0.1%) for every feature.
Search algorithm	Hill-climbing or best-first search
Termination condition	Algorithm dependent (see below)

A **complexity penalty** was added to the evaluation function, penalizing feature subsets with many features so as to break ties in favor of smaller subsets. The penalty was set to 0.1%, which is very small compared to the standard deviation of the accuracy estimation, which is aimed to be below 1%.

4.4 The Search Engine

You need not fash yourself anymore about that, man; I have now made an engine that shall not waste a particle of steam.
—James Watt, 1765

In this section we evaluate different search engines for the wrapper approach. We begin with a hill-climbing (greedy) search engine, and show that it terminates at local maxima too often. We then use a best-first search engine and show that it works much better.

4.4.1 A Hill-climbing Search Engine

The simplest search technique is hill-climbing, also called greedy search or steepest ascent. Table 4.2 describes the algorithm, which expands the current node and moves to the child with the highest accuracy, terminating when no child improves over the current node.

Table 4.3 and Figures 4.7 and 4.8 show a comparison of ID3 and Naive-Bayes, both with and without feature subset selection. Table 4.4 and Figure 4.9 and 4.10 show the average number of features used for each algorithm (averaged over the ten folds when relevant). The following observations can be made:

- For the real datasets and ID3, this simple version of feature subset selection provides an indirect pruning mechanism. By hiding features from ID3, the leaves cannot be made pure. This type of pruning is global in the sense that a feature is either present or absent, but it cannot be present at one subtree and not at another. As shown in

Table 4.2: A hill-climbing search algorithm

1. Let v $\leftarrow$ initial state.
2. Expand v: apply all operators to v, giving v's children.
3. Evaluate the children of v, giving $f(w_i)$ for each child w_i.
4. Let w = $\arg\max_{w_i} f(w_i)$ (get the child with highest evaluation).
5. If $w > v$ then $v \leftarrow w$; goto 2.
6. Return v.

Table 4.3: A comparison of ID3 and Naive-Bayes with a feature subset selection wrapper. The first p-val column indicates the probability that feature subset selection (FSS) improves ID3 and the second column indicates the probability that FSS improves Naive-Bayes (see Section 2.6 for a description of p-values).

	Dataset	ID3	ID3-FSS	p-val	Naive-Bayes	NB-FSS	p-val
1	breast cancer	94.57± 0.9	94.71± 0.5	0.58	97.00± 0.5	96.57± 0.6	0.22
2	cleve	72.35± 2.3	78.24± 2.0	1.00	82.88± 2.3	79.56± 3.9	0.15
3	crx	81.16± 1.4	85.65± 1.6	1.00	87.10± 0.8	85.36± 1.6	0.08
4	DNA	90.64± 0.9	94.27± 0.7	1.00	93.34± 0.7	94.52± 0.7	0.96
5	horse-colic	81.52± 2.0	83.15± 1.1	0.84	79.86± 2.5	83.15± 2.0	0.93
6	Pima	68.73± 2.5	69.52± 2.2	0.63	75.90± 1.8	74.34± 2.0	0.21
7	sick-euthyroid	96.68± 0.6	97.06± 0.5	0.76	95.64± 0.6	97.35± 0.5	1.00
8	soybean-large	90.62± 0.9	90.77± 1.1	0.56	91.80± 1.2	92.38± 1.1	0.69
9	corral	100.00± 0.0	75.00± 3.8	0.00	90.62± 2.6	75.00± 3.8	0.00
10	m-of-n-3-7-10	91.60± 0.9	77.34± 1.3	0.00	86.43± 1.1	77.34± 1.3	0.00
11	Monk1	82.41± 1.8	75.00± 2.1	0.00	71.30± 2.2	75.00± 2.1	0.96
12	Monk2-local	82.41± 1.8	67.13± 2.3	0.00	60.65± 2.3	67.13± 2.3	1.00
13	Monk2	69.68± 2.2	67.13± 2.3	0.13	61.57± 2.3	67.13± 2.3	0.99
14	Monk3	90.28± 1.4	97.22± 0.8	1.00	97.22± 0.8	97.22± 0.8	0.50
	Average real:	84.53	86.67		87.94	87.90	
	Average artif.	86.06	76.47		77.96	76.47	

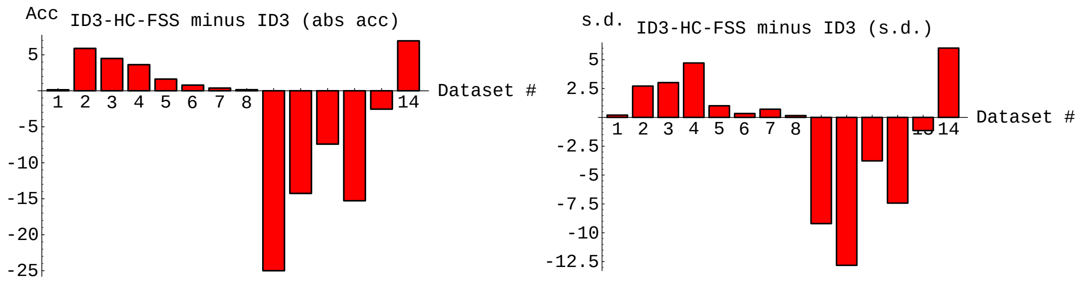

Figure 4.7: ID3: Absolute difference (FSS minus ID3) in accuracy (left) and in std-devs (right).

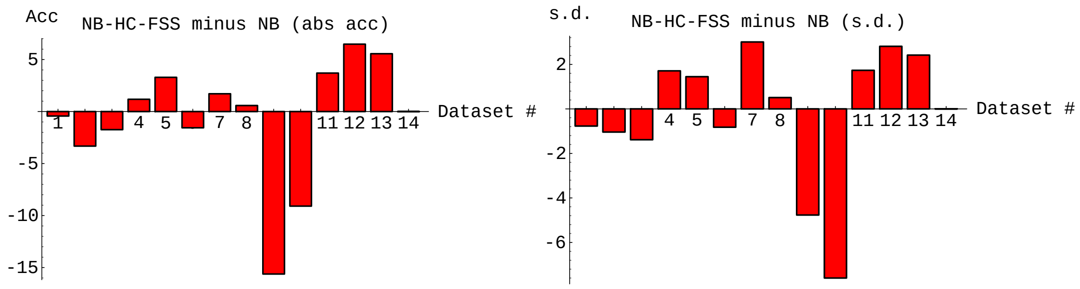

Figure 4.8: Naive-Bayes: Absolute difference in accuracy (left) and in std-devs (right).

Table 4.4 and Figures 4.9 and 4.10, the number of features selected is small compared to the original set and compared to those selected by ID3. For ID3, the accuracy goes up from 84.53% to 86.67%, which is a 13.8% relative reduction in the error rate. The accuracy uniformly improves or remains the same for all the real datasets.

- For the artificial datasets and ID3, the story is different. All the artificial datasets, except Monk3 involve high-order interactions. In the corral dataset, after the correlated feature is chosen, no single addition of a feature will lead to an improvement, so the hill-climbing process stops too early; similar scenarios happen with the other artificial datasets, where adding a single feature at a time does not help.

  The concept for Monk3 is

$$\begin{gathered}(\text{jacket-color} = \text{green and holding} = \text{sword}) \text{ or}\\(\text{jacket-color} \neq \text{blue and body-shape} \neq \text{octagon})\end{gathered}$$

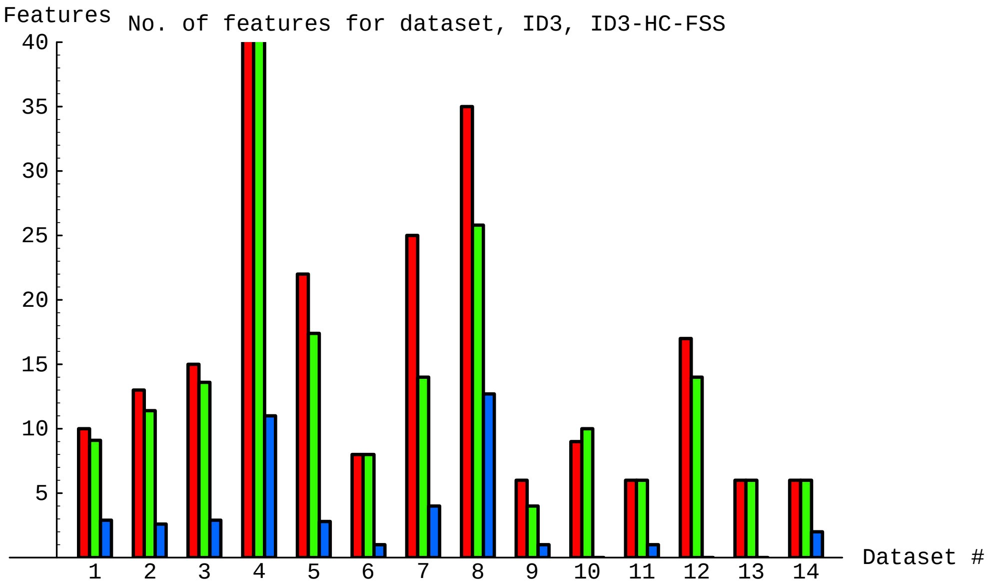

Figure 4.9: ID3: Number of features in original dataset (left), used by ID3 (middle), and selected by hill-climbing feature subset selection (right). The DNA has 180 features (not shown).

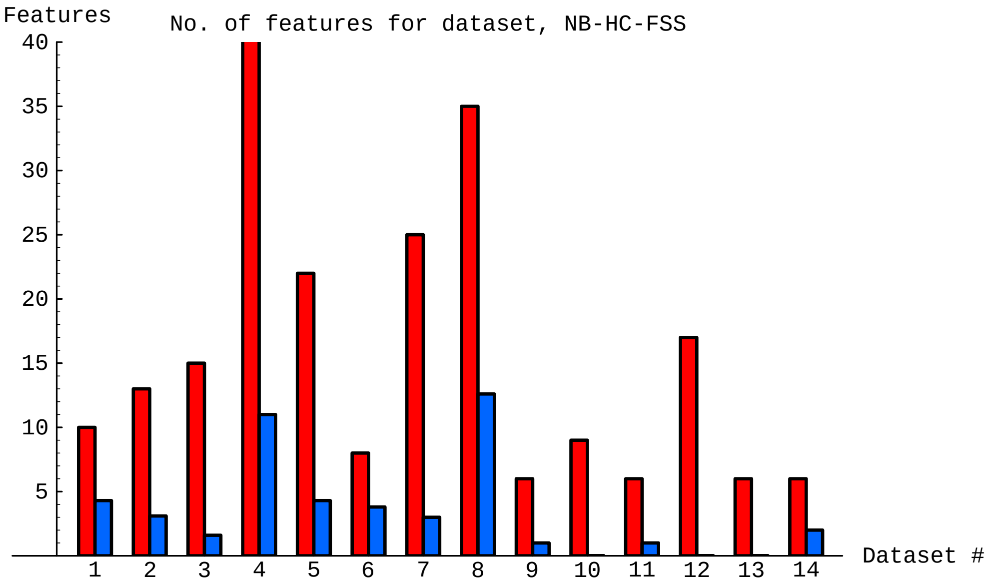

Figure 4.10: Naive-Bayes: Number of features in original dataset (left) and selected by hill-climbing feature subset selection (right).

Table 4.4: The number of features in the dataset, the number used by ID3 (since it does some feature subset selection), the number selected by feature subset selection (FSS) for ID3, and the number selected by FSS for Naive-Bayes. Numbers without a decimal point are for single runs, number with a decimal point are averages for the ten-fold cross-validation.

	Dataset	Number of Features			
		Original Dataset	ID3	ID3-FSS	NB-FSS
1	breast cancer	10	9.1	2.9	4.3
2	cleve	13	11.4	2.6	3.1
3	crx	15	13.6	2.9	1.6
4	DNA	180	72	11	11
5	horse-colic	22	17.4	2.8	4.3
6	Pima	8	8.0	1.0	3.8
7	sick-euthyroid	25	14	4	3
8	soybean-large	35	25.8	12.7	12.6
9	corral	6	4	1	1
10	m-of-n-3-7-10	10	10	0	0
11	Monk1	6	6	1	1
12	Monk2-local	17	14	0	0
13	Monk2	6	6	0	0
14	Monk3	6	6	2	2

and the data contains 5% mislabelled instances. The feature subset selection algorithm quickly finds body-shape and jacket-color, which together yield the second conjunction in the above expression, which has accuracy 97.2%. With more features, a larger tree is built which is inferior.

- For the real datasets and Naive-Bayes, the average accuracy is about same, but very few features are used.

- For the artificial datasets and Naive-Bayes, the average accuracy degrades because of corral and m-of-n-3-7-10 (the relative error increases by 6.7%). Both of these require a better search than hill climbing can provide. An interesting observation is the fact that the performance on the Monk2 and Monk2-local datasets improves simply by hiding all features, forcing Naive-Bayes to predict the majority class. The independence assumption is so wrong for this dataset that it is better to predict the majority class.

- For the DNA dataset, both algorithms selected only 11 features out of 180. While the selected set differed, nine features were the same, indicating that these nine are crucial for both types of inducers.

The results, especially on the artificial datasets where we know what the relevant feature are, indicate that the feature subset selection is getting stuck at local maxima too often. The next section deals with improving the search engine.

4.4.2 A best-first Search Engine

Best-first search (Russell & Norvig 1995, Ginsberg 1993) is a more robust method than hill-climbing. The idea is to select the most promising node we have generated so far that has not already been expanded. Table 4.5 describes the algorithm, which varies slightly from the standard version because there is no explicit goal condition in our problem. Best-first search usually terminates upon reaching the goal. Our problem is an optimization problem, so the search can be stopped at any point and the best solution found so far can be returned (theoretically improving over time), thus making it an anytime algorithm (Boddy & Dean 1989). In practice, we must stop the run at some stage, and we use what we call a **stale search:** if we have not found a new best node in the last k expansions, we terminate the search. A new best node is defined as a node with an accuracy estimation at least ϵ higher than the best one found so far. In the following experiments, k was set to five and epsilon was 0.1%.

Table 4.6 and Figures 4.11 and 4.12 show a comparison of ID3 and Naive-Bayes, both with hill-climbing feature subset selection and best-first search feature subset selection. Table 4.7 shows the average number of features used for each algorithm (averaged over the ten folds when relevant). The following observations can be made:

- For the real datasets and both algorithms (ID3 and Naive-Bayes), there is almost no difference between hill climbing and best-first search. Best-first search usually finds a larger feature subset, but the accuracies are approximately the same. The only statistically significant difference is for Naive-Bayes and soybean, where there was a significant improvement with a p-value of 0.95.

- For the artificial datasets, there is a significant improvement for ID3. Performance drastically improves on three datasets (corral, Monk1, Monk2-local), remains the same

Table 4.5: The best-first search algorithm

1. Put the initial state on the OPEN list, CLOSED list $\leftarrow \emptyset$, BEST $\leftarrow$ initial state.
2. Let $v = \arg\max_{w_i \in \text{OPEN}} f(w_i)$ (get the state from OPEN with maximal $f(v)$).
3. Remove v from OPEN, add v to CLOSED.
4. If $f(v) - \epsilon > f(\text{BEST})$, then BEST $\leftarrow v$.
5. Expand v: apply all operators to v, giving v's children.
6. For each child not in the CLOSED list, evaluate and add to the OPEN list.
7. If BEST changed in the last k expansions, goto 2.
8. Return BEST.

on two (m-of-n-3-7-10, Monk3), and degrades on only one (Monk2). Analyzing the selected features, the optimal feature subset was found for corral, Monk1, Monk2-local, and Monk3 (only two features out of the three relevant ones were selected for Monk3 because this correctly led to better prediction accuracy).

The search was unable to find the seven relevant features in m-of-n-3-7-10. Because of the complexity penalty of 0.1% for extra features, only subsets of two features were tried, and such subsets never improved over the majority prediction (ignoring all features) before the search was considered stale (five non-improving node expansions). The local maxima in this dataset is too large for the current setting of best-first search to overcome. A specific experiment was conducted to determine how long it would take best-first search to find the correct feature subset. The stale limit (originally set to five) was increased until a node better than the node using zero features, and predicting the majority label value, was found. The first stale setting that overcame the local maximum was 29 (any number above would do). At this setting, a node with three features from the seven is found that is more accurate than majority. Nine more node expansions lead to the correct feature subset. Overall, 193 nodes were evaluated out of the 1024 possibilities. The total running time to find the correct feature subset was 33 CPU minutes, and the prediction accuracy was 100%.

Table 4.6: A comparison of a hill-climbing search and a best-first search. The first p-val column indicates the probability that best-first search feature subset selection (BFS-FSS) improves hill-climbing feature subset selection (HC-FSS) for ID3 and the second column is analogous but for Naive-Bayes.

	Dataset	ID3		p-val	Naive-Bayes		p-val
		HC-FSS	BFS-FSS		HC-FSS	BFS-FSS	
1	breast cancer	94.71± 0.5	94.57± 0.7	0.41	96.57± 0.6	96.00± 0.6	0.17
2	cleve	78.24± 2.0	79.52± 2.3	0.73	79.56± 3.9	80.23± 3.9	0.57
3	crx	85.65± 1.6	85.22± 1.6	0.39	85.36± 1.6	86.23± 1.0	0.75
4	DNA	94.27± 0.7	94.27± 0.7	0.50	94.52± 0.7	94.60± 0.7	0.55
5	horse-colic	83.15± 1.1	82.07± 1.5	0.21	83.15± 2.0	83.42± 2.0	0.55
6	Pima	69.52± 2.2	68.73± 2.2	0.36	74.34± 2.0	75.12± 1.5	0.67
7	sick-euthyroid	97.06± 0.5	97.06± 0.5	0.50	97.35± 0.5	97.35± 0.5	0.50
8	soybean-large	90.77± 1.1	91.65± 1.0	0.81	92.38± 1.1	93.70± 0.4	0.95
9	corral	75.00± 3.8	100.00± 0.0	1.00	75.00± 3.8	90.62± 2.6	1.00
10	m-of-n-3-7-10	77.34± 1.3	77.34± 1.3	0.50	77.34± 1.3	77.34± 1.3	0.50
11	Monk1	75.00± 2.1	97.22± 0.8	1.00	75.00± 2.1	72.22± 2.2	0.10
12	Monk2-local	67.13± 2.3	95.60± 1.0	1.00	67.13± 2.3	67.13± 2.3	0.50
13	Monk2	67.13± 2.3	63.89± 2.3	0.08	67.13± 2.3	67.13± 2.3	0.50
14	Monk3	97.22± 0.8	97.22± 0.8	0.50	97.22± 0.8	97.22± 0.8	0.50
	Average real:	86.67	86.64		87.90	88.33	
	Average artif.	76.47	88.55		76.47	78.61	

In the Monk2 dataset, a set of three features were chosen, and accuracy significantly degraded compared to hill-climbing, which selected the empty feature subset. This is the only significant accuracy difference where performance degraded because best-first search was used (p-value of 0.08). The Monk2 concept in this encoding is unsuitable for decision trees, as a correct tree (built from the full space) contains 439 nodes and 296 leaves. Because the standard training set contains only 169 instances, it is impossible to build the correct tree using the standard recursive partitioning techniques.

- For the artificial datasets, there was a significant improvement for Naive-Bayes only for corral (p-value of 1.00), and performance significantly degraded for Monk1 (p-value of 0.10). The rest of the datasets were unaffected.

  The chosen feature subset for corral contained features A_0, A_1, B_0, B_1, and the "correlated" feature. It is known that only the first four are needed, yet because of the

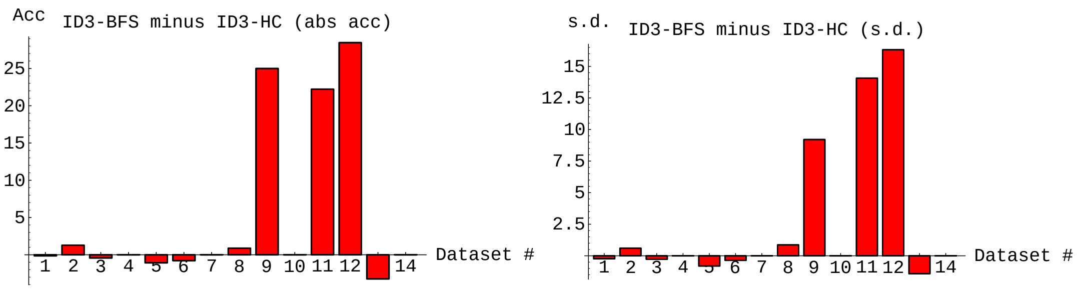

Figure 4.11: ID3: Absolute difference (best-first search FSS minus hill-climbing FSS) in accuracy (left) and in std-devs (right).

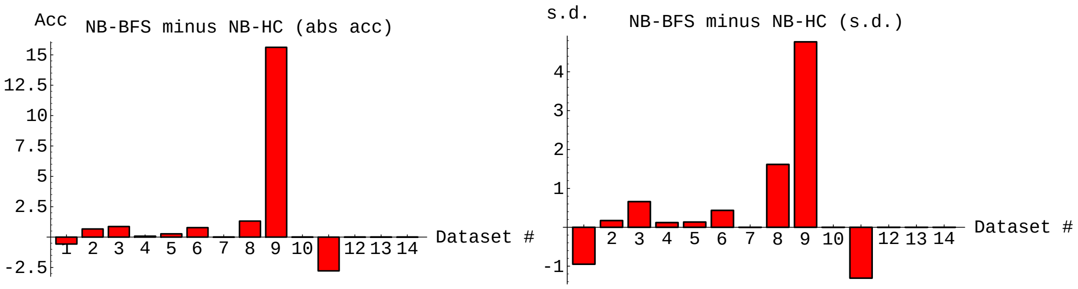

Figure 4.12: Naive-Bayes: Absolute difference in accuracy (left) and in std-devs (right).

limited representation power of the Naive-Bayes, performance using the "correlated" feature is better than performance using only the first four features. If Naive-Bayes is given access only to the first four features, the accuracy degrades from 90.62% to 87.50%. This dataset is one example where the feature subset for different induction algorithms is known to be different. Decision trees are hurt by the addition of the "correlated" feature (performance degrades), yet Naive-Bayes improves with this feature.

The Monk1 dataset degrades in performance because the features head-shape, body-shape, is-smiling, and jacket-color were chosen, yet performance is better if only jacket-color is used. Note that both head-shape and body-shape are part of the target concept, yet the representation power of Naive-Bayes is again limited and cannot utilize this information well. As with the Monk2 dataset for ID3, this may be an example of the search overfitting in the sense that some subset seems to slightly

Table 4.7: The number of features in the dataset, the number used by ID3 (since it does some feature subset selection), the number selected by hill-climbing FSS for ID3, best-first search FSS for ID3, and analogously for Naive-Bayes.

	Dataset	Number of Features					
		Original dataset	ID3	ID3-FSS		NB-FSS	
				HC	BFS	HC	BFS
1	breast cancer	10	9.1	2.9	3.6	4.3	5.2
2	cleve	13	11.4	2.6	3.4	3.1	3.6
3	crx	15	13.6	2.9	3.6	1.6	5.9
4	DNA	180	72	11	11	11	14
5	horse-colic	22	17.4	2.8	3.4	4.3	5.1
6	Pima	8	8.0	1.0	2.3	3.8	4.0
7	sick-euthyroid	25	14	4	4	3	3
8	soybean-large	35	25.8	12.7	13.7	12.6	13.8
9	corral	6	4	1	4	1	5
10	m-of-n-3-7-10	10	10	0	0	0	0
11	Monk1	6	6	1	3	1	4
12	Monk2-local	17	14	0	6	0	0
13	Monk2	6	6	0	3	0	0
14	Monk3	6	6	2	2	2	2

improve the accuracy estimation, but not the accuracy on the independent test set (see Section 4.7 for further discussion on issues of overfitting).

The datasets m-of-n-3-7-10, Monk2-local, Monk2, and Monk3, all had the same accuracy with best-first search as with hill-climbing. The Monk3 dataset cannot be improved by any other feature subset. As with ID3, the search was unable to find a good feature subset for m-of-n-3-7-10 (the correct feature subset allows improving the accuracy to 87.5%). For the Monk2 and Monk2-local datasets, the optimal feature subset is indeed the empty set! Naive-Bayes on the set of relevant features yields inferior performance to a majority inducer, which is how Naive-Bayes behaves on the empty set of features.

While best-first search is better than hill-climbing, high-level interactions occuring in m-of-n-3-7-10 cannot be caught with a search that starts at the empty feature subset unless the stale parameter is drastically increased. An alternative approach, which suffers less from feature interaction, starts with the full set of features; however, the running time would make the approach infeasible in practice, especially if there are many features. The current running times range from about 5-10 minutes of CPU time for small problems such as Monk1, Monk2, Monk3, and corral, to 15 hours for DNA. In the next section, we attempt to reorder the search space dynamically to allow the search to reach better nodes faster and make the backward feature subset selection feasible.

4.5 The State Space: Compound Operators

If we try to gild the lily by using both options together...
—Quinlan (1993)

In the previous section, we looked at two search engines. In this section, we look at the topology of the state space and dynamically modify it based on accuracy estimation results. As previously described, the state space is commonly organized such that each node represents a feature subset, and each operator represents the addition or deletion of a feature. The main problem with this organization is that the search must expand (*i.e.*, generate successors of) every node from the initial feature subset that is on the path to the best feature subset. This section introduces a new way to change the search space topology by creating dynamic operators that directly connect to nodes considered promising given the evaluation of their children. These operators better utilize the information available in the evaluated children.

The motivation for compound operators comes from Figure 4.13 that partitions the feature subsets into core features (strongly relevant), weakly relevant features, and irrelevant features. An optimal feature subset for a hypothesis space must be from the relevant feature subset (strongly and weakly relevant features). A backward elimination search starting from the full set of features (as depicted in Figure 4.13) and that removes one feature at a time after expanding all children reachable using one operator, will have to expand all the children of each node before removing a single feature. If there are i irrelevant features and f features, $(i \cdot f)$ nodes must be evaluated. Similar reasoning applies to forward selection search starting from the empty set of features. In domains where feature subset selection might be most useful, there are many features but such a search may be prohibitively expensive.

Compound operators are operators that are dynamically created *after* the standard set of children, created by the add and delete operators, have been evaluated. Intuitively, there

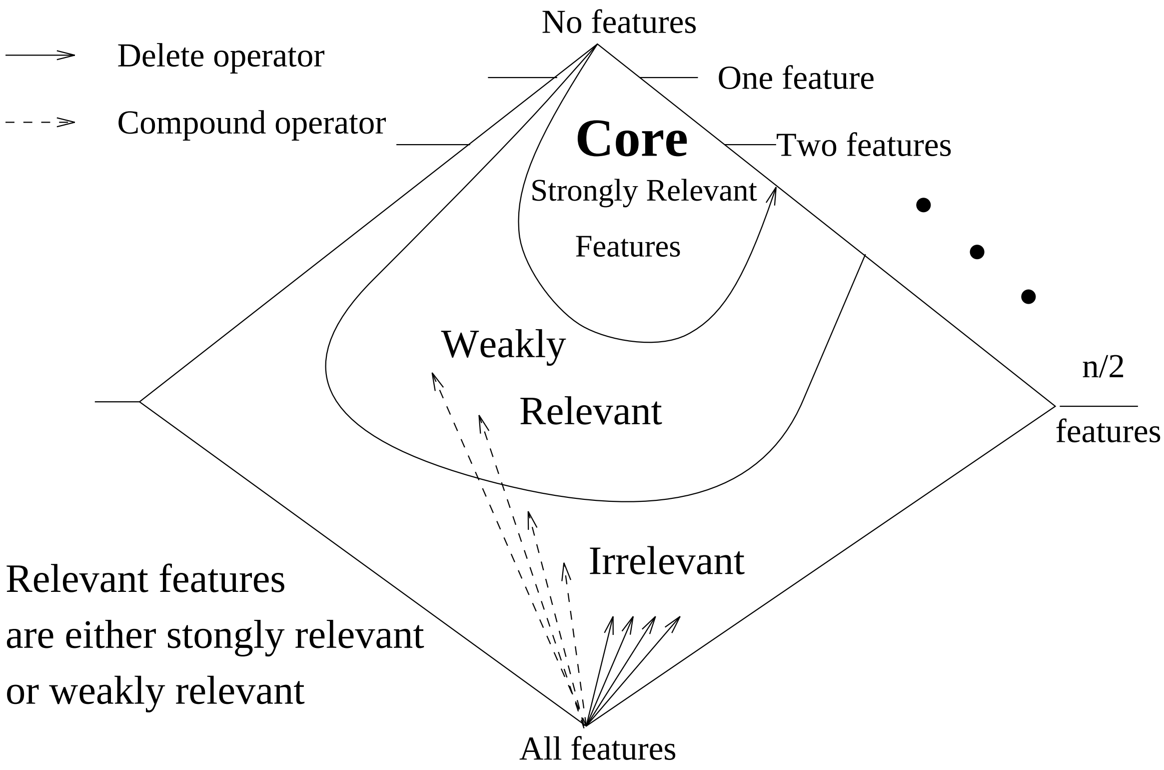

Figure 4.13: The state space. If a feature subset contains an irrelevant feature, it is in the irrelevant area; if it contains only strongly relevant features it is in the core region; otherwise, it is in the relevant region. The dotted arrows indicate compound operators.

is more information in the evaluation of the children than just the node with the maximum evaluation. Compound operators combine operators that led to the best children into a single dynamic operator. Figure 4.14 depicts a possible set of compound operators for forward selection.

The root node containing no features was expanded by applying four add operators, each one adding a single feature. The operators that led to $0, 1, 0, 0$ and $0, 0, 1, 0$ were combined into the first compound operator (shown in a dashed line going left) because they led to the two nodes with the highest evaluation (evaluation not shown). If the first compound operator led to a node with an improved estimate, the second compound operator (shown in a dashed line going right) is created that combines the best three of the original operators, *etc.*

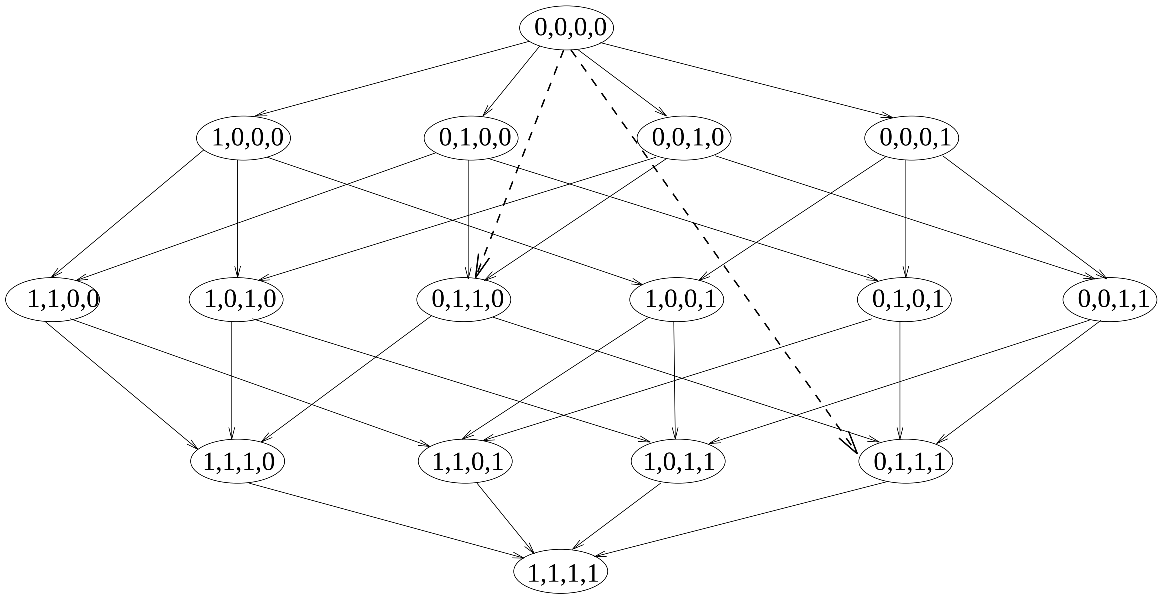

Figure 4.14: The state space search with dotted arrows indicating compound operators.

Formally, if we rank the operators by the estimated accuracy of the children, then we can define **compound operator** c_i to be the combination of the best $i+1$ operators. For example, the first compound operator will combine the best two operators. If the best two operators each added a feature, then the first compound operator will add both; if one operator added and one operator deleted, then we try to do both in one operation.

The compound operators are applied to the parent, thus creating children nodes that are farther away in the state space. Each compound node is evaluated and the generation of compound operators continues as long as the estimated accuracy of the compound nodes improves.

Compound operators generalize a few suggestions previously made in the literature. Kohavi (1994*c*) suggested that the search might start from the set of strongly relevant features (the core). If one starts from the full set of features, removal of any single strongly relevant feature will cause a degradation in performance, while removal of any irrelevant or weakly relevant feature will not. Since the last compound operator connects the full feature subset to the core, the compound operators from the full feature subset plot a path leading to the core. The path is explored by removing one feature at a time until estimated accuracy deteriorates. Caruana & Freitag (1994) implemented a SLASH version of feature

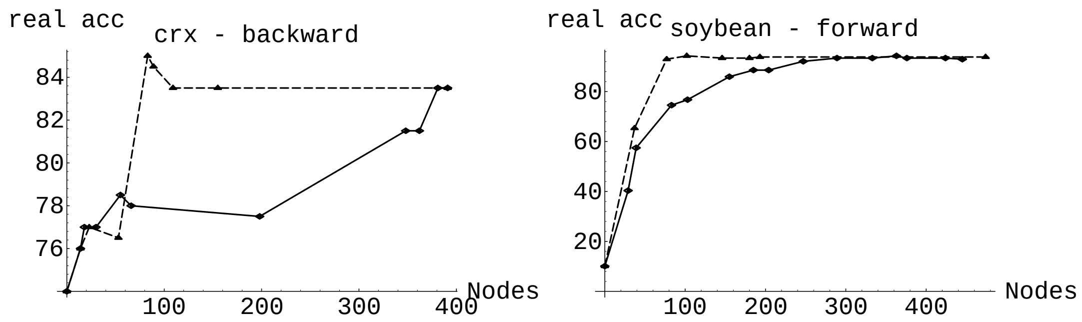

Figure 4.15: Comparison of compound (dotted line) and non-compound (solid line) searches. The accuracy (y-axis) is that of the best node on an independent test set after a given number of node evaluations (x-axis).

subset selection that eliminates the features not used in the derived decision tree. If there are no features that improve the performance when deleted, then (ignoring orderings due to ties) one of the compound operators will lead to the same node that slash would take the search to. While the SLASH approach is only applicable for backward elimination, compound operators are also applicable to forward selection.

Figure 4.15 shows two searches with and without compound operators. Compound operators improve the search by finding nodes with higher accuracy faster; however, whenever it is easy to overfit, they cause overfitting earlier (see Section 4.7). Experimental results using compound operators are similar to those without them, except that they are sometimes faster.

The main advantage of compound operators is that they make backward feature subset selection possible. Table 4.8 and Figures 4.16 and 4.17 show the results of running the best-first search algorithm with compound operators but starting with the full set of features (backward elimination). Table 4.9 shows the number of features used for each of the different methods. When one starts from the full set of features, feature interactions are easier for the search to identify. The following observations can be made:

1. Except for m-of-n-3-7-10, the accuracy results for backward FSS with ID3 generally degraded. The main improvement was for m-of-n-3-7-10, where the correct seven bits were correctly identified, resulting in 100% accuracy. The feature subsets were generally larger, and apparently even best-first search cannot overcome some local

Table 4.8: A comparison of a forward best-first search and backward best-first search with compound operators. The p-val columns indicates the probability that backward is better than forward.

	Dataset	ID3		p-val	Naive-Bayes		p-val
		BFS-FSS forward	BFS-FSS back		BFS-FSS forward	BFS-FSS back	
1	breast cancer	94.57± 0.7	93.85± 0.5	0.11	96.00± 0.6	96.00± 0.6	0.50
2	cleve	79.52± 2.3	75.89± 3.7	0.12	80.23± 3.9	82.56± 2.5	0.76
3	crx	85.22± 1.6	83.33± 1.5	0.10	86.23± 1.0	84.78± 0.8	0.05
4	DNA	94.27± 0.7	91.23± 0.8	0.00	94.60± 0.7	96.12± 0.6	0.99
5	horse-colic	82.07± 1.5	82.61± 1.7	0.63	83.42± 2.0	82.33± 1.3	0.26
6	Pima	68.73± 2.2	67.44± 1.4	0.24	75.12± 1.5	76.03± 1.6	0.72
7	sick-euthyroid	97.06± 0.5	97.06± 0.5	0.50	97.35± 0.5	97.35± 0.5	0.50
8	soybean-large	91.65± 1.0	91.35± 1.0	0.38	93.70± 0.4	94.29± 0.9	0.81
9	corral	100.00± 0.0	100.00± 0.0	0.50	90.62± 2.6	90.62± 2.6	0.50
10	m-of-n-3-7-10	77.34± 1.3	100.00± 0.0	1.00	77.34± 1.3	87.50± 1.0	1.00
11	Monk1	97.22± 0.8	97.22± 0.8	0.50	72.22± 2.2	72.22± 2.2	0.50
12	Monk2-local	95.60± 1.0	95.60± 1.0	0.50	67.13± 2.3	67.13± 2.3	0.50
13	Monk2	63.89± 2.3	64.35± 2.3	0.58	67.13± 2.3	67.13± 2.3	0.50
14	Monk3	97.22± 0.8	97.22± 0.8	0.50	97.22± 0.8	97.22± 0.8	0.50
	Average real:	86.64	85.35		88.33	88.68	
	Average artif.	88.55	92.40		78.61	80.30	

maxima. For example, DNA stopped with 36 features, but pruning more features would improve the performance because the forward search found a subset of 11 features that was significantly better (the accuracy estimation for the 11 feature subset was higher than the one for the 36 feature subset, and because the same folds are used, if the best-first search were to get to this 11-feature node, it would prefer it over the final node selected in the backward search). In the next section, we use the backward search with C4.5 that prunes, and the backward search then becomes much easier for the best-first search algorithm.

2. For Naive-Bayes, backward FSS is a slight win in terms of accuracy. Only on crx did the accuracy degrade significantly (p-val=0.05), while on m-of-n-3-7-10 and DNA it significantly improved (p-val=1.00 and 0.99 respectively). In fact, for the DNA dataset, *no other known algorithm outperformed Naive-Bayes on the selected feature subset.* Taylor *et al.* (1994, page 159) compared 23 algorithms on this dataset (with

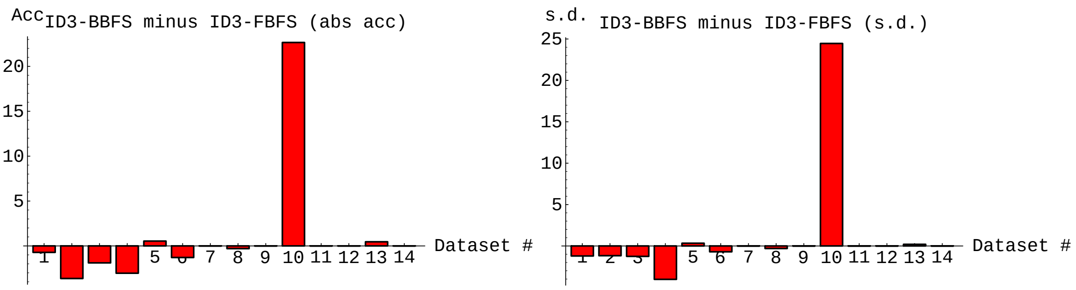

Figure 4.16: ID3: Absolute difference (best-first search FSS backward with compound operators minus forward) in accuracy (left) and in std-devs (right).

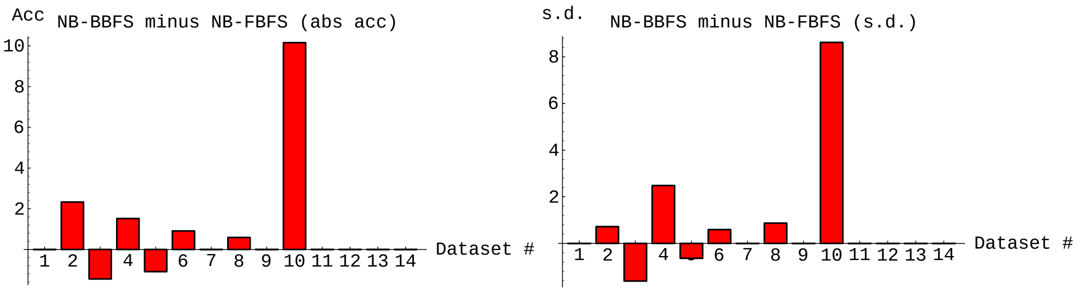

Figure 4.17: Naive-Bayes: Absolute difference in accuracy (left) and in std-devs (right).

the same split of train/test sets), and the highest ranking one was RBF (radial basis functions) using 720 centers with an accuracy of 95.9%. The Naive-Bayes with backward elimination had an accuracy of 96.12%.

3. The m-of-n-3-7-10 dataset with Naive-Bayes is a very interesting case. The feature subset selection finds six out of the seven relevant features, and the seventh selected feature is an irrelevant one. Although m-of-n can be represented using a hyperplane, and although in a Boolean domain the surface represented by Naive-Bayes is always a hyperplane, it turns out that Naive-Bayes is unable to learn this target concept.

   Table 4.10 was constructed by giving Naive-Bayes all possible instances and their correct classification for the 3-of-7 concept, and testing it on the same instances. We can see that Naive-Bayes is unable to learn 3-of-7, but what is intriguing is that fact that hiding a bit improves the accuracy.

Table 4.9: The number of features in the dataset, the number used by ID3 (since it does some feature subset selection), the number selected by hill-climbing FSS for ID3, best-first search FSS for ID3, and analogously for Naive-Bayes.

	Dataset	Number of Features					
		Original dataset	ID3	ID3-FSS		NB-FSS	
				Forward	Backward	Forward	Backward
1	breast cancer	10	9.1	3.6	5.3	5.2	5.9
2	cleve	13	11.4	3.4	4.6	3.6	7.9
3	crx	15	13.6	3.6	7.7	5.9	9.1
4	DNA	180	72	11	36	14	48
5	horse-colic	22	17.4	3.4	7.2	5.1	6.1
6	Pima	8	8.0	2.3	5.7	4.0	4.4
7	sick-euthyroid	25	14	4	4	3	3
8	soybean-large	35	25.8	13.7	17.7	13.8	16.7
9	corral	6	4	4	4	5	5
10	m-of-n-3-7-10	10	10	0	7	0	7
11	Monk1	6	6	3	3	4	4
12	Monk2-local	17	14	6	6	0	5
13	Monk2	6	6	3	3	0	0
14	Monk3	6	6	2	2	2	2

Table 4.10: The number of features given to the Naive-Bayes and the Perceptron inducers in m-of-n-3-7-10 and the resulting accuracy. The training set was the whole instance space.

Features given	Naive-Bayes accuracy	Perceptron accuracy
7 (all)	83.59	100.00
6	88.28	88.28
5	82.03	82.03

The explanation for this result is as follows. There are $\binom{7}{0} + \binom{7}{1} + \binom{7}{2} = 29$ instances out of $2^7 = 128$ that have label 0. There are $\binom{7}{1} + \binom{7}{2} \cdot 2 = 49$ ones in these 29 instances, so each of the seven feature has $49/7 = 7$ ones. We thus get the following:

$$\Pr(Y = 0 \mid X_i = 1) = 7/29$$
$$\Pr(Y = 0 \mid X_i = 0) = 22/29$$

Similarly, $\sum_{i=3}^{7} \binom{7}{i} * i = 399$, thus each of the seven features has $399/7 = 57$ ones, giving the following:

$$\Pr(Y = 1 \mid X_i = 1) = 57/99$$
$$\Pr(Y = 1 \mid X_i = 0) = 42/99$$

If there are only two ones in an instance, the probabilities computed by Naive-Bayes are:

$$\Pr(Y = 0) \propto 29/128 \cdot (7/29)^2 \cdot (22/29)^5 = 0.00331674$$
$$\Pr(Y = 1) \propto 99/128 \cdot (57/99)^2 \cdot (42/99)^5 = 0.00352351$$

giving the label one a small advantage, and making the wrong prediction. Thus there are $\binom{7}{2} = 21$ mistakes out of the 128 possible instances, which is exactly 83.59% accuracy.

With only six features, the best thing to do is to predict a label of one when there two "on" bits, which is what the Naive-Bayes does (the calculation is omitted). This will correctly capture all instances that originally had three bits, but will continue to be wrong for those instances that had only two bits. However, out of the 21 instances that had two bits on, six will now have only one bit on because there were 42 bits total, and each of the seven bits had a one six times. Thus Naive-Bayes will now make only $21 - 6 = 15$ mistakes, which yields an accuracy of 88.28%.

This example shows that although the hypothesis space for Naive-Bayes in Boolean domains is a space of hyperplanes, it is unable to correctly identify this target concept, while a perceptron can. More interesting, however, is the fact that any filter approach to features subset selection that ranks features independently (conditioned on the label) must give the same rank to each one of the seven relevant features (due to symmetry), and thus such an approach will never pick a subset of six features as the wrapper approach does. The wrapper approach indeed finds the optimal subset for this target concept.

Running times for the backward feature subset selection were about five times longer than the forward, which is not bad considering the fact that we started with the full set of features (also see the next section where compound operators help more when C4.5 is used).

4.6 Global Comparison

Godwin's Rule of Nazi Analogies: As a USENET discussion grows longer, the probability of a comparison involving Nazis or Hitler approaches one.
—Rand Lindsly's quotation file

We used ID3 and Naive-Bayes as our basic inducers for feature subset selection because they do no pruning and, therefore, the effect of feature subset selection can be seen more clearly. We have seen improvements in both algorithms, but an important remaining question is how they compare with a state-of-the-art algorithm, such as C4.5, and whether C4.5 itself can be improved with feature subset selection.

With compound operators, running C4.5 is faster than running ID3 because the compound operators can easily remove the features that were pruned by C4.5, hence it makes more sense to run the feature subset selection search backwards, which is what we have done. Figures 4.18 and 4.19 show how the number of features used changes as the search progresses, *i.e.*, as more nodes are evaluated. Notice how before each node expansion, the compound operators are applied and combine the operators leading to the best children, thus drastically decreasing the number of nodes. Without compound operators, the number of features could only decrease or increase by one at every node expansion. For example, in the DNA dataset with C4.5, "only" 3555 nodes were evaluated and a subset of 12 features was selected; without compound operators, the algorithm would have to expand $(180 - 12) \cdot 180 = 30,240$ nodes just to get to this feature subset.

Running times for backward FSS with C4.5 are still very slow, but faster than backward FSS with ID3. The largest run by far was DNA with 46 CPU hours; sick-euthyroid was the second runner up with 2.1 CPU hours; soybean-large, crx, horse-colic took about one hour; breast-cancer took 33 minutes; cleve and Pima took about 14 minutes, Monk2-local took 11 minutes; m-of-n-3-7-10 took 4 minutes; and the Monk problems and corral all took less than 2 minutes.

Table 4.11 and Figures 4.20–4.22 show a comparison with C4.5, ID3 with feature subset

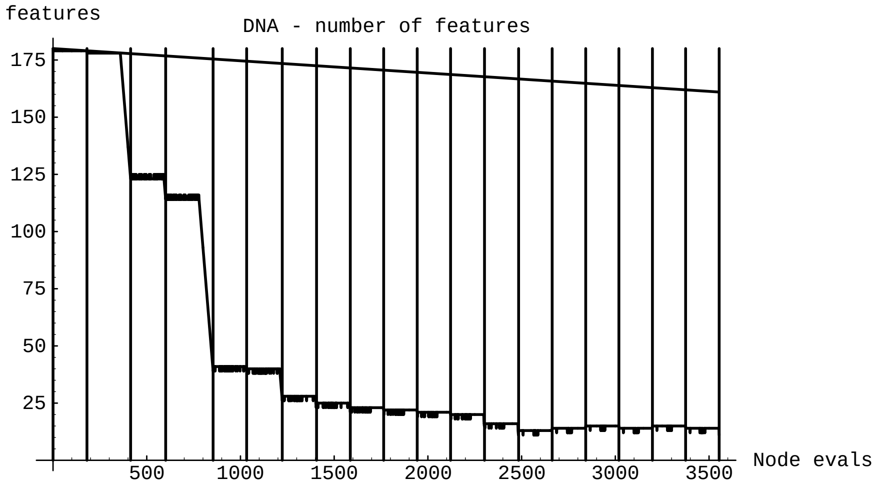

Figure 4.18: DNA: Number of features evaluated as the search progresses (C4.5, best-first search, backward). The vertical lines signify a node expansion, where the children of the best node are expanded. The slanted line on the top shows how ordinary backward selection would progress.

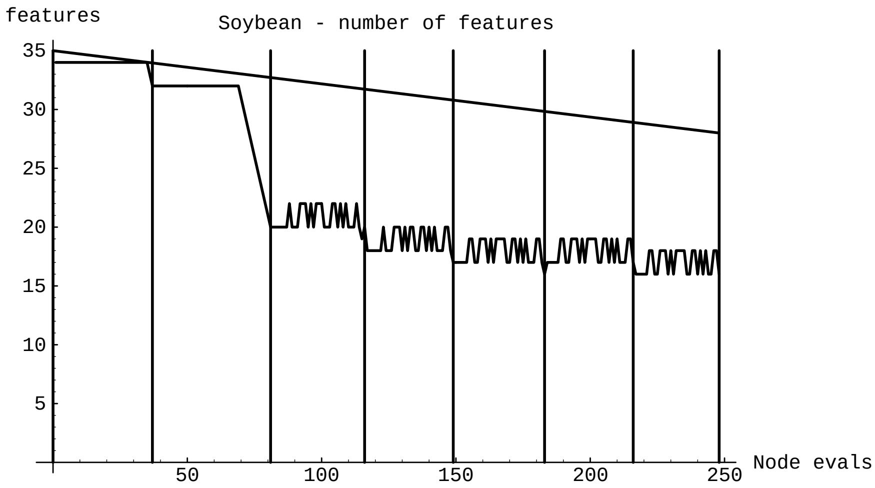

Figure 4.19: Soybean: Number of features evaluated as the search progresses (C4.5, best-first search, backward).

Table 4.11: A comparison of C4.5 with ID3-FSS, C4.5-FSS, and Naive-Bayes-FSS. The p-val columns indicates the probability that the column before it is improving upon C4.5

	Dataset	C4.5 original	ID3-FSS Frwd-BFS	p-val	C4.5-FSS Back-BFS	p-val	NB-FSS Back-BFS	p-val
1	breast cancer	95.42± 0.7	94.57± 0.7	0.11	95.28± 0.6	0.41	96.00± 0.6	0.81
2	cleve	72.30± 2.2	79.52± 2.3	1.00	77.88± 3.2	0.98	82.56± 2.5	1.00
3	crx	85.94± 1.4	85.22± 1.6	0.31	85.80± 1.3	0.46	84.78± 0.8	0.15
4	DNA	92.66± 0.8	94.27± 0.7	0.99	94.44± 0.7	0.99	96.12± 0.6	1.00
5	horse-colic	85.05± 1.2	82.07± 1.5	0.01	84.77± 1.3	0.41	82.33± 1.3	0.01
6	Pima	71.60± 1.9	68.73± 2.2	0.08	70.18± 1.3	0.20	76.03± 1.6	0.99
7	sick-euthyroid	97.73± 0.5	97.06± 0.5	0.09	97.91± 0.4	0.66	97.35± 0.5	0.21
8	soybean-large	91.35± 1.6	91.65± 1.0	0.59	91.93± 1.3	0.65	94.29± 0.9	0.99
9	corral	81.25± 3.5	100.00± 0.0	1.00	81.25± 3.5	0.50	90.62± 2.6	1.00
10	m-of-n-3-7-10	85.55± 1.1	77.34± 1.3	0.00	85.16± 1.1	0.36	87.50± 1.0	0.97
11	Monk1	75.69± 2.1	97.22± 0.8	1.00	88.89± 1.5	1.00	72.22± 2.2	0.05
12	Monk2-local	70.37± 2.2	95.60± 1.0	1.00	88.43± 1.5	1.00	67.13± 2.3	0.07
13	Monk2	65.05± 2.3	63.89± 2.3	0.31	67.13± 2.3	0.82	67.13± 2.3	0.82
14	Monk3	97.22± 0.8	97.22± 0.8	0.50	97.22± 0.8	0.50	97.22± 0.8	0.50
	Average real:	86.51	86.64		87.27		88.68	
	Average artif.	79.19	88.55		84.68		80.30	

selection (best-first search forward), C4.5 with feature subset selection (backward with compound operators), and Naive-Bayes with feature subset selection (backward with compound operators). Table 4.12 shows the number of features used by C4.5 and C4.5 with feature subset selection. The following observations can be made:

1. For real datasets, ID3-FSS and C4.5 perform approximately the same, but ID3-FSS uses fewer features. For the artificial datasets, ID3-FSS significantly outperforms C4.5 on three datasets (corral, Monk1, Monk2-local), and is significantly inferior in one (m-of-n-3-7-10).

2. C4.5-FSS significantly outperforms C4.5 on two real datasets (cleve and DNA), two artificial datasets (Monk1 and Monk2-local), and is never significantly outperformed by C4.5. The relative error is reduced by 5.6% for real datasets and by 26.4% for the artificial datasets.

3. What is perhaps most interesting is how C4.5 and Naive-Bayes with feature subset

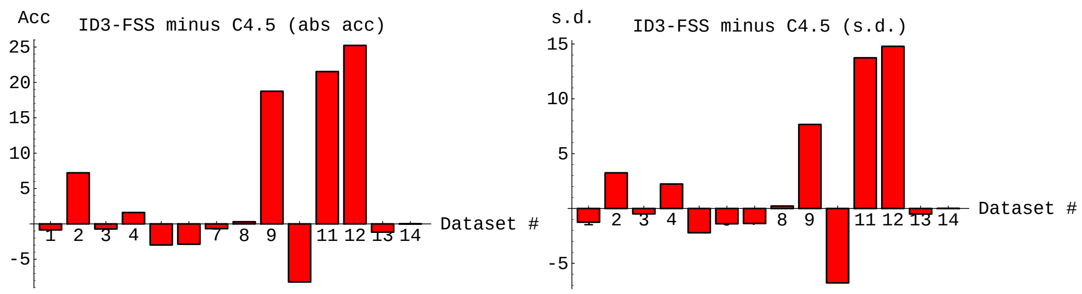

Figure 4.20: ID3: Absolute difference (FSS-ID3 minus C4.5) in accuracy (left) and in std-devs (right).

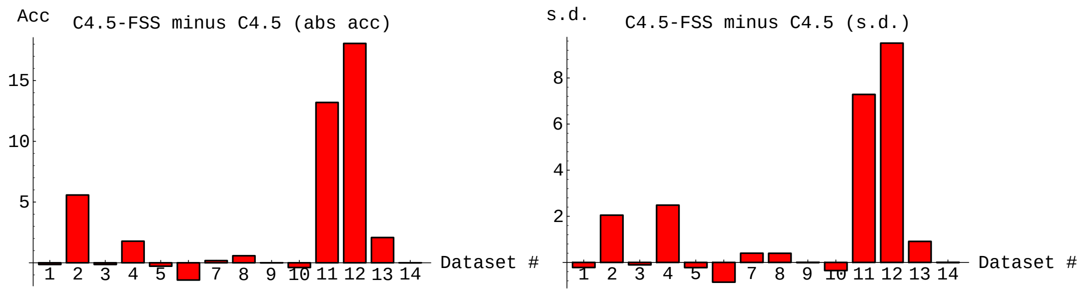

Figure 4.21: C4.5: Absolute difference (FSS-C4.5 minus C4.5) in accuracy (left) and in std-devs (right).

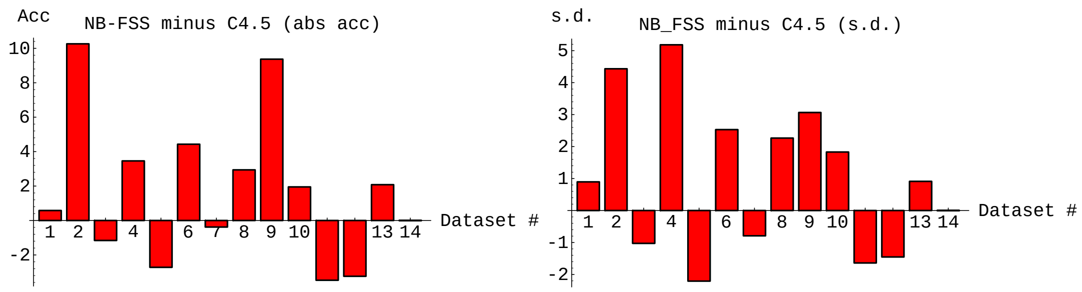

Figure 4.22: NB: Absolute difference (FSS-NB minus C4.5) in accuracy (left) and in std-devs (right).

Table 4.12: The number of features in the dataset, the number used by C4.5, and the number selected by C4.5-FSS

	Dataset	Original dataset	C4.5	C4.5-FSS Backward
1	breast cancer	10	7.0	3.9
2	cleve	13	9.1	5.3
3	crx	15	9.9	7.7
4	DNA	180	46	12
5	horse-colic	22	5.5	4.3
6	Pima	8	8.0	4.8
7	sick-euthyroid	25	4	3
8	soybean-large	35	22.0	17.1
9	corral	6	4	2
10	m-of-n-3-7-10	10	9	6
11	Monk1	6	5	3
12	Monk2-local	17	12	6
13	Monk2	6	6	0
14	Monk3	6	2	2

selection compare. While there are datasets for which either one is better than the other, on the real datasets, C4.5 is significantly better only for the horse-colic dataset, but Naive-Bayes is significantly better for cleve, DNA, Pima, and soybean-large. The relative error of Naive-Bayes is smaller by 16.1%. For the artificial datasets, the two are about equal: C4.5 is significantly better on two datasets (Monk1, Monk2-local), and Naive-Bayes is better on two (corral, m-of-n-3-7-10).

In summary, feature subset selection using the wrapper approach significantly improves ID3, C4.5 and Naive-Bayes on the datasets tested. Perhaps the most surprising result is how well Naive-Bayes performs on real datasets once discretization and feature subset selection are done.

4.7 Overfitting

Still, it is an error to argue in front of your data. You find yourself insensibly twisting them round to fit your theories.
—Sherlock Holmes / The Adventure of Wisteria Lodge.

An induction algorithm **overfits** the dataset if it models the given data too well and its predictions are poor. An example of an over-specialized hypothesis, or classifier, is a lookup table on all the features. Overfitting is closely related to the bias-variance tradeoff (Geman *et al.* 1992, Breiman *et al.* 1984): if the algorithm fits the data too well, the variance term is large, and hence the overall error is increased.

Most accuracy estimation methods, including cross-validation, evaluate the predictive power of a given hypothesis over a feature subset by setting aside instances (holdout sets) that are not shown to the induction algorithm and using them to assess the predictive ability of the induced hypothesis. A search algorithm that explores a large portion of the space and that is guided by the accuracy estimates can choose a bad feature subset: a subset with a high accuracy estimate but poor predictive power.

If the search for the feature subset is viewed as part of the induction algorithm, then overuse of the accuracy estimates may cause overfitting in the feature-subset space. Because there are so many feature subsets, it is likely that one of them leads to a hypothesis that has high predictive accuracy for the holdout sets. A good example of overfitting can be shown using a *no-information* dataset (Rand) where the features and the label are completely random. The top graph in Figure 4.23 shows the estimated accuracy versus the true accuracy for the best node the search has found after expanding k nodes. One can see that especially for the small sample of size 100, the estimate is extremely poor (26% optimistic), indicative of overfitting. The bottom graphs in the figure show overfitting in small real-world datasets.

Recently, a few machine learning researchers have reported the cross-validation estimates that were used to guide the search as a final estimate of performance, thus achieving overly optimistic results. Instead, experiments using cross-validation to guide the search must report the accuracy of the selected feature subset on a *separate* test set or on holdout sets generated by an external loop of cross-validation that were never used during the feature subset selection process.

The problem of overfitting in feature subset space has been previously raised in the machine learning community by Wolpert (1992*a*) and Schaffer (1993), and the subject has received much attention in the statistics community (cf. Miller (1990)).

Although the theoretical problem exists, our experiments indicate that overfitting is mainly a problem when the number of instances is small. Kohavi & Sommerfield (1995*a*) reported that out of 70 searches for feature subsets with datasets containing over 250 instances, ten searches were optimistically biased by more than two standard deviations and

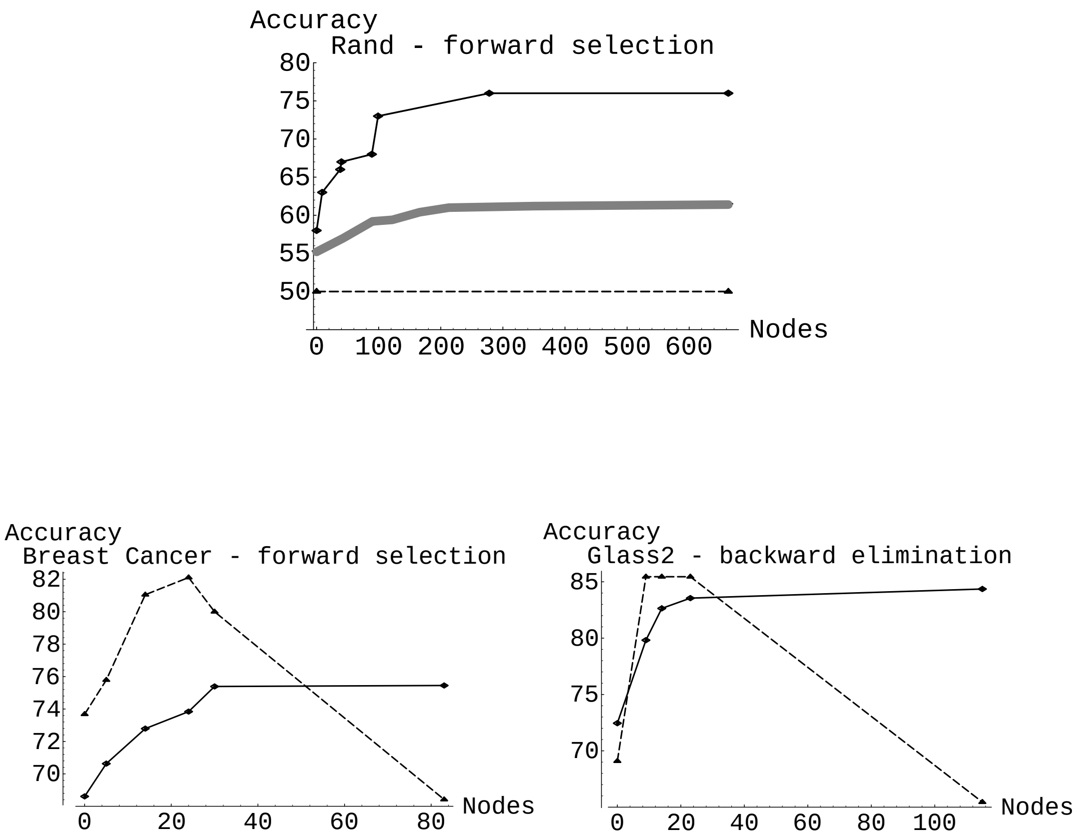

Figure 4.23: Overfitting in feature subset selection. The top graph shows the estimated and true accuracies for a random dataset and ID3. The solid line represents the estimated accuracy for a training set of 100 instances, the thick grey line for a training set of 500 instances, and the dotted line shows the real accuracy. The bottom graphs graphs show the accuracy for real-world datasets. The solid line is the estimated accuracy, and the dotted line is the accuracy on an independent test set.

one was pessimistically biased by more than two standard deviations. While the problem clearly exists, it is not very severe, especially for large datasets.

4.8 Feature Subset Selection as Search With Probabilistic Estimates

Even the best decision has a high probability of being wrong. Even the most effective one eventually becomes obsolete.
—Peter Drucker, The Effective Executive

We now look at the problem of feature subset selection as search with probabilistic estimates. This abstract view generalizes the problem, and we believe it can lead to new practical results, if the abstract problem can be solved using a different approach than the one used in previous sections.

The wrapper approach, which uses accuracy estimation as the evaluation and heuristic function, complicates the common state-space search paradigm. The fact that the accuracy estimation is a random variable implies that there is uncertainty in the returned estimate.

One way to decrease the variance is to run the accuracy estimation (*e.g.*, k-fold cross-validation) more than once and average the results, as we have done. Increasing the number of runs shrinks the confidence interval for the mean, but requires more time. The tradeoff between more accurate estimates and more extensive exploration of the search space is referred to as the exploration versus exploitation problem (Kaelbling 1993) and leads to the following abstract search problem.

Definition 4.7 (Search with Probabilistic Estimates)
Let $\mathcal{S}$ be a state space with operators between states. Let $f : \mathcal{S} \mapsto \mathbb{R}$ be an unbiased probabilistic evaluation function that maps a state to a real number, indicating how good the state is. The number returned by $f(s)$ comes from a distribution $D(s)$ with mean $f^*(s)$, which is the actual (unknown) value of the state. The goal is to find the state s with the maximal value of $f^*(s)$.

The mapping of this definition to the feature subset selection problem is as follows. The states are the subsets, and the operators are the common ones (add, delete, compound). The evaluation function is the accuracy estimation accuracy. Although some accuracy estimation techniques, such as cross-validation, are biased, they can be viewed as unbiased estimators for a different quantity; for example, k-fold cross-validation is unbiased for datasets of size $m - m/k$ (see Section 3.4.1). Furthermore, for model selection, this pessimism is of minor importance because the bias may cancel out. We now describe work that falls under this general framework of search with probabilistic estimators:

Greiner (1992) described how to conduct a hill-climbing search when the evaluation function is probabilistic. The algorithm stops at a node that is a local optimum with high probability, based on the Chernoff bound. Yan & Mukai (1992) analyzed an algorithm based on simulated annealing and showed that it will find the global optimum if given enough time.

Maron & Moore (1994), in an approach very similar to Greiner's, attempted to shrink the confidence interval of the accuracy for a given set of models, until one model can be proven to be optimal with high probability. The evaluation function is a single step in leave-one-out cross-validation, *i.e.*, the algorithm is trained on randomly chosen $n - 1$ instances and tested on the one that is left. The induction algorithm used is instance-based learning, which leads to an extremely fast evaluation because training is not necessary. A step of leave-one-out is merely a test of whether an instance is classified correctly by its nearest-neighbor. Note, however, that $f(s)$ always returns either a zero or a one. The instance is either correctly classified, or not. This step must be repeated many times to get a reasonable confidence bound.

The general idea is to *race* competing models, until one is a clear winner. Models drop out of the race when the confidence interval of the accuracy does not overlap with the confidence interval of the accuracy of the best model (this is analogous to imposing a higher and lower bound on the estimation function in the B^* algorithm (Berliner 1981)). The race ends when there is a winner, or when all n steps in the leave-one-out cross-validation have been executed. The confidence interval is defined according to Hoeffding's formula (Hoeffding 1963):

$$\Pr\left(\left|f^*(s) - \widehat{f}(s)\right| > \epsilon\right) < 2e^{-2m\epsilon^2/B^2}$$

where $\widehat{f}(s)$ is the average of m evaluations and B bounds the possible spread of point values. Given a confidence level, one can determine ϵ, and hence a confidence interval for $f^*(s)$, from the above formula. The paper (Maron & Moore 1994), however, does not discuss any search heuristic, and assumes that a fixed set of models is given by some external source.

Moore & Lee (1994) describe an algorithm for feature subset selection that has both ingredients of the abstract problem: it has a search heuristic, and it uses the probabilistic estimates in a non-trivial manner.

The algorithm does a forward selection and backward elimination, but instead of estimating the accuracy of each added (deleted) feature using leave-one-out cross-validation, all the features that can be added (deleted) are raced in parallel. Assuming that the distribution of $f(s)$ is normal, confidence intervals are used to eliminate some features from the race.

Schemata search is another search variant that allows taking into account interactions between features. Instead of starting with the empty or full set of features, the search begins with the unknown set of features. Each time a feature is chosen and raced between being "in" or "out." All combinations of unknown features are used in equal probability, thus a feature that should be "in" will win the race, even if correlated with another feature. Although this method uses the probabilistic estimates in a Bayesian setting, the basic search strategy is simple hill-climbing.

4.9 Automatic Parameter Tuning for C4.5

Too many machine learning programs suffer from an extreme case of this deficiency [requirement for parameter tuning], which is named the "China Syndrome" because sometimes the only person who is able to make a program run is in China.
*—Buchanan, 1987 talk [paraphrased in Catlett (1991*a*)]*

We now describe an application of the wrapper approach to parameter tuning for the C4.5 induction algorithm (Quinlan 1993). The C4.5-AP algorithm is a wrapper around C4.5 that attempts to automatically tune some parameters in C4.5 that can be adjusted.

We chose to automatically set all of the C4.5 tree-building parameters (m,c,g, and s) shown in Table 4.13. The g and s parameters are Boolean and the m and c parameters are continuous, so we discretized them as shown in Table 4.14. The decision to discretize the continuous values in a non-linear way was based on our observations that fine granularity was not important for large values of the parameters. There are are a total of 1156 possible states, so a search engine must be used to avoid trying all of them. In our experiments, we used best-first search. The operators are defined such that each node, defining a vector of values for these parameters, is connected to all nodes that vary on one parameter, which

Table 4.13: Parameters to the C4.5 algorithm.

Name	Possible Values	Default Value	Description
mm	$1 \ldots \infty$	2	Stopping parameter in tree construction. Halts the recursive partitioning process when no partition of the current node results in children all having weight $\geq m$. (*Weight* is equal to the number of instances unless there are missing values.)
cc	$[0, 100]$	25	Confidence level parameter in tree pruning. Small values of c cause heavy pruning, large values cause little pruning.
g	on,off	off	Splitting criterion: information gain or gain ratio. When g is specified, information gain is used as the splitting criterion in tree construction. When not specified, gain ratio is used.
s	on,off	off	Subset splits. When s is specified, subset splits are considered during tree construction. When unspecified, all splits on k-valued nominal features result in k children.

is either adjacent to the current value or five entries away in the array of possible values. More formally, for the binary parameters, there is an operator to the opposite value; for each of the numeric parameters, there is an operator increasing or decreasing the value by going one position or five positions up or down in the array of possible values (Table 4.14). The initial state is the default setting for C4.5: ($m = 2, c = 25$, no g, no s).

Table 4.15 and Figure 4.24 show C4.5 and C4.5 with automatic parameter tuning using the wrapper approach. The following observations can be made:

1. For the real datasets, there is little improvement on average: 5.3% reduction in the error rate. Most differences are insignificant, except for the cleve dataset, which is significant with a p-value of 0.99.

2. For all the artificial datasets, the accuracy either remained the same (Monk3) or improved (the rest). Four datasets (corral, mofn, Monk1, Monk2-local) significantly improved with a p-value of 1.00. The dataset Monk2, improved slightly by increasing the pruning factor so much (c was set to 4), that the tree was pruned down to a single node, predicting the majority class.

Table 4.14: The state space searched by C4.5-AP.

Parameter	Considered Values
m	1,2,3,4,5,10,15,20,25,30,40,50,60,70,80,90,100
c	1,2,3,4,5,10,15,20,25,30,40,50,60,70,80,90,100
s	on,off
g	on,off

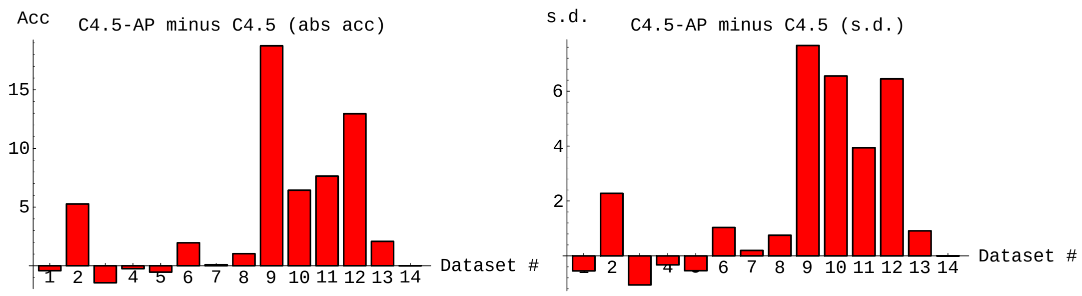

Figure 4.24: C4.5: Difference of accuracy between automatic parameter tuning and original C4.5. Absolute difference on the left, and in std-devs on the right.

Given these results, it is hard to judge whether C4.5 with automatic parameter tuning (C4.5-AP) is significantly superior to C4.5. In Kohavi & John (1995), an extensive comparison was made using 33 datasets, and we repeat the main results in Table 4.16. (A slightly different search scheme was used, but the differences are minor and not important.) Figure 4.25 depicts the accuracies on a scatter plot, where C4.5-AP's accuracy is on the x axis, and C4.5's is on the y-axis. Points below the 45 degree line indicate that C4.5-AP is outperforming C4.5.

In this large scale experiment, there were nine significant differences in favor of C4.5-AP (at the 90% significance level), and only one significant result in favor of C4.5. At a 95% confidence level, C4.5-AP outperformed C4.5 on six datasets, and is never outperformed by C4.5. In the one case where C4.5-AP loses at a 90% confidence level, the labor-negotiation dataset, note that the entire dataset is very small with only 57 instances. The parameters chosen varied widely, with the $-m1$ parameter chosen in many cases. Changing C4.5's default parameter to $-m1$ did not give better average performance over all datasets. While the results from the large scale experiment were very encouraging, the average reduction in

Table 4.15: C4.5 versus C4.5 with automatic parameter tuning.

	Dataset	C4.5	C4.5-AP	p-val
1	breast cancer	95.42± 0.7	95.00± 0.8	0.29
2	cleve	72.30± 2.2	77.57± 2.5	0.99
3	crx	85.94± 1.4	84.49± 1.4	0.14
4	DNA	92.66± 0.8	92.41± 0.8	0.37
5	horse-colic	85.05± 1.2	84.51± 0.8	0.29
6	Pima	71.60± 1.9	73.56± 1.9	0.85
7	sick-euthyroid	97.73± 0.5	97.82± 0.5	0.58
8	soybean-large	91.35± 1.6	92.38± 1.1	0.77
9	corral	81.25± 3.5	100.00± 0.0	1.00
10	m-of-n-3-7-10	85.55± 1.1	91.99± 0.9	1.00
11	Monk1	75.69± 2.1	83.33± 1.8	1.00
12	Monk2-local	70.37± 2.2	83.33± 1.8	1.00
13	Monk2	65.05± 2.3	67.13± 2.3	0.82
14	Monk3	97.22± 0.8	97.22± 0.8	0.50
	Average real:	86.51	87.22	
	Average artif.	79.19	87.17	

relative error, which was 13%, was mostly a result of improvements on artificial datasets. The average improvement for the real datasets was 6.4%, a similar result to the one we achieved for the common datasets in this dissertation.

Table 4.16 has an extra column, C4.5*, which is the accuracy achieved when the heuristic evaluation function was the accuracy on the actual test set. The C4.5* results cannot realistically be achieved in practice, but they show an upper bound to our approach: even if we were to use the perfect error estimation method instead of cross-validation, we could never surpass the C4.5* results.

4.10 Related Work

Opalko's Observation: The probability of one's supervisor entering one's office unannounced is inversely proportional to the work-relatedness of the activity one is engaged in at the time.
—Unix fortune

We now review related work in three areas: feature subset selection, parameter tuning, and the wrapper approach.

Table 4.16: Accuracies for the C4.5, C4.5-AP, and C4.5* upper bound, which uses the test set as the evaluation function (instead of cross-validation). $P(t)$ gives the p-value of a paired t-test for the folds, and the "Better" column indicates significantly better at the 90% level.

Dataset	Size	Accuracy Default C4.5	Accuracy C4.5-AP	Accuracy C4.5*	$P(t)$	C4.5-AP Better
australian	690	85.36±1.13	85.07±1.47	88.70±1.58	.372	
breast (W)	699	95.42±0.70	96.29±0.77	96.98±0.74	.961	✓
breast (L)	286	73.87±2.76	73.87±2.76	77.68±1.90	.500	
chess	3196	99.50±0.13	99.62±0.10	99.75±0.08	.800	
cleve	303	72.30±2.17	75.61±2.43	84.82±1.42	.931	✓
corral	32/129	81.20±3.44	100.00±0.00	100.00±0.00	1.00	✓
crx	690	85.94±1.37	85.07±1.26	88.84±0.87	.186	
diabetes	768	71.75±1.02	75.26±1.26	82.03±1.42	.984	✓
DNA	2000/1186	92.30±0.77	93.20±0.73	93.2±0.73	.800	
german	1000	72.50±1.41	72.70±1.65	79.42±1.45	.553	
glass	214	65.48±3.22	68.20±2.92	76.17±2.69	.847	
glass2	163	70.55±2.00	74.85±3.94	87.61±2.96	.789	
heart	270	80.00±2.77	82.22±2.19	86.11±1.90	.703	
hepatitis	155	80.04±3.65	84.50±2.47	89.67±2.98	.924	✓
horse-colic	368	85.05±1.16	84.24±0.78	88.86±0.85	.139	
hypothyroid	3163	99.11±0.18	99.27±0.08	99.40±0.21	.894	
iris	150	95.33±1.42	95.33±1.42	95.33±1.42	.500	
labor-neg	57	85.67±3.48	80.67±3.87	89.00±3.02	.097	×
letter	15000/5000	86.80±0.48	87.00±0.48	87.10±0.48	.620	
lymphography	148	78.38±1.65	74.14±3.19	86.48±1.41	.103	
Monk1	124/432	75.70±2.06	100.0±0.00	100.0±0.00	1.00	✓
Monk2	169/432	65.00±2.29	62.50±2.33	82.40±1.83	.220	
Monk3	122/432	97.20±0.79	97.20±0.79	100.00±0.00	.500	
mushroom	8124	100.00±0.00	100.00±0.00	100.00±0.00	.500	
Pima	768	71.60±1.93	76.67±2.05	79.55±1.55	.999	✓
satimage	4435/2000	85.20±0.79	86.10±0.77	85.20±0.79	.790	
segment	2310	96.36±0.33	96.80±0.37	97.75±0.36	.846	
sick-euthyroid	3163	97.69±0.25	97.63±0.46	98.20±0.19	.444	
soybean	47	100.00±0.00	100.00±0.00	100.00±0.00	.500	
tic-tac-toe	958	85.59±1.08	93.73±0.52	96.34±0.65	1.00	✓
vehicle	846	69.84±1.77	72.44±1.73	78.18±0.94	.973	✓
vote	435	95.64±0.52	95.41±0.47	97.71±0.68	.172	
vote1	435	88.02±1.77	87.58±1.52	92.40±1.20	.342	

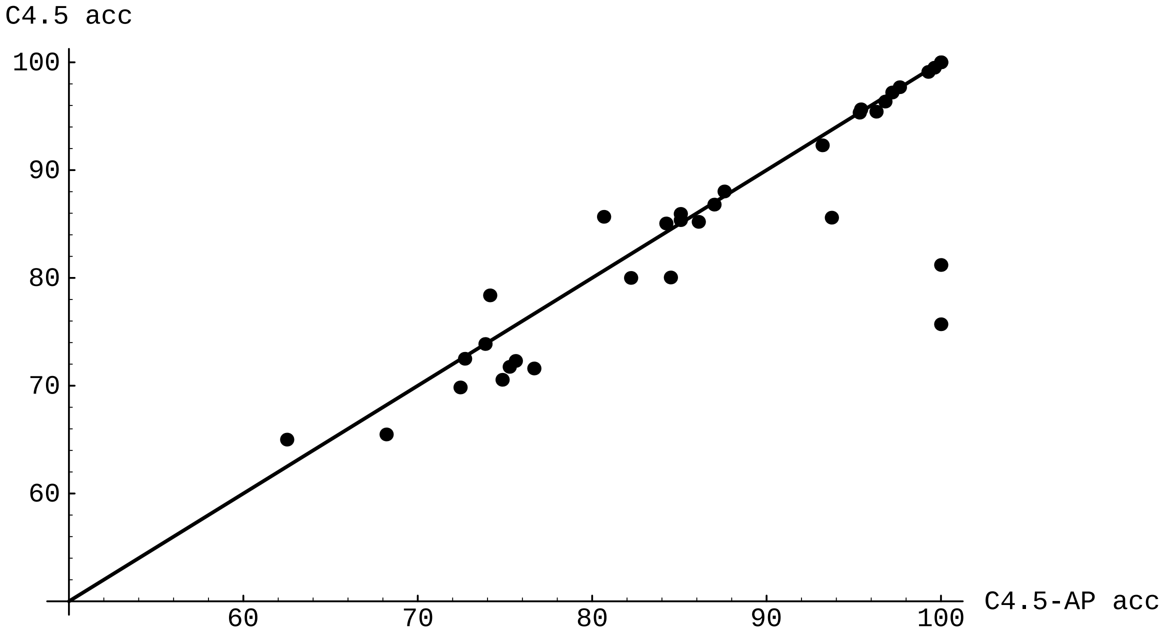

Figure 4.25: Accuracies of the 33 datasets. C4.5-AP on x-axis and original C4.5 on y axis. Points below the 45 degree line indicate that C4.5-AP is outperforming C4.5.

4.10.1 Feature Subset Selection

The pattern recognition literature (Devijver & Kittler 1982, Kittler 1986, Ben-Bassat 1982), statistics literature (Draper & Smith 1981, Miller 1984, Miller 1990, Neter *et al.* 1990), and recent machine learning papers (Almuallim & Dietterich 1991, Almuallim & Dietterich 1994, Kira & Rendell 1992*a*, Kira & Rendell 1992*b*, Kononenko 1994) consist of many such measures for feature subset selection that are all based on the data alone.

Most measures in the pattern recognition and statistics literature are monotonic, *i.e.*, for a sequence of nested feature subsets $F_1 \supseteq F_2 \supseteq \cdots \supseteq F_k$, the measure f obeys $f(F_1) \geq f(F_2) \geq \cdots \geq f(F_k)$. Notable selection measures that do satisfy monotonicity assumption are residual sum of squares (RSS), adjusted R-square, minimum mean residual, Mallow's C_p (Mallows 1973), discriminant functions, and distance measures, such as the Bhattacharyya distance and divergence. The PRESS measure (Prediction sum of squares), however, does not obey monotonicity. For monotonic functions, branch and bound techniques can be used to prune the search space. Furnival & Wilson (1974) show how to compute the residual sum of squares (RSS) for all possible regressions of k features in less than six (!) floating-point operations per regression; furthermore, the technique can be combined with branch and bound algorithms as described in their paper.[3] Narendra & Fukunaga (1977) apparently rediscovered the branch-and-bound technique, which was later improved in Yu & Yuan (1993). Most machine learning induction algorithms do not obey monotonic restrictions, and so this type of dynamic programming cannot be used. Even when branch and bound can be used, the search is usually exponential, and when there are more than 30 features, suboptimal methods are used.

Searching in the space of feature subsets has been studied for many years. Sequential backward elimination, sometimes called sequential backward selection, was introduced by Marill & Green (1963). Kittler (1978) generalized the different variants including forward methods, stepwise methods, and "plus ℓ–take away r." Cover & Campenhout (1977) showed that even for multivariate normally distributed features, no hill-climbing procedure that uses a monotonic measure and that selects one feature at a time can find the best feature subset of a desired size; even a 2-1 algorithm that adds the best pair and removes the worst single feature can fail. More recent papers attempt to use AI techniques, such as beam search and bidirectional search (Siedlecki & Sklansky 1988), best-first search (Xu, Yan & Chang 1989), and genetic algorithms (Vafai & De Jong 1992, Vafai & De Jong 1993). All the algorithms described above assume that the evaluation function is deterministic. Langley (1994) reviewed feature subset selection methods in machine learning and contrasted the wrapper and filter approaches.

The theory of rough sets defines notions of relevance that are closely related to the ones defined here (Pawlak 1991). Pawlak (1993) wrote that one of the most important and fundamental notions to the rough sets philosophy is the need to discover redundancy and dependencies between features, and there has been a lot of work on feature subset selection coming from the rough sets community (cf. Modrzejewski (1993) and Ziarko (1991)). While the goal of finding a good feature subset is the same, Kohavi & Frasca (1994) have claimed that relevance does not necessarily imply usefulness for induction tasks (see also Section 4.2.3).

[3]The Forest Service must have been really interested in this problem. Furnival was at the School of Forestry at Yale University, and Wilson was from the USDA Forest Service! One would think that they should have been working on tree pruning and not on linear regression.

Turney (1993) defines a feature to be **primary** if there is one feature value such that the probability of a class changes when conditioned on this value. He then defines a **contextual feature** in the same way we have defined strongly relevant, except that it cannot be primary. A feature is contextual only if it helps in the context of all others. Contextual features are harder to find because they involve interactions.

David Heckerman and Dan Geiger (personal communication) pointed out that a weaker notion of relevance can be defined than our notion of weak relevance. We required that the probability change for a given value, but this could be generalized to a set of values, *i.e.*, the probability that some feature takes on a set of values changes.

Our principal motivation for feature subset selection was increased accuracy or increased comprehensibility resulting from the smaller subset of features used. Some of the work on feature subset selection was motivated by reducing the measurement costs associated with the features: if fewer features are used, less measurements have to be taken. Because the wrapper approach optimizes any desired function, such costs can easily be taken into account in the evaluation function.

4.10.2 Parameter Tuning

Parameter tuning is a widely-studied problem in statistics. CART (Breiman *et al.* 1984) is a prime example of the automatic setting of a parameter (the cost-complexity parameter) in decision tree induction. CART uses stratified ten-fold cross-validation to set this parameter. There are still many parameters left to be set by the user, however, and it would be interesting to compare a fully automated CART to the standard CART. Nearly all statistical methods of regression contain a single *smoothing parameter* λ (similar to the cost-complexity parameter in CART, and the `m` and `c` parameters in C4.5), which attempts to address the bias-variance dilemma: how to trade off fit to the training data with some measure of "complexity" of the model. Quinlan (1993) discusses the importance of the C4.5 parameters, and suggests that the user manually perform a search through the space of parameter values using cross-validation to evaluate each parameter. John (1994) reports preliminary results on using cross-validation and exhaustive search to set the `m` parameter in C4.5.

4.10.3 The Wrapper Approach

Since the introduction of the wrapper approach, we have seen it used in a few papers that reference our original paper (John *et al.* 1994). Langley & Sage (1994*a*) used the wrapper approach to select features for Naive-Bayes (but without discretization) and Langley & Sage (1994*b*) used it to select features for a nearest-neighbor algorithm. Pazzani (1995) used the wrapper approach to select features and join features (create super-features that compound others) for Naive-Bayes and showed that it indeed finds correct combinations when features interact. Singh & Provan (1995) and Provan & Singh (1995) used the wrapper approach to select features for Bayesian networks and showed significant improvements over the original K2 algorithm. Street, Mangasarian & Wolberg (1995) use the wrapper in the context of a linear programming generalizer. All the algorithms mentioned above use a hill-climbing search engine.

Aha & Bankert (1994) used the wrapper for identifying feature subsets in a cloud classification problem with 204 features and 1633 instances; they concluded that their empirical results strongly support the claim that the wrapper strategy is superior to filter methods. Aha & Bankert (1995) compare forward and backward feature subset selection using the wrapper approach and a beam-search engine and conclude that forward selection is better. Mladenić (1995) independently extended the use of wrappers from feature subset selection to parameter tuning. Doak (1992) has developed a method similar to the wrapper approach independently, and compared many search engines for feature subset selection; however, he was not aware of the fact that one should use an independent test set for the final estimation and used the accuracy estimation used to guide the search (see Section 4.7).[4]

4.11 Future Work

Similarly, in every course and on every project there is someone who just cannot believe that C++ features can be affordable and therefore sticks to the familiar and trusted C subset for future work.
—Stroustroup (1994, p. 173)

Many variations and extensions of the current work are possible. We have examined hill-climbing and best-first search engines. Other approaches could be examined, such as

[4]The results in both papers by Aha and Bankert, those of Mladenić, and those of Doak must be interpreted cautiously because they were using the cross-validation accuracy used during the search as the final estimated performance as opposed to an independent test set or an external loop of cross-validation as we have done.

simulated annealing approaches that evaluate the better nodes more times (van Laarhoven & Aarts 1987). Looking at the search, we have seen that one general area of the search space is explored heavily when it is found to be good. It might be worthwhile to introduce some diversity into the search, following the genetic algorithm and genetic programming approaches (Holland 1992, Goldberg 1989, Koza 1992). The problem has been abstracted as search with probabilistic estimates (Section 4.8), but we have not done experiments in an attempt to understand the tradeoff between the quality of the estimates and the search size.

The search for a good subset is conducted in a very large space. We have started the search from the empty set of features and from the full set of features, but one can start from some other initial node. One possibility is to estimate which features are strongly relevant, and start the search from this subset, although compound operators seem to be a partial answer to this problem. Another possibility is to start at random points and conduct a series of hill-climbing searches.

The wrapper approach is very slow. For larger datasets, it is possible to use cheaper accuracy estimation methods, such as holdout, or decrease the number of folds. Furthermore, some inducers allow updating their internal structure, leading to the possibility of doing incremental cross-validation as suggested in Kohavi (1995*a*), thus drastically reducing the running time. Although C4.5 does not support incremental operations, Utgoff (1994) has shown that this is possible, and has implemented a fast version of leave-one-out as suggested in Kohavi (1995*a*).

Recently, aggregation techniques, sometimes called stacking, have been advocated by many people in machine learning, neural networks, and Statistics (Wolpert 1992*b*, Breiman 1994*a*, Freund & Schapire 1995, Schapire 1990, Freund 1990, Perrone 1993, Krogh & Vedelsby 1995, Buntine 1992, Kwok & Carter 1990). It is possible to build many models, each one with a different parameter setting or with a different feature subset, and let them vote on the class. Aggregation techniques reduce the variance of the models by aggregating them, but they make it extremely hard to interpret the resulting classifier.

4.12 Summary

We have described the wrapper approach to parameter tuning and selected two problems by which to study this approach: feature subset selection and tuning of C4.5's parameters.

We have investigated the relevance and irrelevance of features, and concluded that weak and strong relevance are needed to capture our intuition better. We have then shown that these definitions are mainly useful with respect to an optimal rule, *i.e.*, Bayes rule, and that in practice one should look for optimal features with respect to the specific learning algorithm at hand. Relevance does help motivate compound operators, which are currently the only practical way to conduct backward searches for feature subsets.

The wrapper approach requires a search space, operators, a search engine, and an evaluation function. For the evaluation function, we used cross-validation as our accuracy estimation technique, based on the results in Chapter 3. We have used the common search space with add and delete operators as the basis for comparing two search engines: hill-climbing and best-first search. We have then defined compound operators that use more information in the children of an expanded node, not just the maximum value. These compound operators make a backward search, starting from the full set of features, practical. Best-first search with compound operators seems to be a strong performer and improves ID3, C4.5, and Naive-Bayes, both in accuracy, and in comprehensibility, as measured by the number of features used.

We have also shown some problems with this approach, namely overfitting and the large amounts of CPU times required, and we defined the search problem as an abstract state space search with probabilistic estimates, a formulation that may capture other general problems and that might be studied independently to solve the existing problems. The time issue seems to be the most important, although with larger amounts of data, cross-validation can be replaced with holdout accuracy estimation for an immediate improvement in time by a factor of five.

We have shown how the wrapper approach can be used to set C4.5's parameters, again with significant results, and concluded with a discussion of related work and future work.

We find it interesting that all the Statistics literature on feature subset selection that we are aware of either does an exhaustive search (or a dynamic programming variant) or a hill-climbing search, but not best-first search. Because the best-first search engine is applicable to standard statistical measures, and because we have found it to be superior to hill-climbing, we believe it should be used more in the Statistics community.

Chapter 5

Decision Tables

Write the vision, and make it plain upon tables, that he may run that readeth it.
—Habakkuk 2:2

In this chapter, we evaluate the power of feature subset selection as the only inductive process in an inducer. After continuous features are discretized (see Section 2.3), a set of features is selected by the induction algorithm, and a decision table is built by projecting the training set on the selected set of features. Given an unlabelled test instance, the classifier, called DTM (Decision Table Majority), predicts the majority class of the training-set instances whose values on the selected set of features match the test instance; if no instances are found, the classifier "gives up" and predicts the majority of the whole training set.

One advantage of decision tables is that they can easily be updated. Instances can be inserted and deleted quickly, thus allowing *incremental cross-validation* to be performed. The time required to cross-validate the DTM induction algorithm and a dataset is linear in the number of instances, the number of features, and the number of label values. The time for incremental cross-validation is independent of the number of folds chosen, hence leave-one-out cross-validation and ten-fold cross-validation take the same time.

We have designed DTM as a simple classifier on purpose. The motivation is to see how much of inductive power comes only from selecting a good set of features.

5.1 Introduction

Everything should be as simple as possible, but no simpler.
—Albert Einstein (1879-1955)

The results on feature subset selection in Chapter 4 showed that a small subset of features suffices to make accurate predictions in many real-world problems. For example, Table 4.12 on page 114 showed that when C4.5-FSS-backward was used as the induction algorithm, only 3 features out of 25 were needed to make accurate predictions in the sick-euthyroid domain, only 12 features out of 180 were needed to make accurate predictions in the DNA domain, and only about 3.9 features out of 10 were needed to make predictions in the breast-cancer domain.

In this chapter, we investigate the power of one of the simplest hypothesis spaces possible: a decision table with a default rule mapping to the majority class. This representation, called **DTM** (Decision Table Majority), has two components: a schema, which is a set of features that are included in the table, and a body consisting of labelled instances from the space defined by the features in the schema. To build a DTM, the induction algorithm must decide which features to include in the schema and which instances to store in the body. In this chapter, we assume that the full training set is stored in the decision table; the only inductive process is thus selecting the feature subset for the schema. We select the features using the wrapper approach as defined in Chapter 4.

The goal of this chapter is to evaluate the representation power of DTMs, which will then indicate how much of the generalization power comes from selecting a globally good set of features. While we use a specific technique for selecting the features, the wrapper approach, our aim is not to show that the specific method for selecting features is good, but rather to show that at least one method for selecting the schema works well. It is conceivable that other methods, perhaps better and faster, exist.

This chapter is organized as follows. In Section 5.2, we formally define DTMs and the problem of finding an optimal feature subset. In Section 5.3, we briefly describe the induction algorithm, IDTM, which is basically a null algorithm wrapped around with a feature subset selector, and then describe incremental cross-validation, which is a way to speed up the accuracy estimation step of the wrapper approach. In Section 5.4, we describe the results from experiments using IDTM. In Section 5.5, we describe related work on decision tables. We conclude with a summary in Section 5.6.

5.2 Definitions

It's useless to try to plan for the unexpected... by definition!
—Alfred Hitchcock

Given a training sample containing labelled instances, an induction algorithm builds a hypothesis in some representation. The representation we investigate here is a decision table with a default rule mapping to the majority class, which we abbreviate as **DTM**. A DTM has two components:

1. A **schema**, which is a set of features.

2. A **body**, which is a multiset of labelled instances. Each instance consists of a value for each of the features in the schema and a value for the label.

Given an unlabelled instance, $\vec{x}$, the label assigned to the instance by a DTM classifier is computed as follows. Let $\mathcal{I}$ be the set of labelled instances in the DTM that exactly match the given instance $\vec{x}$, where only the features in the schema are required to match and all other features are ignored. If $\mathcal{I} = \emptyset$, return the majority class in the DTM; otherwise, return the majority class in $\mathcal{I}$, breaking ties arbitrarily. Unknown/missing values are treated as distinct values in the matching process.

The goal of the induction algorithm is to find an optimal feature subset for the decision table. Formally, Let $\vec{X} = \{X_1, \ldots, X_n\}$ be a set of features and let $\mathcal{D}$ be the training set. Given a subset of features $\vec{X'} \subseteq \vec{X}$, denote by $\boldsymbol{DTM(\vec{X'}, \mathcal{D})}$ the DTM with schema $\vec{X'}$ and a body consisting of all instances in $\mathcal{D}$ projected on $\vec{X'}$. The goal of the induction algorithm is to chose a schema $\vec{X}_{\text{opt}}$ such that

$$\vec{X}_{\text{opt}} = \mathop{\arg\max}_{\vec{X'} \subseteq \vec{X}} \ \mathtt{acc}(DTM(\vec{X'}, \mathcal{D})), \tag{5.1}$$

where $\mathtt{acc}$ denotes the probability of the DTM correctly classifying a randomly selected instance from the space $\mathcal{X} \times \mathcal{Y}$. Note that the schema $\vec{X}_{\text{opt}}$ consists of an optimal feature subset for a DTM under the assumption that all instances from the training set are stored in the body of the decision table.

5.3 The IDTM Algorithm and Incremental Cross-Validation

`Incremental implementation`: Delivering several partial products each for the price of a complete one
—Glossary of Software Engineering Terms

The goal of **IDTM** algorithm (Inducer of DTMs) is to find the feature subset $\vec{X}_{\text{opt}}$ that is described in Equation 5.1. For this goal, we use the wrapper approach (Chapter 4) that wraps around a *null induction algorithm.* This null induction algorithm creates a DTM from the full dataset it receives; it does no induction of any sort, which is exactly what we want. The wrapper selects the appropriate features to pass on to this algorithm, and it is the part responsible for generalizing the instances into a classifier. We use forward-selection with compound operators because backward selection makes little sense for this null induction algorithm (it will predict majority for any instance not in the training set). Because, as we will explain below, the accuracy estimation is fast, we increase the stale parameter for the wrapper from five to ten, allowing more feature subsets to be searched.

One of the main problems with the wrapper approach is the time it takes to evaluate a node (a feature subset in our case). When k-fold cross-validation is used as the accuracy estimator, the induction algorithm has to be executed k times. We now explain how to speed up cross-validation time for algorithms that support incremental addition and deletion of instances. We feel that this digression is important because the simple idea of incremental cross-validation is what makes the IDTM algorithm practical.

The idea in incremental cross-validation is that instead of training k times on the union of $k-1$ folds each time, we train once on the full dataset, then for each fold we delete the instances in the fold, test on the fold instances, and insert the instances back. The delete-test-insert phase is repeated for each of the k folds. The name incremental is used because this type of cross-validation can be applied to any incremental algorithm that supports addition and deletion of instances. If the induction algorithm is guaranteed to produce the same classifiers in incremental mode as in batch mode, this incremental version of cross-validation is guaranteed to produce the exact same accuracy estimate as batch cross-validation.

Proposition 3 (Incremental Cross-Validation)
The running time of incremental cross-validation is

$$O(T + m(t_d + t_c + t_i)) \ ,$$

where T is the running time of the induction algorithm on the full dataset, m is the number of instances, and t_d, t_c, and t_i represent the time it takes to delete an instance, classify an instance, and insert an instance, respectively.

Proof: Incremental cross-validation starts out by running the original induction algorithm on the full dataset. Since each instance appears in exactly one fold, it is deleted once, classified once, and inserted once during the overall incremental cross-validation phase. ∎

Example 5.1 (Incremental Cross-Validation)
Conducting k-fold cross-validating of a decision tree induction algorithm and a dataset is deemed an expensive operation because one typically builds k decision trees from scratch, one for each fold. However, Utgoff (1994, 1995) shows how to incrementally add and delete instances in a manner that is guaranteed to generate the same tree as a batch algorithm. Thus, one can incrementally cross-validate decision trees much faster.

Nearest-neighbor algorithms support incremental addition and deletion of instances by simply adding and removing prototype points. Since these operations are fast, it can be shown that incremental cross-validation of a dataset with m instances and n features with a simple nearest-neighbor induction algorithm takes $O(m(m \cdot n))$ time; incremental cross-validation of a weighted regression nearest-neighbor takes $O(m(m \cdot n^2 + m^3))$ time as shown in Maron & Moore (1994) and Moore, Hill & Johnson (1992). ∎

We now describe the data structures that allow fast incremental operations on DTMs. The underlying data structure that we use is a universal hash table (Cormen, Leiserson & Rivest 1990). The time to compute the hash function is $O(n')$ where n' is the number of feature values in the DTM's schema, and the expected lookup time (given the hashed value of the instance) is $O(1)$ if all objects (unlabelled instances) stored are unique. To ensure that all stored objects are unique, we store with each unlabelled instance ℓ counter values, where ℓ is the number of label values. Each counter value c_i represents the number of instances in the training set having the same underlying unlabelled instance and label l_i.

To classify an instance, the unlabelled instance is found in the hash table and the label matching the highest counter value is returned.[1] The overall expected time to classify an instance is thus $O(n' + \ell)$.

[1]The running time could be further decreased to $O(n')$ by computing the majority of every unlabelled instance in advance, but the counters are needed for the incremental operations.

To delete an instance, the underlying unlabelled instance is found and the appropriate label counter is decreased by one; if all counters are zero, the underlying unlabelled instance is deleted from the table. To insert an instance, we search the underlying unlabelled instance; if it is found, the appropriate label counter incremented; otherwise, a new instance is added. Class counts must be kept for the whole body of the DTM in order to change the majority class.

Corollary 4 (Incremental Cross-Validation of IDTMs)
The overall time to cross-validate an IDTM with n' features in the schema and a dataset with m instances and ℓ label values is $O(m(n' + \ell))$.

Proof: All DTM operations have time complexity $t_d = t_c = t_i = O(n' + \ell)$. The overall time to build a DTM from scratch is the same as m insertions; thus by Proposition 5.1, the overall time for the cross-validation $O(m(n' + \ell))$. ∎

5.4 Experiments with IDTM

The fundamental principle of science, the definition almost, is this:
the sole test of the validity of any idea is experiment.
—Richard Phillips Feynman (1918-1988)

Our experiments with IDTM are aimed at showing the power of feature subset selection, but the wrapper approach is not perfect because it does a limited search and because it uses cross-validation at each node (feature subset) for accuracy estimation. In a manner similar to the upper bound computed on the C4.5 wrapper, C4.5*, in Section 4.9 on page 119, we compute an upper bound on the accuracy of IDTM, which we refer to as IDTM*. The upper bound is computed by using the test set to estimate the accuracy of each feature subset in the wrapper (but the test set is *not* used in building the DTM's body) and by increasing the stale parameter of the wrapper to 30. We stress that this upper bound may not be achievable in practice because we may be overfitting the test set. IDTM* is an upper bound on the possible performance, *not* an induction algorithm.

Table 5.1 and Figure 5.1 show a comparison of C4.5 and IDTM. The following observations can be made:

1. On the real datasets and at the 90% confidence level, IDTM significantly outperforms C4.5 on three datasets (cleve, DNA, Pima) and is significantly inferior on only

Table 5.1: A comparison of C4.5, IDTM, and IDTM*. The p-val column indicates the probability that IDTM is better than C4.5.

	Dataset	C4.5	IDTM	p-val	IDTM*
1	breast cancer	95.42± 0.7	95.42± 0.7	0.50	98.71±0.50
2	cleve	72.30± 2.2	75.24± 2.3	0.91	92.40±2.04
3	crx	85.94± 1.4	85.51± 1.3	0.37	91.74±0.68
4	DNA	92.66± 0.8	94.60± 0.7	1.00	94.94±0.64
5	horse-colic	85.05± 1.2	83.68± 1.4	0.15	92.94±1.08
6	Pima	71.60± 1.9	76.04± 1.5	0.99	78.24±1.38
7	sick-euthyroid	97.73± 0.5	97.35± 0.5	0.21	97.44±0.49
8	soybean-large	91.35± 1.6	85.64± 1.2	0.00	92.23±0.88
9	corral	81.25± 3.5	100.00± 0.0	1.00	100.00±0.00
10	m-of-n-3-7-10	85.55± 1.1	77.34± 1.3	0.00	99.22±0.28
11	Monk1	75.69± 2.1	100.00± 0.0	1.00	100.00±0.00
12	Monk2-local	70.37± 2.2	100.00± 0.0	1.00	100.00±0.00
13	Monk2	65.05± 2.3	64.35± 2.3	0.38	81.94±1.85
14	Monk3	97.22± 0.8	97.22± 0.8	0.50	97.22±0.79
	Average real:	86.51	86.69		92.33
	Average artif.	79.19	89.82		96.40

one (soybean-large). On the other four datasets, C4.5 is slightly better, but not significantly. The average of the two inducers is about the same. Performance on soybean-large is bad partly because it has 19 classes. Predicting majority when a match cannot be found in the table is very inaccurate when only 13% of the instances have the majority label.

2. On the artificial datasets, IDTM significantly outperforms C4.5 on three datasets (corral, Monk1, and Monk2-local) and was significantly outperformed on only one (m-of-n-3-7-10), where the wrapper did not find the optimal feature subset. The average accuracy for IDTM is much higher: a 51.1% reduction in error rate.

3. Table 5.2 shows that the number of features used by IDTM is small. In fact, it is so small that for many datasets, we can easily project the space onto two dimensions using General Logic Diagrams (GLDs). The GLDs are shown in Appendix B.

4. The upper bounds shown by IDTM* are very high for some datasets, but close to the performance of IDTM on others (*e.g.*, DNA and sick-euthyroid barely improved). It

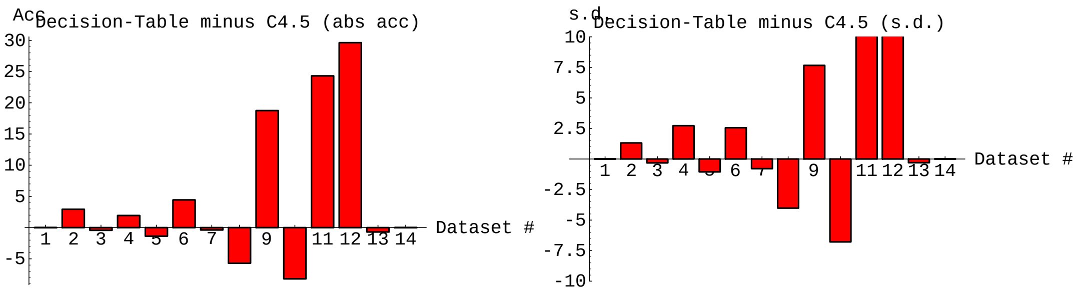

Figure 5.1: IDTM minus C4.5: Absolute difference in accuracy (left) and in std-devs (right). The std-devs graph is truncated at ten standard deviations.

is conceivable that with a better accuracy estimation, the performance of IDTM will improve.

The running times were 5.6 hours for DNA, 2.7 hours for sick-euthyroid, 40 minutes for soybean-large, and less than 15 minutes for all the other datasets. The results demonstrate that IDTM can achieve high accuracy in both artificial and real-domains using the simple hypothesis space of DTMs.

5.5 Related Work

Because they permit one to display succinctly the conditions that must be satisfied before prescribed actions are to be performed, decision tables are becoming popular in computer programming and system design as devices for organizing logic.
—Reinwald & Soland (1966)

In the early sixties, algorithms were created to convert decision tables into optimal computer programs (decision trees) under different measures of optimality using branch and bound procedures (Reinwald & Soland 1966, Reinwald & Soland 1967). In the early seventies, these procedures were improved using dynamic programming techniques (Garey 1972, Schumacher & Sevcik 1976). Hyafil & Rivest (1976) showed that building an optimal decision tree from instances (or from a table) is NP-complete. Hartmann, Varshney, Mehrotra & Gerberich (1982) showed how to convert a decision table into a decision tree using mutual information. The algorithm is very similar to ID3. All these approaches,

Table 5.2: The number of features in the dataset, the number used by C4.5, and the number selected by IDTM

	Dataset	Original dataset	C4.5	IDTM
1	breast cancer	10	7.0	2.8
2	cleve	13	9.1	4.0
3	crx	15	9.9	4.7
4	DNA	180	46	11
5	horse-colic	22	5.5	4.0
6	Pima	8	8.0	4.0
7	sick-euthyroid	25	4	2
8	soybean-large	35	22.0	9.4
9	corral	6	4	4
10	m-of-n-3-7-10	10	9	0
11	Monk1	6	5	3
12	Monk2-local	17	12	6
13	Monk2	6	6	3
14	Monk3	6	2	2

however, dealt with conversions that are information preserving, *i.e.*, all entries in the table are correctly classified and the structures are not used for making predictions.

The rough sets community has been using the hypothesis space of decision tables for a few years (Pawlak 1987, Pawlak 1991, Slowinski 1992). Researchers in the field of rough sets suggested using the degrees-of-dependency of a feature on the label (called γ) to determine which features should be included in a decision table (Ziarko 1991, Modrzejewski 1993). Another suggestion was to use normalized entropy (Pawlak, Wong & Ziarko 1988), which is similar to the information gain measure of ID3 and CART.

Almuallim & Dietterich (1991) described the FOCUS algorithm which is equivalent to finding the DTM with the smallest number of features in the schema, such that there are no conflicting instance projections if there were none originally. An exhaustive search was conducted to find this smallest feature subset, making the algorithm impractical in practice. Almuallim & Dietterich (1992*a*) described FOCUS-II, which greedily selects features to reduce the conditional entropy of the label; FOCUS-II thus finds a small set of features that reduces the entropy to zero. The main problem with FOCUS and FOCUS-II is that they do not regularize or prune. In both versions, the features were selected until there were no conflicting instances.

Almuallim & Dietterich (1992*b*) discussed the "Multi-balls" algorithm that has **high coverage**: for a given sample size, the number of concepts it can learn in Valiant's PAC model (Valiant 1984, Angluin 1992) is close to the upper bound of any learning algorithm. DTMs can be viewed as a multi-balls hypothesis space because the centers are equidistant by their definitions. However, the induction method is completely different, and the maximal number of balls a DTMs creates is less than the upper bound given by the Gilbert-Varshamov bound, which the authors use.

A nearest-neighbor algorithm can be viewed as a generalization of a DTM with a zero weight on each feature not included in the schema. However, while nearest-neighbor algorithms use the nearest neighbor to classify instances, a DTM classifier defaults to the majority whenever the distance is greater than zero. Langley & Sage (1994*b*) used oblivious decision trees in an algorithm called **Oblivion**. The hypothesis space is similar to decision tables because the space is partitioned on a subset of features. However, when a perfect match in the table is not found, the nearest-neighbor is used.

5.6 Summary

We have used a simple hypothesis space, the space of decision tables with a default majority rule (DTMs), to test the conjecture that feature subset selection is a very powerful bias (in the machine learning sense of bias). The average accuracy of IDTM on the real-world datasets tested was equivalent to C4.5 and higher for the artificial datasets. We thus believe that this bias is appropriate for problems similar to those we used. The fact that a small subset of relevant features suffices to learn accurate classifiers is one good example that the uniform assumption over target concepts that is made in the no-free-lunch theorems (Wolpert 1994*b*, Schaffer 1994) is irrelevant (pun intended) to the real world.

We have shown that the resulting decision tables are *very* small and use few features, allowing one to concisely display them using General Logic Diagrams (Appendix B). Alternatively, one can convert the decision tables into oblivious decision trees or graphs (see future work in Section 6.8 on page 187).

The ability to incrementally cross-validate the IDTM algorithm and a dataset in time that is linear in the number of instances, the number of features, and the number of label values, makes the wrapper approach feasible for large problems that would be impractical with other induction algorithms. The subset found for DTMs might also serve as a good starting feature subset for running the wrapper with other induction algorithms. Of course, it is conceivable that better and faster approaches to feature subset selection exist, that do not use the wrapper at all.

Our goal in this chapter has not been to claim that decision tables provide a very good hypothesis space for induction algorithms; rather, we have shown that such a simple hypothesis space can lead to high performance, a point previously made by Holte (1993), although he used a different algorithm. The IDTM algorithm described here performs much better than Holte's algorithm, which was significantly inferior to C4.5, and it also confirms our conjecture that global feature subset selection alone, with an extremely simple classifier, provides good generalization ability.

DTMs can be improved in some obvious ways. The weakest point of the hypothesis space is the use of the training set's majority label when a perfect match is not found. This can be replaced with something more sensible, such as finding a match on fewer features, which would better estimate the density at the given test instance. Another weak point is the fact that missing values are considered distinct values, while research indicates that this is usually the worst possible approach to handling them (Quinlan 1989).

Chapter 6

Oblivious Read-Once Decision Graphs

Trying to display the "prettiest" arrangement of nodes and arcs that diagrams a given graph is like nailing jelly to a tree, because nobody's sure what "prettiest" means algorithmically.
—The AI Hackers Dictionary

Decision trees have proven to be a good hypothesis space for many real-world concepts. Top-down recursive-partitioning algorithms for inducing decision trees generally yield accurate classifiers, and the trees that are produced are generally comprehensible if they are not too big. In this chapter, we look at some of the limitations of decision trees and top-down recursive-partitioning algorithms. We propose a different hypothesis space—oblivious read-once decision graphs—and investigate its advantages and disadvantages. We propose an algorithm for inducing oblivious decision graphs and investigate its advantages and limitations. We then use the wrapper approach and an entropy-based method to overcome many of the limitations. We experimentally compare our algorithm with C4.5, both in terms of accuracy and in terms of comprehensibility.

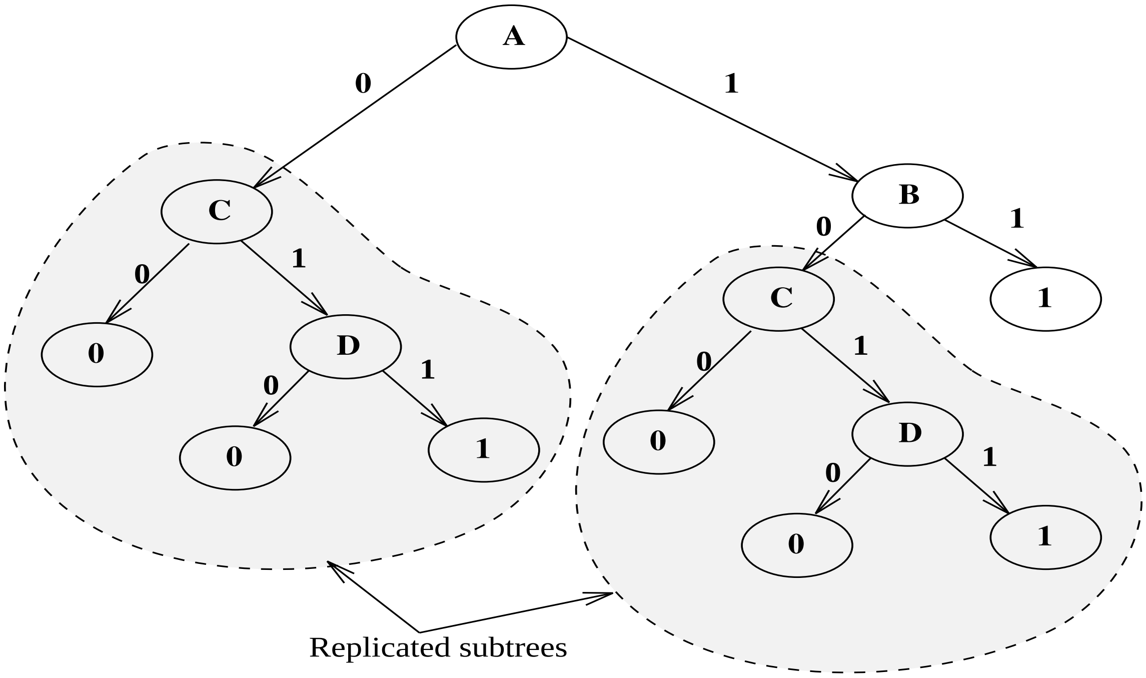

Figure 6.1: The smallest possible tree for the concept $(A \wedge B) \vee (C \wedge D)$. Note how one term ($C \wedge D$ in the figure) must be replicated.

6.1 Introduction

The way to eliminate bureaucracy and flatten organization is by reengineering the processes so that they are no longer fragmented. Then the company can manage without its bureaucracy.
—Hammer and Champy, Reengineering the Corporation, 1993

Top-down algorithms for inducing decision trees generally yield accurate classifiers, and the trees that are produced are generally comprehensible if they are not too big. As mentioned in Chapter 2, Michie (1987) reported that when ID3's output on the chess domain was shown to a domain expert, *i.e.*, a chess master, it was completely opaque. Although it was very accurate, the tree was large, obscure, and the chess master was in a "total blackout."

The tree structure and existing algorithms for inducing decision trees suffer from some well-known problems, most notably the replication problem and the fragmentation problem (Cendrowska 1987, Pagallo & Haussler 1990).

The **replication** problem is exemplified by the concept $(A \wedge B) \vee (C \wedge D)$ shown in

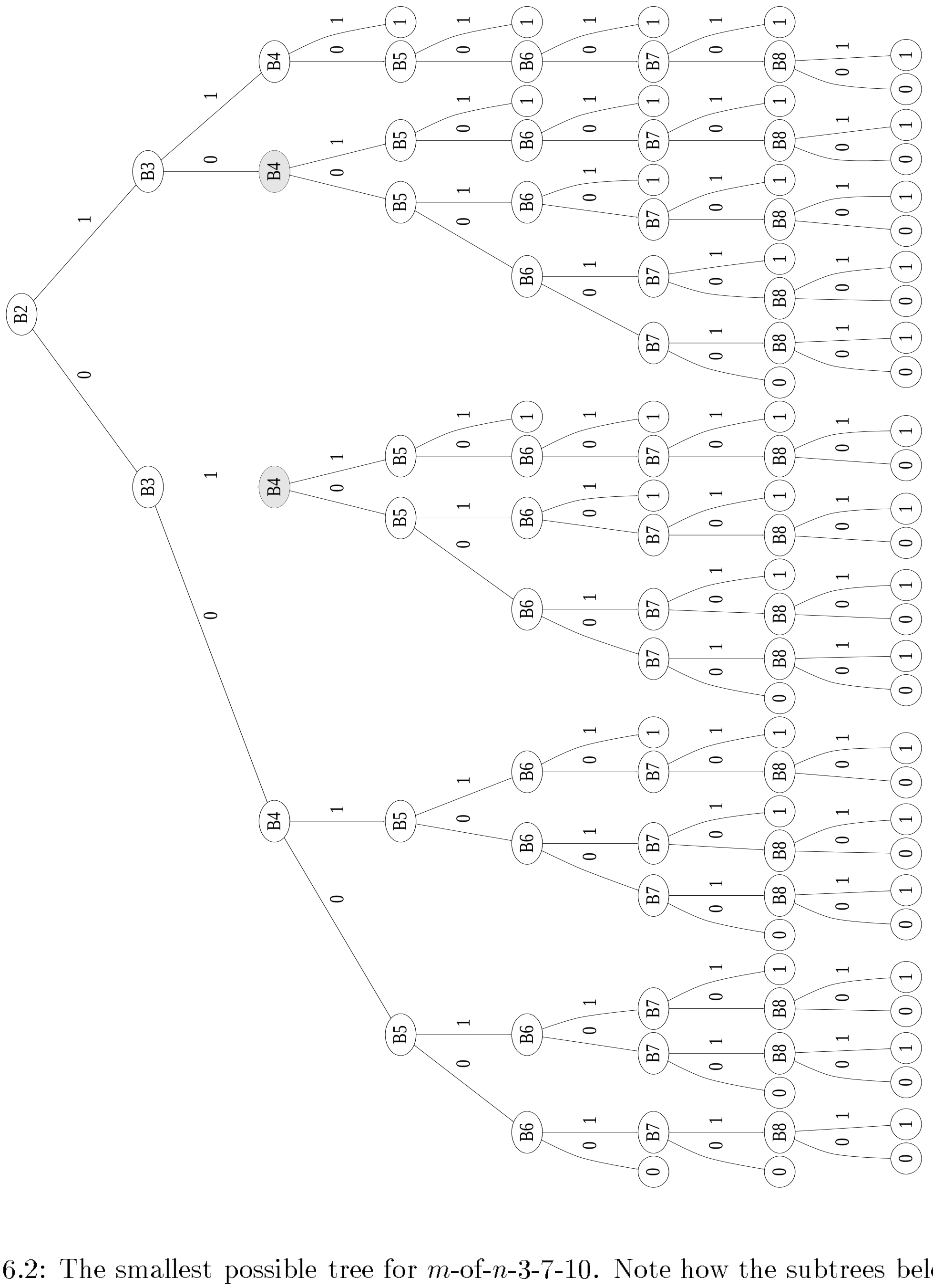

Figure 6.2: The smallest possible tree for m-of-n-3-7-10. Note how the subtrees below the grey nodes are isomorphic.

Figure 6.1, where the tree corresponding to the term ($C \wedge D$) is replicated. Decision trees must include multiple copies of subtrees implementing terms in disjunctive concepts. A more extreme example is the decision tree shown in Figure 6.2, where the smallest decision tree for the m-of-n-3-7-10 concept is shown. The tree has 111 nodes and 56 leaves, and many subtrees are replicated many times. For example, the subtree below B2=1 and B3=0 is isomorphic to the subtree below B2=0 and B3=1 (both shown in grey in the figure).

The replication problem is an inherent limitation of the decision tree structure and it is independent of the induction algorithm. The smallest decision trees for most **symmetric functions**—functions which yield the same value for all permutations of the input features—are exponentially-sized. Symmetric functions, such as m-of-n, are known to occur in medical domains (Spackman 1988) and were useful in converting neural-networks to decision rules (Towell & Shavlik 1993, Craven & Shavlik 1993). Zheng (1995) recently showed that adding X-of-N splits to C4.5 sometimes improves the accuracy on real-world domains. (X-of-N counts the number of features that have pre-specified values; the split is then done on a threshold value.)

The **fragmentation** problem is the problem of having the data fragmented at lower levels of the tree. The fragmentation problem arises because of the top-down recursive partitioning method commonly used to build decision trees. All common algorithms find the best test to conduct at the root node of the tree (see Section 2.2 on page 19), then split the instances according to the test, and finally solve the subproblems recursively. If each binary test satisfies about half the data, then after ten splits only 1/1024th of the data trickles down to a given node. If the dataset contains m instances and the resulting tree is approximately balanced, then the paths cannot be much longer than $\log_2 m$, and the tests conducted at lower levels are not really meaningful because they are based on a small number of instances.

Pimat, Kononenko, Janc & Bratko (1989) report that during real experiments in medical domains, physicians were not prepared to use the induced decision trees in practice because the paths were too short and contained only the most informative features, which they felt poorly described the patient, in order to make a reliable decision.[1]

The fragmentation problem is usually more severe when multi-valued features create multi-way splits, as is the case with C4.5. For example, Figure 6.3 shows the unpruned

[1]Although the physicians were not prepared to use the decision trees in practice, the decision trees did outperform them in terms of accuracy.

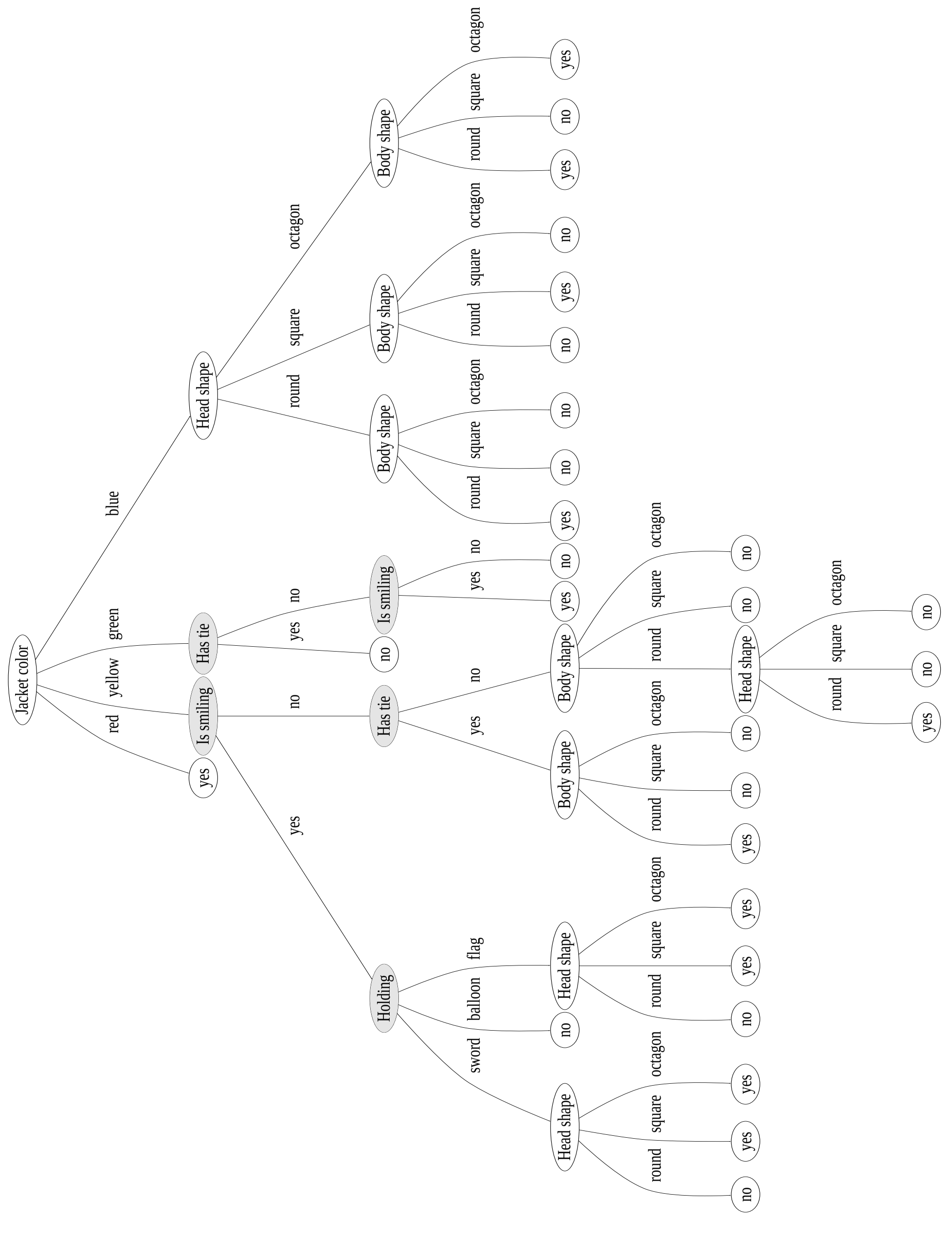

Figure 6.3: The unpruned tree induced by C4.5 on the Monk1 problem "(head-shape = body-shape) or (jacket-color = red)." Shaded nodes mark tests on irrelevant features. Pruning replaces the whole subtree below (jacket-color = yellow) with a leaf marked "no."

tree induced by C4.5 on the Monk1 problem. The shaded nodes test irrelevant features; one reason for the splits on irrelevant features is that the first split is a multi-way split that fragments the data. Quinlan (1993, p. 64) claims that groupings should be determined from background knowledge and given in additional features. Promoting comprehensibility and avoiding artificial groupings, he wrote that "Against the requirement for binary splits is the argument that they can force artificial divisions—some partitions are inherently multi-valued, as illustrated by medical *triage*." Although binary partitions are usually considered a partial solution to the fragmentation problem, Kononenko (1995*a*) showed a counter-example to the strong claim that they are always superior.

In this chapter, we describe Oblivious Read-Once Decision Graphs (OODGs) as an alternative hypothesis space for supervised classification learning. OODGs retain most of the advantages of decision trees, while overcoming the replication problem mentioned above; the induction algorithm avoids the inherent fragmentation in recursive partitioning. OODGs are similar to Ordered Binary Decision Diagrams (OBDDs) (Bryant 1986), which have been used in the engineering community to represent state-graph models of systems, allowing verification of finite-state systems with up to 10^{120} states.[2]

The chapter is organized as follows. In Section 6.2, we formally define OODGs and some variants. In Section 6.3, we describe the properties of OODGs and investigate their representational power. In Section 6.4, we describe the basic bottom-up algorithm for inducing OODGs, but leave the two main issues open: feature ordering and incomplete projections. In Section 6.5, we show that the problem of incomplete projections cannot be easily solved because the decision problem is NP-hard; we then propose a greedy heuristic for incomplete projections and for feature orderings and discuss some of the limitations of the approach. In Section 6.6, we use the wrapper approach to select an ordering on the features, which indirectly provides a pruning mechanism. We present experimental results and comparisons with C4.5. We discuss related work in Section 6.7, suggest future work in Section 6.8, and summarize in Section 6.9

[2] The name OODG was originally preferred over OBDD because the term *decision graph* was already used in the machine learning community by (*e.g.*, Oliver (1993)) when OODGs were introduced in Kohavi (1994*a*). The features in many problems are neither binary, nor even discrete (*i.e.*, they are continuous), and ODD (ordered decision diagrams) was not a familiar acronym.

6.2 Definitions

By definition, one divided by zero is undefined.
—Unix fortune file

In this section, we formally define the OODG structure and describe some variants. The name OODG is a combination of the terms "Oblivious" and "read-Once" that are used in theoretical complexity analysis of branching programs, and the term "Decision Graph" that is used in the machine learning community, most notably by Oliver (1993).

Without loss of generality, we assume that the classes are labelled 0 to $k-1$ and that for each class there is at least one training instance that maps to it. We also initially assume that the domains for all features are discrete. We will concentrate on deterministic concepts in this chapter. We can always transform a probabilistic concept into a deterministic one by mapping each unlabelled instance to the class with the highest posterior probability for that unlabelled instance.

6.2.1 Decision Graphs

We begin with a description of decision graphs and then impose restrictions that make the decision graph oblivious and read-once.

A **k-classification function** is a deterministic function f mapping each unlabelled instance in the instance space to one of k classes, *i.e.*, $f : \vec{x} \mapsto \{0, \ldots, k-1\}$.
A **decision graph** for a k-classification function over features $X_1, X_2, \ldots, X_n$ with domains $\mathrm{Dom}(X_1)$, $\mathrm{Dom}(X_2)$, $\ldots$, $\mathrm{Dom}(X_n)$, is a directed acyclic graph (DAG) with the following properties:

1. There are exactly k leaf nodes (nodes with outdegree zero), called **category nodes**, that are labelled $0, 1, \ldots, k-1$.
2. Non-category nodes are called **branching nodes.** Each branching node is labelled by a feature X_i and has $|\mathrm{Dom}(X_i)|$ outgoing edges, each labelled by a distinct value from $\mathrm{Dom}(X_i)$. In the figures shown below, we draw one edge labelled "always" if all edges emanating from a given node branch to the same node.
3. There is one distinguished node—the **root**—that is the only node with indegree zero.

The class assigned by a decision graph to a given instance is determined by tracing the unique path from the root to a category node, branching to the appropriate edge label

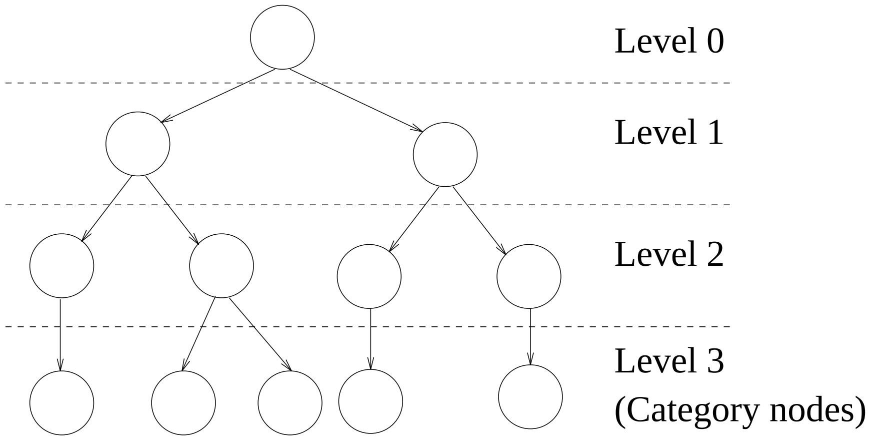

Figure 6.4: A levelled graph can be divided into levels, such that outgoing edges from each level terminate at the next level. For example, an edge from the root to level two is not allowed.

corresponding to the test outcome at each node. We now describe restrictions on the general graph structure, that together yield the oblivious read-once decision graph structure:

Read-once Each feature occurs at most once along any path from the root. Although in a decision tree branching multiple times on the same discrete feature gives no extra power (because all instances in the subtree have the same value on a feature tested above), branching twice on the same feature in a decision graph is known to increase the representation power (see Section 6.2.2 on page 151).

Levelled The nodes in the graph are partitioned into a sequence of pairwise disjoint sets, the levels, such that outgoing edges from each level terminate at the next level. Figure 6.4 shows the levels of a levelled decision graph. This restriction is weak because it does not change the polynomial-size representation power (polynomial in the number of features). Levelling a graph can only square its size, as dummy nodes can be added for edges that jump to lower nodes in the graph.

Oblivious All nodes at a given level of a levelled decision graph are labelled by the same feature. The name "oblivious" denotes the fact that testing of features depends only on their order within the levels, independent of the input instance. The oblivious restriction strictly limits the number of functions representable in polynomially sized

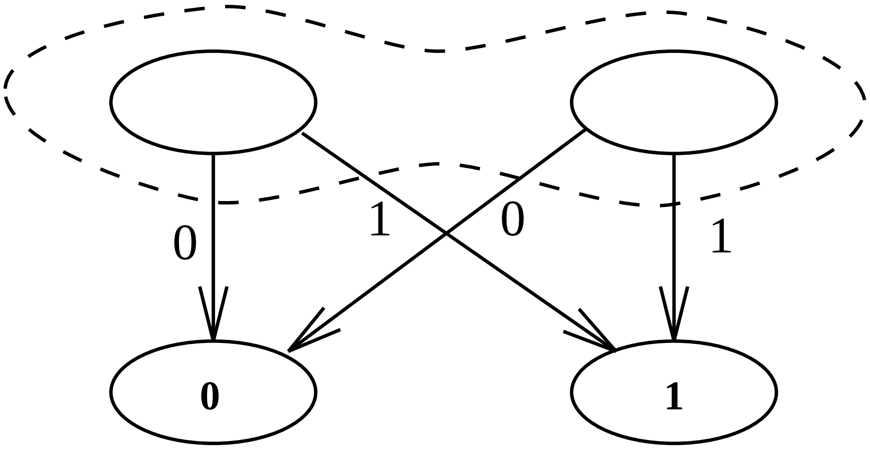

Figure 6.5: Two nodes that branch the same way must be collapsed in a reduced graph.

decision graphs (see Section 6.2.3 on page 152).

Reduced There do not exist two distinct nodes at the same level that branch in exactly the same way on the same values. If two such nodes exist, they can be collapsed into one, as shown in Figure 6.5

An **OODG** is a reduced, oblivious, read-once decision graph. Figure 6.6 shows an OODG for the Monk1 problem. The **size** of an OODG is the number of nodes in the graph (14 in Figure 6.6). The **width** of a level is the number of nodes at that level. The **width** of a graph is the width of the widest level in the graph (three in Figure 6.6). We will assume the OODG is drawn top-down, so that the root is the **highest** node and the leaves are the **lowest** nodes. We will use the term **oblivious tree** to denote a tree where all branching nodes test the same feature at a given level.

A **constant node** in an OODG is a branching node where all emanating edges terminate at the same node of the subsequent level; such a node ignores the tested feature. For example in Figure 6.6, the left node testing jacket-color is constant, as are the nodes testing the features has-tie, is-smiling, and holding. If a level has only constant nodes, the feature tested at the level is irrelevant to the implemented function, and the whole level can be removed from the graph without changing the implemented function. The levels testing the features has-tie, is-smiling, and holding in Figure 6.6 can thus be removed. Furthermore, for display purposes, constant nodes can also be removed, with the incoming edges redirected

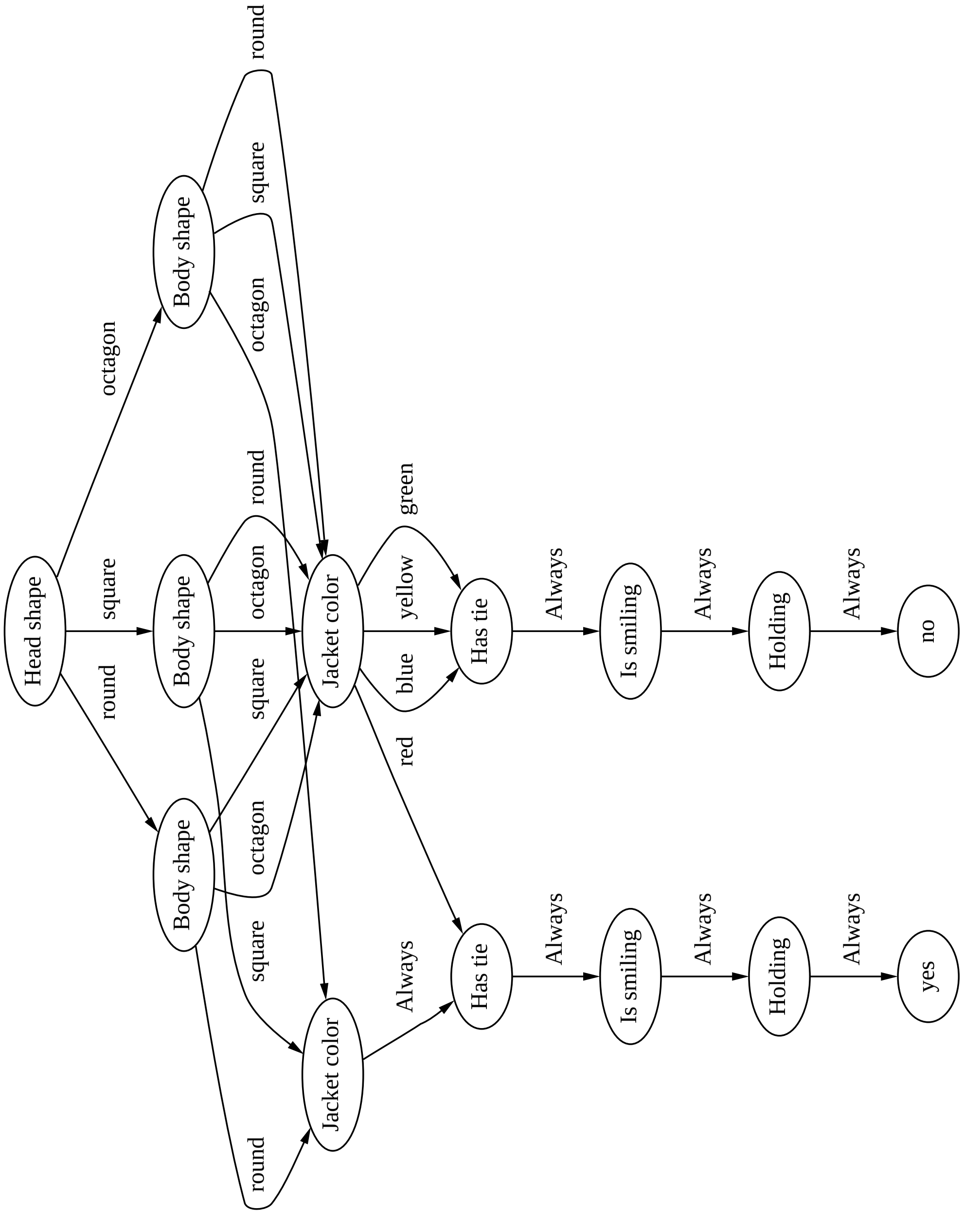

Figure 6.6: An OODG for the Monk1 problem "(head-shape = body-shape) or (jacket-color = red)."

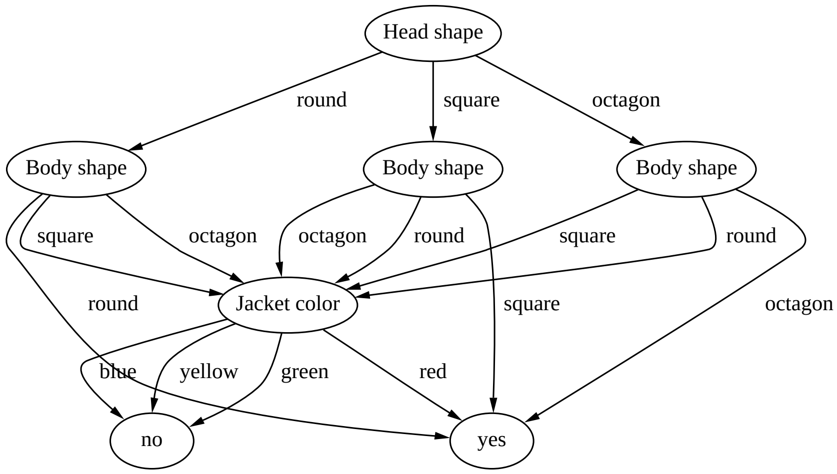

Figure 6.7: An OODG for the Monk1 problem with constant nodes removed.

to the unique destination node of the edges emanating from the constant node. Figure 6.7 shows the same graph as in Figure 6.6 with constant nodes removed. The graph is not oblivious any more, but it can always be made oblivious without an exponential increase in size by adding the constant nodes back.

6.2.2 Comments and Variants of OODGs

While the relation between many complexity classes in computer science is unknown (*e.g.*, $P \stackrel{?}{=} NP$), the two main restrictions for OODGs, *i.e.*, read-once and oblivious, both strictly restrict the number of functions representable by polynomially-sized graphs (polynomial in the number of features). An excellent overview of the different decision graph classes and restrictions is given in Meinel (1992). We now review some of the main results that are relevant to understanding the structure.

Read-Once Decision Graphs

Wegener (1987) proved that the class of polynomially-sized read-once decision graphs is a strict subset of the class of polynomially-sized read-twice decision graphs. Masek (1976) gives an ingenious example, attributed to Michael Fredman, where reading features more than once can be useful in reducing the size of the decision graph for a natural function. A sketch of the idea is given in the example below.

Example 6.1 (Read-many is more powerful than read-once)
Let f be the function that is one iff the sum of the n input bits is m. It can be shown that a read-once decision graph must have at least $m(n-m+1)+2$ nodes in the graph because the function must update the count of ones (0 to m) each time a feature is read (Masek 1976). If at any interior level (excluding the top m and bottom $\log_2 m$) there are less than m nodes, then they cannot represent the sum of ones, and the graph cannot compute the correct function.

If, however, input features can be read multiple times, then the following scheme can be used. The residuals of the n input bits are computed modulo different prime numbers. For each prime number p, the count of ones modulo p can be computed using $O(n \cdot p)$ nodes. For a given number m, and for the first k prime numbers $p_1, \ldots, p_k$ whose product is greater than n, we can precompute $m_i = m \bmod p_i$. By the Chinese Remainder Theorem (Hardy & Wright 1979), the set of m_i's is unique. The graph can then compute the residuals of m modulo p_i for every $1 \le i \le k$ in sequence (this is where the read-many is required). If all residuals agree with the precomputed values m_i given by the Chinese Remainder Theorem, then output one, otherwise output zero. By the Prime Number Theorem (Hardy & Wright 1979), the largest prime needed is $O(\log n)$ and there are $O(\log n / \log\log n)$ primes between 2 and $\log n$. Overall, we need $O(n \log^2 n / \log\log n)$ nodes in the decision graph, a nice improvement over the read-once decision graph, which requires $\Theta(n^2)$ for $m = \Theta(n)$.[3] ∎

6.2.3 Oblivious Decision Graphs and OBDDs

The oblivious restriction by itself restricts the class of polynomially-sized functions representable in a decision graph. It is known that for some functions there exists a polynomially sized decision graph but not a polynomially-sized oblivious decision graph of depth $O(n)$; moreover, there exists a function for which the polynomially-sized decision graph is read-once and there is no read-many polynomially-sized oblivious decision graph that is of depth $O(n)$ (Meinel 1992).

The oblivious and read-once restrictions together are equivalent to defining a total ordering on the features, and hence binary OODGs are isomorphic to Ordered Binary Decision Diagrams (OBDDs) (Bryant 1986), which will be discussed further in the related work

[3]The Θ notation indicates upper and lower asymptotic bounds up to constant factors. For example, m could be $n/2$.

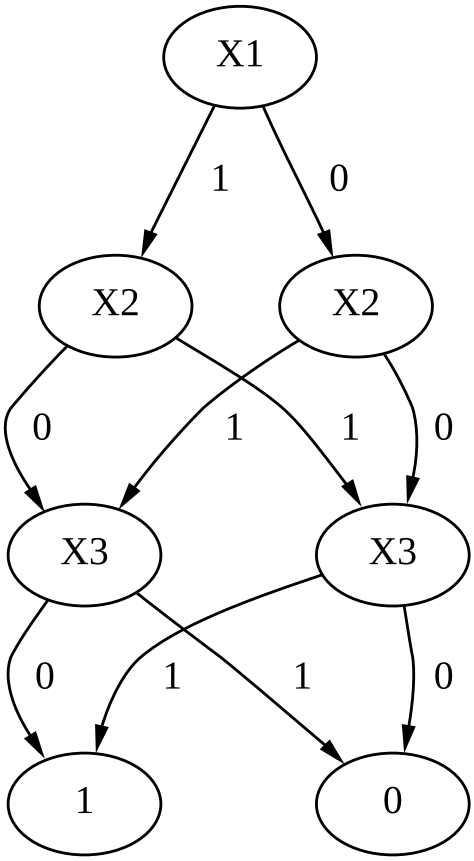

Figure 6.8: An OODG for 3-bit parity. $f = X_1 \oplus X_2 \oplus X_3$.

section (Section 6.7 on page 182). We prefer to work with the oblivious read-once decision graph structure because it is the natural representation for the learning algorithms presented in Section 6.4.

The class of polynomially-sized decision graphs clearly includes the class of polynomially-sized decision trees because every tree is also a graph. The added restrictions—read-once and oblivious—change the class and makes OODGs and decision trees incomparable: some functions can be represented in polynomially-sized OODGs but not in polynomially-sized decision-trees and vice-versa:

1. The parity function, $f = X_1 \oplus X_2 \oplus \cdots \oplus X_n$ (where $\oplus$ denotes the exclusive-or operation), is an example of a function that can be implemented in a small decision

graph of size $2n+1$, yet any decision tree implementing parity must have $2^{n+1}-1$ nodes.

Figure 6.8 shows an OODG for 3-bit parity. Nodes on the left side of the "parity ladder" are reached when the parity is odd; nodes on the right side are reached when the parity is even. A feature value of one causes a change of sides (as the parity changes), while a value of zero leads down the same side.

2. Breitbart, Hunt III & Rosenkrantz (1991) showed that there exists a function over n features that is representable in a tree of size $O(n^2/\log n)$ but for which the smallest OODG is of size $2^{\frac{n}{\log n}-2} \cdot (1-\epsilon_n)$, where ϵ_n is a small number depending on n. The function is a variant of a double multiplexer described as follows. There are $k+2^k$ input features, and define m to be $\lfloor 2^k/k \rfloor$. Partition the $k+2^k$ features into $m+1$ groups, each consisting of k features. Let the first k features be used as an index to determine which subset of k features to index (if the number represented in the first k bits is greater than m, the function is zero). Once the group of k bits is indexed, we use it as a second level index that selects a single bit from the 2^k bits. The selected bit is then the output of the function.

Because OODGs and decision trees are incomparable, there will be problems for which one is better suited than another. If features are relevant all throughout the space, then the oblivious restriction is appropriate; however, if different subspaces have different relevant features, as is the case with multiplexer-type functions (see below), then forcing the graph to be oblivious will require learning a large structure and the learning rate will be slower.

Although we are investigating the representation power of OODGs and trees, it is worth noting that the same problems for which decision tree structures are more compact are also those for which the greedy measures used by most algorithms for building decision trees are ill-suited, as the following two examples show.

Example 6.2 (Union of two datasets)

Suppose we take two datasets and create a combined dataset containing the union of their features, plus an extra feature called "source." For each instance from the first dataset, we assign the value zero to the "source" feature and random values to the features not in the first dataset; for each instance from the second dataset, we assign the value one to the "source" feature and random values to the features not in the second dataset. We now have a combined dataset containing $m_1 + m_2$ instances, where m_1 and m_2 are the number of instances in the first and second datasets, respectively.

To make the example more concrete, suppose we were to combine the horse-colic and sick-euthyroid datasets. Each instance would be marked as "horse" or "person" in the added feature called "source," and the target concept would then be "a horse with a surgical lesion or a person that is sick."

The combined dataset is truly decomposable: once we test on the feature "source," we decomposed the original problem into the problem of learning the two original datasets separately, a much easier task. However, if the two datasets contain approximately the same proportions of each class, then there would seem to be no information gain from splitting on the feature "source" and other features would be chosen for the root. ∎

The next example shows a more extreme example of a decomposable problem.

Example 6.3 (Multiplexers are hard for decision tree inducers)
In the multiplexer domain, there are ℓ "address bits" and 2^ℓ "data" bits. An instance is labelled positive iff the data bit indicated by the numerical interpretation of the address bits is set to one.

Although multiplexer-type functions have a small representation in a decision tree, they are known to be very hard to learn using existing top-down recursive partitioning approaches (Wilson 1987). Quinlan (1988) observed that there is no information gain (mutual information is zero) from a split on an address bit at the root because due to symmetry there are an equal number of instances for each class in each child node. Decision tree inducers that do not employ ℓ-level lookahead will always pick a data bit at the root, creating a tree much larger than the best one.[4] ∎

As the examples show, practical learning algorithms are very limited in the search they conduct for small structures because finding the smallest structure that approximately fits the data is usually NP-hard (Hyafil & Rivest 1976, Hancock 1989, Judd 1988, Blum & Rivest 1992). It is both the type of search and the hypothesis space that ultimately determine the degree of success of an induction algorithm.

[4]Utgoff (1994, 1995) has recently proposed a "direct metric" for decision tree induction, which measures the depth of the tree that would be created if a given feature was used as the root split. The feature that minimizes this criterion is chosen to be the root split. The metric can be efficiently computed using an incremental decision tree algorithm, such as ITI, and seems to be well-suited for decomposable problems, including the multiplexer problem.

6.3 Properties of OODGs

Randomness: The property required to make
statistical calculation come out right.
—Unix fortune

In this section, we describe some properties of OODGs. We begin with the basic properties, which were known for OBDD and therefore hold for OODGs, then give an upper bound on the width of levels in OODGs, and finally give an adjacency condition that can be used to prove properties for the bottom-up construction algorithm. Note that there is no inductive process in this section and that all features are assumed to be discrete.

6.3.1 Basic Properties

The following properties are true for OODGs and OBDDs:

- An OODG can represent any discrete k-classification function.

  This property is easy to see: build an oblivious tree that splits on all the features in any order. The leaves of the tree uniquely define the value of the function at each set of values. The tree can then be collapsed into an OODG.

- Closure under negation: an OODG for a Boolean function f of size s can be converted into an OODG to implement $\overline{f}$ with no change in the size.

  This property is obvious: simply reverse the labels for the two leaves. Note that many hypothesis spaces, such as DNF (disjunctive normal form) and CNF (conjunctive normal form) are not closed under negation for polynomially-sized formulas.

- For any k-classification function f, and for a given ordering of the features for the levels, there is a unique (up to isomorphism) OODG implementing f.

  This property is not surprising because an OODG is very similar to a deterministic finite automaton that has been "unrolled" to avoid cycles. The proof for OBDDs can be found in Bryant (1986) and easily generalized to non-binary graphs.

- There exist functions that have a polynomial (or even linear) size OODG representation under one feature ordering, and an exponential size OODG under another ordering.

One such example for Boolean features is

$$(X_1 \wedge X_2) \vee (X_3 \wedge X_4) \vee \cdots \vee (X_{2n-1} \wedge X_{2n}).$$

The ordering $X_1, X_2, \ldots, X_{2n}$ yields a graph with $O(n)$ nodes, while the ordering $X_1, X_{n+1},\ X_2, X_{n+2},\ \ldots,\ X_n, X_{2n}$ requires at least 2^n nodes.

- There are functions for which no feature ordering results in a polynomial size OODG representation (the Shannon effect).

  This property can easily be shown using counting arguments. There are 2^{2^n} Boolean functions over n features, yet for any polynomial $p(n)$ representing an upper bound on the size of the class of OODGs allowed, the number of bits needed to describe the OODG is $O(p(n) \log p(n))$ (each node can be described by $O(\log p(n))$ bits for the node number and two edges). The number of Boolean functions that can be represented is thus only $2^{O(q(n))}$ for some polynomial q, and hence is vanishingly small when compared to the family of Boolean functions over n features.

  Wegener (1987, p. 417) showed that for almost all Boolean functions, the smallest branching programs (branching programs represent a larger family than OODGs) are at least of size $(1/3)2^n/n$. This result provides a tight asymptotic bound (up to constant multiplicative factors) because it is known that any Boolean function can be represented by a branching program of size $O(2^n/n)$.

  Bryant (1986) gave a "natural function" that blows up under all orderings. He showed that at least one of the $2n$ bits of integer multiplication requires an exponential sized OBDD (and hence OODG) for all orderings.

- All symmetric Boolean functions—functions which yield the same value for all permutations of the features—have OODGs of size $O(n^2)$. Examples of symmetric Boolean functions are parity, majority, "exactly k-of-n", and "at least k-of-n".

  Because all symmetric functions are only based on the count of the features having value one, the OODG needs to have at most n states per level; the total number of branching nodes is thus bounded by n^2. Representations such as decision trees, DNF (disjunctive normal form), and CNF (conjunctive normal form), all require exponentially-sized structures to represent such functions.

Many other properties of OODGs and OBDDs can be found in Bryant (1992), Bryant (1986), Meinel (1992), and Chakravarty (1993).

6.3.2 The Kite Theorem

We now present a simple theorem—the Kite Theorem—that gives an upper bound on the width of different levels of an OODG, given an instance space of Boolean features.[5] This theorem shows that the upper bound on the width of OODGs is asymmetric; the graph of the upper bound grows much faster from the bottom than from the top. In Figure 6.9, the width of the "kite" at each level is proportional to the maximum number of nodes possible at that level. The kite is thus an envelope bounding both the overall size and the specific width of every OODG with the given number of levels. This theorem was one of the motivations for designing a bottom-up learning algorithm (Section 6.4); a wrong ordering will explode fast.

Theorem 5 (Kite Theorem)
The width of level i of an OODG with n levels implementing a k-classification function over Boolean features is bounded from above by

$$\min\left\{2^i, k^{2^{(n-i)}}\right\}$$

Proof: The first term is a top-down bound; each Boolean feature can cause the number of branching nodes to grow by a factor of at most two. The second term is a bottom-up bound; if a level has k nodes, the level above it must have at most k^2 nodes because there are only k^2 mappings from $\{0, 1\}$ to $\{0, \ldots, k-1\}$, and each mapping uniquely defines a node at the level above (this is where the reduced property of OODGs is needed). ∎

For non-Boolean (but discrete) domains, the bound in the theorem becomes

$$\min\left\{\prod_{i=1}^{n} |\mathrm{Dom}(X_i)|\,, k^{\prod_{i=1}^{n}|\mathrm{Dom}(X_{n-i})|}\right\}\ .$$

The following corollary indicates that if the OODG is worse-case in terms of its size, then its widest level is not too far from the bottom.

[5] Pat Langley said that the Kite Theorem is what makes the whole idea of OODGs fly.

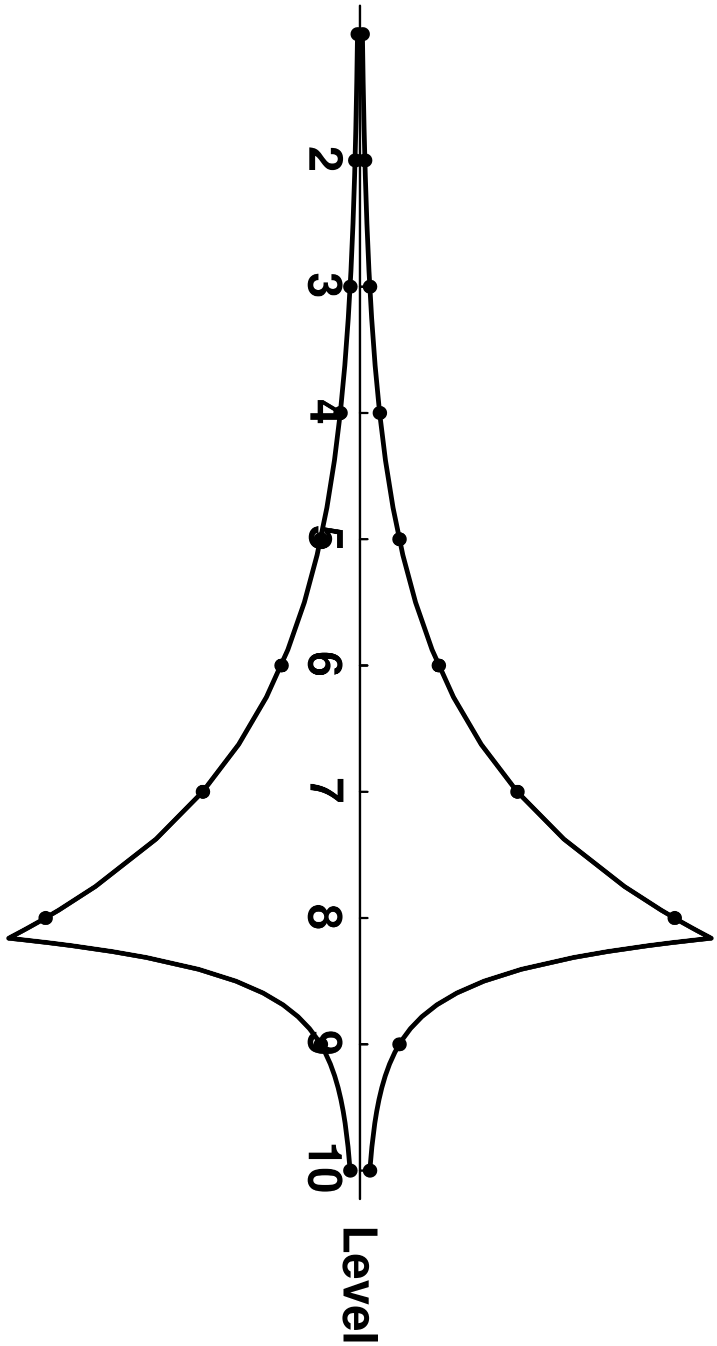

Figure 6.9: The diagram's width at a level is proportional to the maximum number of nodes possible in a binary OODG at that level.

Corollary 6

Let n, the number of features, be greater than 3, and let $\ell = n - \log n + \log\log k$ (all logs are base two here). Then the level with the highest bound on the width of an OODG on Boolean features is between $\lfloor \ell \rfloor$ and $\lceil \ell \rceil + 1$.

Proof: By Theorem 5, the maximum width is achieved when 2^i is equal to $k^{2^{(n-i)}}$. The following steps show the desired inequality.

1	$2^i = k^{2^{(n-i)}}$	From Kite Theorem
2	$i = 2^{n-i} \log k$	Take logs
3	$\log i = n - i + \log\log k$	Take logs again
4	$\log n \geq n - i + \log\log k$	$n \geq i$
5	$i \geq n - \log n + \log\log k$	
6	$\log i = n - i + \log\log k$	Step 3 again
7	$\log(n/2) \leq n - i + \log\log k$	By Step 5, $i \geq n - \log n \geq n/2$
8	$i \leq n - \log n + \log\log k + 1$	
$5+8$	$n - \log n + \log\log k \leq i \leq n - \log n + \log\log k + 1$	∎

Note that the Kite Theorem and its corollary give worst-case scenarios. Functions must be bounded within the kite, but may grow and shrink inside the bound. If, when an OODG is constructed bottom-up, the width of a level i is below the upper bound given by the Kite Theorem, we can re-apply the theorem on the remaining features using the width at level i as the k parameter, and thus derive a new upper bound on the final size of the OODG. If the width at level i is strictly below the upper bound given by the kite theorem, the new upper bound will be strictly smaller than the original upper bound.

6.3.3 The Adjacency Theorem

In this section, we prove results that will be used later in this chapter to motivate the greedy heuristic method for constructing OODGs. The first theorem states that exchanging neighboring features in an OODG only changes the nodes and incident edges at one level of the graph. A local change in feature ordering corresponds to a local change in the shape and size of the graph.

Theorem 7 (Exchanging neighboring features)
Let G be an OODG implementing a function f, and assume, without loss of generality, that the ordering of features is $X_1, \ldots, X_n$ (because we can rename the features). Let $G(i)$, $1 \leq i \leq n-1$, be the unique OODG implementing f with the ordering of the features as $X_1, \ldots, X_{i-1}, X_{i+1}, X_i, X_{i+2}, \ldots, X_n$ (i.e., the features X_i and X_{i+1} are interchanged in $G(i)$). Then the graph structure of $G(i)$ may differ from G only in the nodes at level $i+1$ and on the edges incident to the nodes at this level.

Proof: We construct a non-reduced OODG $G'(i)$ from G by replacing the nodes at level $i+1$ and all edges incident to the nodes at this level with new nodes and edges. We then reduce $G'(i)$ and because an OODG is unique for a given feature ordering (Section 6.3.1), we know we have constructed $G(i)$. The idea in the proof is that features X_i and X_{i+1} implement a mapping from the nodes at level i to the nodes at level $i+2$. The order in which we test X_i and X_{i+1} only determines the structure of the intermediate level, $i+1$, and the edges leaving and entering it.

Formally, for each node v at level i, we create $|\mathrm{Dom}(X_{i+1})|$ edges leaving it, each labelled with a different domain value, and each terminating at a new distinct node at level $i+1$. Each node w at level $i+1$ can be uniquely represented by a pair $\langle v_i, x_{i+1} \rangle$, *i.e.*, the node reached at level i and the value of feature X_{i+1}.

From each node $w = \langle v_i, x_{i+1} \rangle$ at level $i+1$, we construct $|\mathrm{Dom}(X_i)|$ edges, such that each edge labelled x_i terminates at the unique node which is reached in graph G by tracing the edges labelled x_i and x_{i+1} from node v. This defines the non-reduced OODG $G'(i)$. It is easy to see that for each node v and level i and for any values x_i and x_{i+1} for features X_i and X_{i+1} respectively, the path traced from node v reaches the same node at level $i+2$ in both graphs.

It remains to show that reducing $G'(i)$ can only collapse nodes at level $i+1$. Nodes at level $i+2, i+3, \ldots, m$ could not be collapsed in G (because G is an OODG) so clearly they cannot be collapsed in $G'(i)$ as we have not changed those levels. To prove that nodes at levels $0, 1, \ldots, i$ cannot be collapsed, it is sufficient to show that nodes at level i cannot be collapsed. Assume the contrary, *i.e.*, that two nodes at level i in $G'(i)$ can be collapsed. Then because the mapping from these nodes to the nodes at level $i+2$ is the same as in G, these nodes can be collapsed in G too without changing the function, thus contradicting the uniqueness of graph G. ∎

Corollary 8

The width of an OODG at a given level is uniquely determined by the set of features tested at that level and at lower levels.

Proof: Given an OODG built using some feature ordering, we can reorder the features at level i and lower levels by exchanging neighboring features. We can achieve any order by executing a bubble sort with comparisons done according to our desired order on the features. According to Theorem 7, this process will only affect the width of levels $i+1, i+2, \ldots, n$; hence the width of level i depends only on which features are used at level i and lower levels and not on their ordering. ∎

Given a function f, an **optimal ordering** of features for the levels of an OODG implementing f is an ordering that creates the OODG with the smallest possible size. The next theorem shows that in an optimal ordering, the width of a given level can never shrink if we replace its feature by the feature prior to it (*i.e.*, one level higher in the OODG) in an optimal ordering and keep all subsequent features the same.

Theorem 9 (Adjacency Theorem)

Define $W_X(X_{i+1}, X_{i+2}, \ldots, X_n)$ to be the width of level i when feature X labels level i and features $X_{i+1}, X_{i+2}, \ldots, X_n$ label the levels $i+1$, $i+2$, ..., n, respectively. If $X_1, \ldots, X_n$ is an optimal ordering of the features for levels 1 to n of an OODG implementing a function f, then $W_{X_i}(X_{i+1}, X_{i+2}, \ldots, X_n) \leq W_{X_{i-1}}(X_{i+1}, X_{i+2}, \ldots, X_n)$.

Proof: By Theorem 7, exchanging the features at level $i-1$ and i changes only the number of nodes at level i (and incident edges). If $W_{X_i}(X_{i+1}, X_{i+2}, \ldots, X_n)$ were larger than $W_{X_{i-1}}(X_{i+1}, X_{i+2}, \ldots, X_n)$, the exchange would decrease the width of level i without changing the width of other levels, contradicting our assumption that $X_1, \ldots, X_n$ is an optimal ordering. ∎

This theorem does *not* say that a greedy strategy that always selects the feature yielding the smallest width at every stage would generate an optimal ordering. The conditions of the lemma might hold for many other orderings that are non-optimal, and for which multiple exchanges might lead to a smaller OODG. The following example illustrates such a scenario.

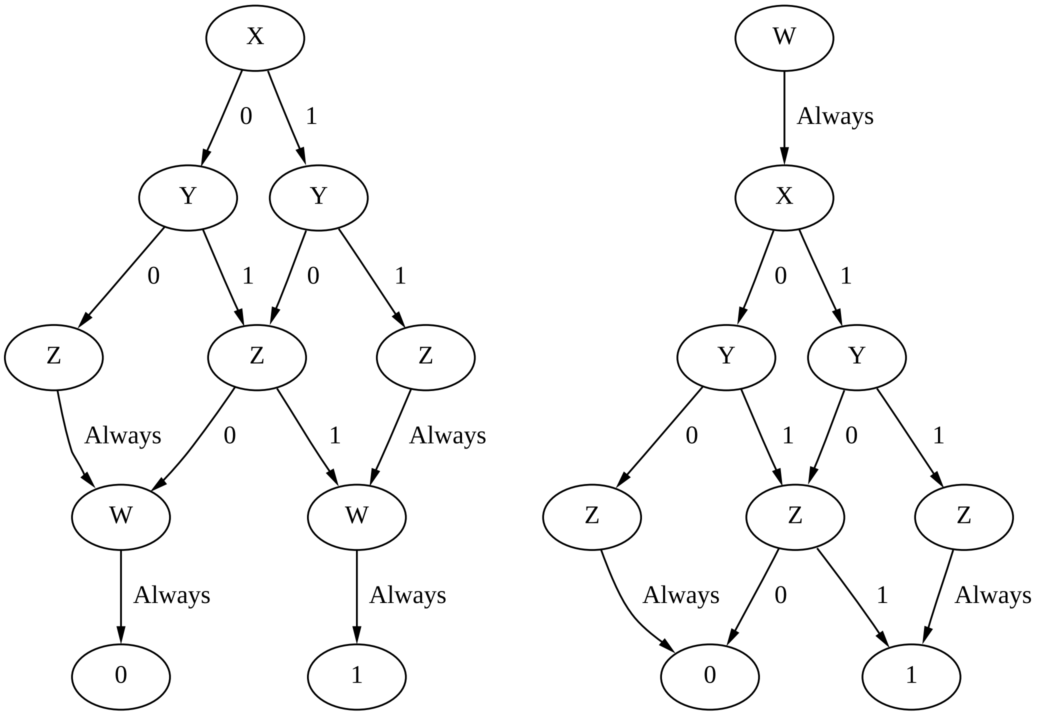

Figure 6.10: $f = (X \wedge Y) \ \vee \ (Y \wedge Z) \ \vee \ (X \wedge Z)$. A non-optimal OODG obeying the Adjacency Theorem on the left and an optimal one on the right.

Example 6.4 (Greedy is not optimal)
Assume four Boolean features W, X, Y, Z, and define f to be

$$(X \wedge Y) \ \vee \ (Y \wedge Z) \ \vee \ (X \wedge Z)$$

The OODG on the left side in Figure 6.10 obeys the Adjacency Theorem and was derived by picking the feature giving the smallest width each time. It is non-optimal, and by the Adjacency Theorem, no single exchange will improve its size; however, making W the first feature will yield a smaller OODG as shown on the right side of the figure. ∎

While the Adjacency Theorem does not guarantee an optimal ordering using a greedy approach, it does allow us to prune the search of orderings as shown in the next two corollaries. We assume that the search for an optimal ordering proceeds in a bottom-up fashion, *i.e.*, choosing the lowest feature in the OODG first and proceeding up.

Corollary 10

At any level except the first, the feature that yields the greatest width should never be chosen.

Proof: The feature above it would necessarily violate the Adjacency Theorem. ∎

Corollary 11

At any level i, the search for the feature immediately after X can be limited to features Y, such that $W_Y(X_{i+1}, X_{i+2}, \ldots, X_n) \geq W_X(X_{i+1}, X_{i+2}, \ldots, X_n)$, i.e., features that form a level wider than X at the level i.

Proof: Same as corollary 10.

Corollary 11 allows pruning of the search space of possible orderings. For example, for $n = 10$ features, finding the optimal ordering by trying all $n!$ combinations requires trying $3,628,800$ of them, while the pruned search space requires searching only $50,521$ possibilities, which is less than 1/71 of the size of the original space. In Smith & Genesereth (1985), a recurrence formula was given for a similar theorem in relation to conjunctive queries, assuming that for each level i the widths for the different features are different. Kohavi, at a talk in Mike Genesereth's group in 1993, conjectured that the number of feature orderings forms an Euler sequence (the numbers correctly match the sequence for $n = 1$ to 13). Assuming that the sequence is indeed an Euler sequence, one can show that the asymptotic saving is a factor of $2\,(2/\pi)^{(n+1)}$ for n features. While the saving is considerable for small n, the search space size is still dominated by $n!$.

The following corollary shows that a greedy selection of features that chooses the feature minimizing the next level from the bottom up achieves a local optimum that cannot be improved by any single exchange of neighboring features. Such exchanges, for example, were done to improve the size of OBDDs when it was created top-down in Fujita, Matsunaga & Kakuda (1991).

Corollary 12 (Local optimum)

If the feature that creates the smallest width at each level is chosen in a bottom-up construction of an OODG, no exchange of two adjacent features will improve the size of the OODG.

Proof: By Theorem 7, swapping the features for levels $i-1$ and i, changes only the width of level i. Since each feature is selected to minimize the width, any neighbor must create a larger level, increasing the overall size. ▮

6.4 A Framework for Bottom-up Construction of OODGs

In this section, we present a recursive algorithm for constructing OODGs. To make the exposition simple, we make the following simplifying assumptions that will be handled separately:

1. We assume that the complete labelled instance space, $\mathcal{X} \times \mathcal{Y}$, is given as the training set. This assumption avoids the induction problem altogether. We simply have to find an OODG that perfectly fits the data.

2. We assume that the feature ordering is $X_1, X_2, \ldots, X_n$. Because we know that the feature ordering is crucial for finding a small OODG (Section 6.3.1), this assumption delays a hard problem for later sections.

3. We assume that all input features are Boolean but do not restrict the label to be Boolean. Continuous values are discretized in advance, and the extension from Boolean inputs to discrete inputs is simple. This assumption is made merely to avoid cumbersome notation in a few places. The fact that the label is non-Boolean is needed because the algorithm is recursive and builds subproblems that have more than two label values, even if the original problem has a Boolean label.

The input to the algorithm is a set of sets, $\{C_0, C_1, \ldots, C_{k-1}\}$, where each set C_i is the set of *all* unlabelled instances in the space that should be mapped to class i. The output of the algorithm is an OODG that correctly classifies the training set.

The algorithm, shown in Figure 6.11, creates sets of instances, such that each set corresponds to one node in the graph (the input sets corresponding to the category nodes). Intuitively, we would like an instance in a set C_i to reach node V_i (corresponding to the set), when the instance's path is traced from the root of the completed OODG, branching at branching nodes according to the feature values.

Given the input, the algorithm selects a feature X to test at the penultimate level of the OODG. As assumed, we currently ignore the feature ordering problem and always

Input: k mutually exclusive and exhaustive sets $C_0, \ldots, C_{k-1}$ such that $\mathcal{X} = \bigcup_{i=0}^{k-1} C_i$ (the whole instance space). Each set contains all the instances that should be mapped to the given class.

Output: The OODG that correctly classifies all instances in $\mathcal{X}$ with feature ordering $X_1, X_2, \ldots, X_n$.

1. If $(k = 1)$, then return a graph with one node.
2. Feature selection step: Let X be X_n, the last feature. (This step will be replaced in variants of the algorithm.)
3. Project the instances in $C_0, \ldots, C_{k-1}$ onto the instance space $\mathcal{X}'$, such that feature X is deleted. Formally, if X is the ith feature,
$$\mathcal{X}' \leftarrow \pi_{(X_1,\ldots,X_{i-1},X_{i+1},\ldots,X_n)} \bigcup_{i=0}^{k-1} C_i \quad \text{(where } \pi_{(\vec{X})} \text{ means project on } \vec{X}\text{)} .$$
4. For all $i, j \in \{0, \ldots, k-1\}$, let C'_{ij} be the set containing instances from $\mathcal{X}'$ such that the instances are in set C_i when augmented with $X = 0$, and in C_j when augmented with $X = 1$. Formally,
$$C'_{ij} = \left\{ \langle X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n \rangle \;\middle|\; \begin{array}{l} \langle X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n \rangle \in C_i \text{ and} \\ \langle X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n \rangle \in C_j \end{array} \right\} .$$
5. Let k' be the number of non-empty sets from $\left\{C'_{ij}\right\}$. Call the algorithm recursively with the k' non-empty sets in the space $\mathcal{X}'$, and let G be the OODG returned.
6. Label the k' leaf nodes of G, corresponding to the non-empty sets C'_{ij} with feature X. Create a new level with k nodes corresponding to the sets $C_0, \ldots, C_{k-1}$. From the node corresponding to each C'_{ij}, create two edges: one labelled 0, terminating at the category node corresponding to C_i, and the other labelled 1, terminating at the category node corresponding to C_j.
7. Return G.

Figure 6.11: The basic bottom-up algorithm for constructing an OODG.

select the last feature in the given space (the space shrinks in each recursive call, thus we effectively select X_n then X_{n-1}, X_{n-2}, *etc.*). The algorithm then creates new sets of instances (corresponding to the nodes in the penultimate level of the final OODG), which are projections of the original instances with feature X deleted. The sets are created so that a set C'_{xy} (matching a branching node V'_{xy}) contains all projections of instances that are in C_x when augmented with $X = 0$, and in C_y when augmented with $X = 1$. In the graph, the branching node corresponding to C'_{xy} will have the edge labelled 0 terminating at node V_x, and the edge labelled 1 terminating at node V_y.

The new sets now form a smaller problem over $n - 1$ features, and the algorithm calls itself recursively to compute the rest of the OODG with the nonempty sets of the new level serving as the input. The recursion stops when the input to the algorithm is a single set, possibly consisting of the *null* instance (0 features). Before proving correctness of the algorithm, we show an example run.

Example 6.5 (Executing the Algorithm on Parity)

In this example, we show how to run the algorithm for the 3-bit odd parity function with one irrelevant feature, *i.e.*, $f = X_1 \oplus X_2 \oplus X_4$ (X_3 is irrelevant). Figure 6.12 parallels the description below and depicts how the OODG is built.

The input to the algorithm is $\{C_0, C_1\}$. All instances in C_0 have a label 0, and all elements in C_1 have label 1:

$$\begin{aligned} C_0 &= \{0000, 0010, 0101, 0111, 1001, 1011, 1100, 1110\} \\ C_1 &= \{0001, 0011, 0100, 0110, 1000, 1010, 1101, 1111\} \ . \end{aligned}$$

Deleting feature X_4 from each instance gives us the following projected instance space:

$$\mathcal{X}' = \{000, 001, 010, 011, 100, 101, 110, 111\} \ .$$

Because we started with the full instance space, each of these projections has a defined **destination** (a set name shown after the right-arrow below) for each possible value of X_4. Creating sets from all projected instances in $\mathcal{X}'$ that have the same destinations for the same values of X_4, we get the following sets:

$$\begin{aligned} C_{01}(0 \to C_0, 1 \to C_1) &= \{000, 001, 110, 111\} \\ C_{10}(0 \to C_1, 1 \to C_0) &= \{010, 011, 100, 101\} \ . \end{aligned}$$

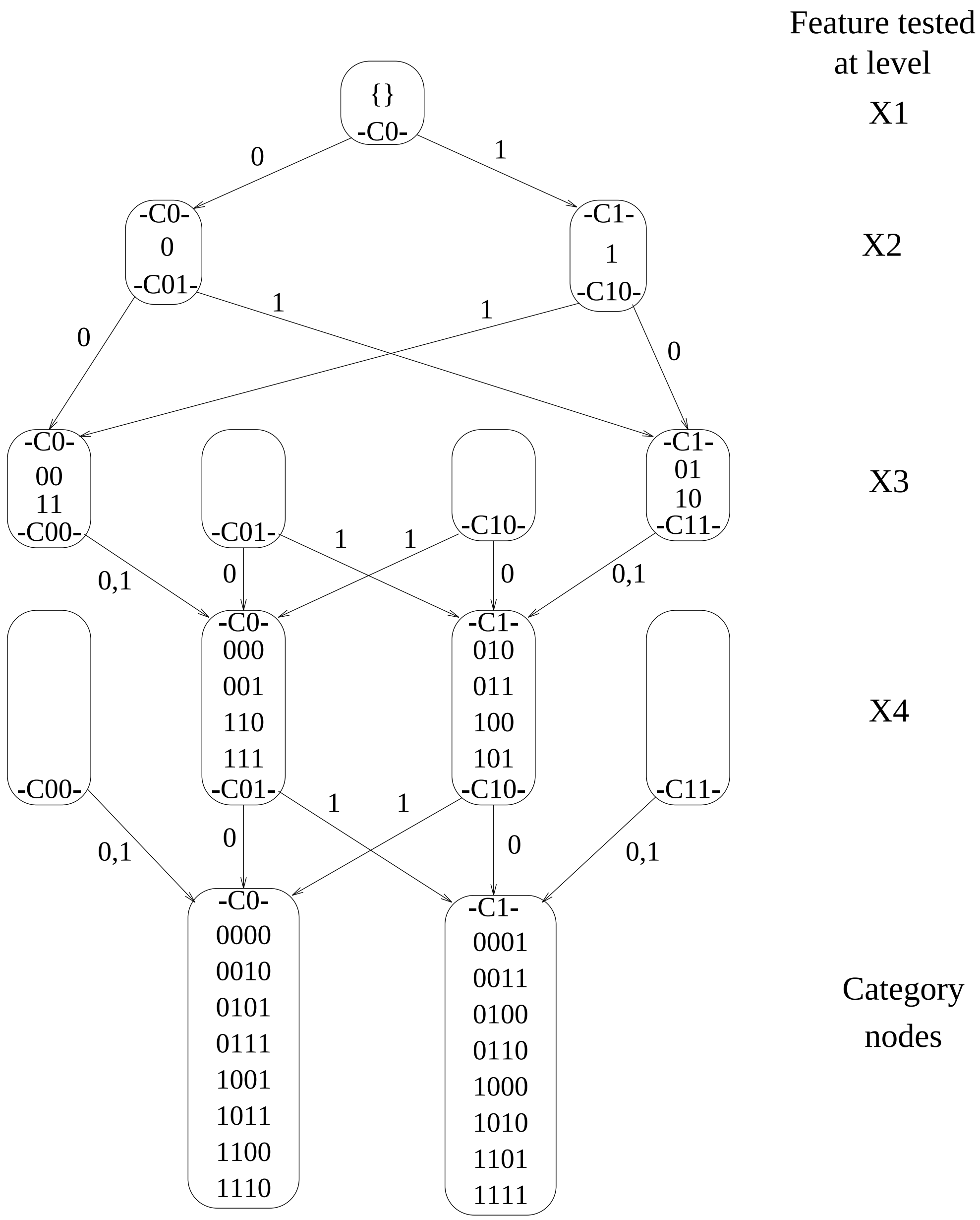

Figure 6.12: Example run of the bottom-up construction algorithm. The nodes correspond to the formed sets. The set name is represented at the bottom of each node (*e.g.*, C_{10} branches to node C_1 at the next level on value zero and to node C_0 on value one), and the name used for the recursive call is at the top of each node. The target concept is $f = X_1 \oplus X_2 \oplus X_4$.

Note that out of four possible sets, only two were needed. We now construct the OODG recursively using the two non-empty sets C_{01} and C_{10} as our input sets (the sets will be C_0 and C_1 in the recursive call). Selecting feature X_3 to delete gives us the following projected instance space:

$$\mathcal{X}'' = \{00, 01, 10, 11\} \ .$$

Creating the appropriate sets from the projected instances in $\mathcal{X}''$ yields the following sets:

$$\begin{aligned} C_{00}(0 \to C_{01}, 1 \to C_{01}) &= \{00, 11\} \\ C_{11}(0 \to C_{10}, 1 \to C_{10}) &= \{01, 10\} \ . \end{aligned}$$

Note that each of the two new sets implements a constant function that ignores the value of the given feature (*i.e.*, it branches to the same node regardless of the feature value). A level for which all nodes are constant nodes implies that the feature is irrelevant, and indeed, $X3$ is irrelevant for the target concept. Continuing the execution yields the OODG depicted in Figure 6.13. ▮

To extend the algorithm to discrete features, the projection step must create nodes that have ℓ edges for a tested feature with ℓ discrete values. Instead of creating sets of the form C_{ij}, we would have sets of the form $C_{i_1,i_2,\ldots,i_\ell}$. We now prove correctness of the algorithm for general discrete features and any ordering of features, *i.e.*, the selection step which currently selects the last feature may be replaced with a function that selects any one of the features in the projected instance space.

Theorem 13 (Correctness of the bottom-up algorithm)
The algorithm terminates and generates an OODG that correctly classifies all instances in the space $\mathcal{X}$.

Proof: We prove termination, non-redundancy of nodes (reduced decision graph), and correctness.

Termination After at most n recursive calls, the algorithm must terminate because all features will have been used, and the projected instance space will be the space of no features, containing only the "empty" instance in set C_0 (so $k = 1$).

Reduced OODG The sets C_{ij} (or $C_{i_1,i_2,\ldots,i_\ell}$ in general) are by definition different in their destinations, so no two nodes can be collapsed.

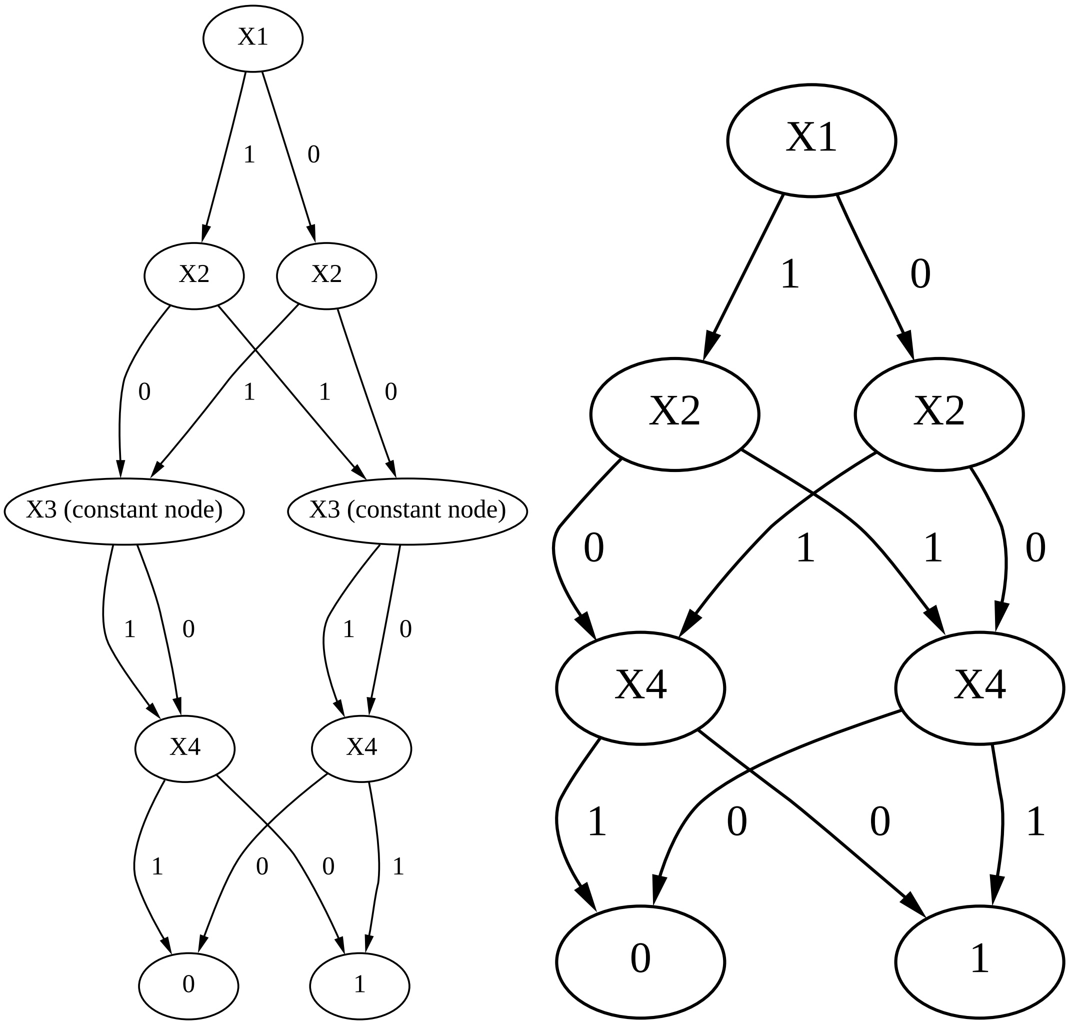

Figure 6.13: The OODG constructed for the function $X_1 \oplus X_2 \oplus X_4$ (left), and the same OODG after removal of constant nodes (right).

Correctness We prove that the algorithm generates a structure to classify the inputs correctly by induction on the number of levels. If there is only one level, there is just one category, and all instances belong to it. If the first $n-1$ levels correctly categorize the input into the categories required by the first recursive call, then by construction, nodes will map the instances to the right category node. ∎

6.5 Hardness Results and the HOODG Algorithm

I never came across one of Laplace's "thus it plainly appears" without feeling sure that I have hours of hard work before me to fill up the chasm and find out and show how it plainly appears
—Nathaniel Bowditch, American Astronomer and Mathematician who translated Laplace's Traite de Mecanique Céleste

In this section, we show that constructing the smallest possible OODG from a partial instance space in polynomial time is impossible if P≠NP. Even the single projection step of the bottom-up construction algorithm (Step 3 in Figure 6.11) requires solving an NP-hard problem when only a portion of the instance space is available. We then define a greedy heuristic for the projection step and a greedy heuristic for feature ordering. We conclude the section with experimental results and open problems.

6.5.1 Incomplete Projections

If the full instance space is not given, there will be projections of instances for which some values of the deleted feature will be missing (*e.g.*, a projected instance must branch to some node on value 0 but the destinations for value 1 is unknown). Such projections are called **Incomplete Projections,** or **IPs.** Assigning values to the missing destinations of IPs defines the generalizations made by the algorithm because it determines how unseen instances will be classified.

An IP is **consistent** with another projection, P, (at the same level of the graph), if they do not have conflicting destinations on the same value of the deleted feature. For example, IP $\{0,0\}$ with destination C_0 on value zero is consistent with projection $\{1,1\}$ with destination C_0 on value zero and C_1 on value one; it is inconsistent with $\{1,0\}$ with destination C_1 on value zero. An IP is **included** in another projection, P, if they are consistent, and if all destinations defined for the IP are also defined for the projection P. (Note that *included* is an asymmetric relation.)

Following Occam's razor, we would like to find the smallest OODG consistent with the data (we ignore pruning, or regularizing, at this stage). We are thus looking for a minimal set of branching nodes that "covers" all projections, *i.e.*, a minimal cover.

The following results show that it is unlikely that an algorithm finding the smallest consistent OODG will be found, even for a given ordering. In Takenaga & Yajima (1993) and in Bollig & Wegener (1994), it was shown that identifying whether there exists an OBDD with k nodes that is consistent with labelled instances is NP-complete, and this result applies to OODGs too. The following theorem shows that minimizing even a single level in an OBDD or OODG is NP-complete:

Theorem 14 (Minimal projection is NP-complete)
The following decision problem is NP-complete:
Given a set of labelled instances, an ordering on the n features, and two positive integers w and ℓ; is there an OODG that has width $\leq w$ at level ℓ and that correctly classifies all instances?

The proof is given in Appendix C.

The reduction in the proof is done from graph k-colorability (chromatic number) using only Boolean features. This is a strong negative result because it is known that the chromatic number of a graph cannot be approximated to within any constant multiplicative factor unless P=NP (Lund & Yannakakis 1993).

Finding the minimal representation using other common structures is also difficult in the worst case, yet heuristics seem to work well in practice. It is known that finding an optimal binary decision tree is NP-hard (Hyafil & Rivest 1976, Hancock 1989). For neural-networks, the problem of loading a three-node neural network with a training set is NP-hard if the nodes compute linear threshold functions (Judd 1988, Blum & Rivest 1992).

Given the hardness results, we use a simple greedy heuristic to assign the IPs. The greedy strategy starts creating projection sets (branching nodes) from projections having the greatest number of known destinations, and then proceeds to projections with fewer known destinations. Following a least commitment strategy, each projection is placed in a projection set it is *included* in (see definition above), whenever possible (hence not forcing a new destination); otherwise, it is placed in a set where it is consistent with all instances, if possible; otherwise, a new projection set is created, consisting of the single projection.

Our heuristic breaks ties in favor of a projection set (a node containing instances) that has the most instances differing by at most one feature value, and given equality, breaks ties in favor of adding the minimum number of new destinations (again, least commitment).

6.5.2 Feature Ordering

There are $n!$ possible orders in which to select the features in Step 2 of the algorithm in Figure 6.11. Given the full instance space, it is possible to find the optimal ordering using dynamic programming by checking "only" 2^n orderings, as described in Friedman & Suppowit (1987) and Friedman & Suppowit (1990).

Since searching $n!$ combinations is impractical in practice, our greedy approach selects the feature that yields the smallest width at the next level, excluding constant nodes. Based on Corollary 12, we know that the greedy strategy achieves a local optimum in the space of possible orderings if the whole feature space was given. The approach seems to work well in practice, unless there are ties. We break ties in favor of minimizing the number of edges. If all nodes are constant nodes (the feature is deemed irrelevant), we do another lookahead step and pick the feature that maximizes the number of irrelevant features at the next level.

After a feature with k values, the size of the space is shrunk by a factor of k. The size of the training set for the recursive call, however, is likely to shrink by less than a factor of k, unless for every projected instance there are exactly k instances in the training set, each one with a different value for the deleted feature. The greedy approach tends to prefer irrelevant features early because they never increase the width of the OODG. If irrelevant features are indeed chosen early, the ratio of the number of projected instances to the projected instance space increases, so less generalization is required.

6.5.3 The HOODG Algorithm

The HOODG algorithm implements the two greedy, or hill-climbing, strategies described above. The worst-case time complexity of the HOODG algorithm is $O(ns^2 + is^2(n-1))$ per level, where i is the number of features at the given level that require another step of lookahead, and s is the number of projected instances at that level. This assumes that the number of values per feature is a bounded constant. If we ignore the two-level lookahead, the time complexity of the overall algorithm is $O(n^2m^2)$, where m is the number of instances in the training set. The result follows from the fact that the number of levels is bounded by n, and the number of projections at each level is bounded by m.

Table 6.1: Comparison of C4.5, DGRAPH, and HOODG. Results are averages of ten runs with standard deviation after the ± sign. Number in parentheses denote the training set size and test set size; XV means ten-fold cross-validation. "local" means local encoding, "binary" means binary encoding. The best accuracy for each dataset is shown with a star (★), and accuracies within one half standard deviation of the best, are marked with a check-mark (✓). Such small differences in accuracy indicate comparable performance.

Data Set		C4.5	DGRAPH	HOODG
Monk1	(60/432)	83.56% ± 9.27%	73.89% ± 2.68%	✓★ 100.00% ± 0.00%
Monk2	(169/432)	72.73% ± 5.66%	66.67% ± 1.46%	✓★ 91.30% ± 1.98%
Monk2-local		75.75% ± 7.83%	67.13% ± 0.00%	✓★ 99.14% ± 0.62%
parity5+5	(100/1024)	51.56% ± 2.32%	50.00% ± 0.00%	✓★ 100.00% ± 0.00%
vote	(435/XV)	✓ 95.43% ± 4.31%	✓★ 95.63% ± 3.69%	✓ 94.03% ± 3.46%
breast cancer	(699/XV)	✓★ 94.64% ± 6.16%	✓ 93.85% ± 4.26%	86.99% ± 6.50%
breast cancer	binary	✓★ 96.07% ± 6.16%	92.84% ± 5.94%	✓ 95.03% ± 2.77%

Table 6.1 shows a comparison of the HOODG algorithm with C4.5 and DGRAPH. DGRAPH (Oliver 1993) is an algorithm for building general decision graphs (with no restrictions), using an MDL criterion for splitting and merging nodes. The comparison was originally shown in Kohavi (1994*b*) and is very limited because the HOODG algorithm is neither able to deal with noise nor is the feature ordering very good as we will shortly discuss. The two datasets not previously described in this dissertation are vote and parity5+5. The vote database includes votes for each of the U.S. House of Representatives Congressmen on 16 key votes; the task is to classify each congressman as either a Democrat or a Republican. Parity5+5 is a dataset with ten features, and the target concept is the parity of five of them (the rest are irrelevant).

In the comparison shown, C4.5 was run with the best setting of its m parameter to either one or two and the g parameter to either on or off. DGRAPH was executed with two levels of lookahead and with different p values (the one that yields the minimum message length is reported, as suggested by Oliver).

While HOODG was successful on the artificial concepts, it was not very good at handling real-world domains, and suffered when there were many weakly relevant and irrelevant features. We now list the main limitations of the algorithm:

Feature ordering The greedy approach works well if there are not many ties for the heuristic that attempts to minimize the width of the penultimate level. However, if there are many features, many of them will lead to ties because the set of $n-1$ features that are left after one feature is deleted uniquely determines the class in the training set (this is usually the case if there are enough features). If the class is uniquely determined, the width of the penultimate level is the same as the number of classes, and the nodes implement constant functions ignoring the feature value. The heuristic measure for feature selection is thus unable to determine which of the features it should project on. The extra lookahead rarely helps if there are many features because the next level will usually have similar ties.

Pruning HOODG does no pruning and, therefore, perfectly fits the data. It is well known that pruning, or regularizing, introduces bias and reduces variance, so that the error may decrease for real problems.

Real-valued features The algorithm is defined only for discrete features. This restriction is not severe, as the data can be discretized.

6.6 HOODG with Wrappers

Given the many limitations of the original HOODG algorithm, we now investigate three approaches to feature ordering and feature subset selection. The first method uses the wrapper approach; the second one uses an entropy criterion; and the third one combines both. Selecting a feature subset and an ordering indirectly provides a pruning mechanism because the instances projected on a small set of features might contain conflicts (see related discussion in Section 4.4.1 where feature subset selection provided an indirect pruning mechanism for ID3).

The algorithms that will be discussed below discretize continuous features and two of them contain wrappers around the basic HOODG algorithm. The wrapper searches the space of feature orderings (not all features need to be used) and then runs HOODG on the given ordering where only the covering heuristic for incomplete projections (IPs) is used. The following three operators are used to search the states that represent feature lists,

Add feature A feature is added to the end of the feature list. This operator is applied for every feature not currently in the feature list.

Table 6.2: Summary of datasets with unknown instances removed. The number in parentheses indicate the number of instances including unknowns. CV indicates ten-fold cross-validation.

no.	Dataset	Features			no.	Train	Test
		all	nominal	cont	classes	size	size
1	breast cancer	10	0	10	2	683 (699)	CV
2	cleve	13	7	6	2	296 (303)	CV
3	crx	15	9	6	2	653 (690)	CV
4	DNA	180	180	0	3	2000 (2000)	1186 (1186)
5	horse-colic	22	15	7	2	7 (368)	CV
6	Pima	8	0	8	2	768 (768)	CV
7	sick-euthyroid	25	18	7	2	1351 (2108)	649 (1055)
8	soybean-large	35	35	0	15	562 (683)	CV
9	corral	6	6	0	2	32 (32)	128 (128)
10	m-of-n-3-7-10	10	10	0	2	300 (300)	1024 (1024)
11	Monk1	6	6	0	2	124 (124)	432 (432)
12	Monk2-local	17	17	0	2	169 (169)	432 (432)
13	Monk2	6	6	0	2	169 (169)	432 (432)
14	Monk3	6	6	0	2	122 (122)	432 (432)

Delete feature A feature is deleted from the feature list. This operator is applied for every feature that is currently in the feature list.

Exchange feature A consecutive pair of features in the list is exchanged. This operator is applied for all pairs of two consecutive features in the list (there are $\ell - 1$ for a list of size ℓ).

Although the add feature alone can reach any node in the space theoretically, the other two operators are useful in practice because we rarely do an exhaustive search. Specifically, the "exchange feature" operator allows the ordering to change without changing the feature subset: any ordering of a feature subset can be achieved by exchanging neighboring features (bubble sort).

We describe experiments on the standard set of datasets used in this dissertation. The implemented HOODG algorithm and variants do not support unknown values, so all instances with unknown values were removed from the datasets. The mechanism for handling

Table 6.3: A comparison of C4.5 and HOODG-Frwd. The p-val column indicates the probability that HOODG-Frwd is better. All instances with unknown values were removed.

	Dataset	C4.5	HOODG-Frwd	p-val
1	breast cancer	95.32± 0.9	95.75± 0.7	0.70
2	cleve	74.98± 2.5	81.77± 2.7	1.00
3	crx	84.08± 1.1	85.15± 1.0	0.85
4	DNA	92.66± 0.8	93.93± 0.7	0.96
5	horse-colic	100.00± 0.0	100.00± 0.0	0.50
6	Pima	71.60± 1.9	74.34± 1.7	0.93
7	sick-euthyroid	97.84± 0.6	97.07± 0.7	0.11
8	soybean-large	92.54± 1.4	91.46± 1.3	0.22
9	corral	81.25± 3.5	81.25± 3.5	0.50
10	m-of-n-3-7-10	85.55± 1.1	77.34± 1.3	0.00
11	Monk1	75.69± 2.1	100.00± 0.0	1.00
12	Monk2-local	70.37± 2.2	64.35± 2.3	0.00
13	Monk2	65.05± 2.3	64.35± 2.3	0.38
14	Monk3	97.22± 0.8	97.22± 0.8	0.50
	Average real:	88.63	89.93	
	Average artif.	79.19	80.75	

unknown values in OODGs can be done in a similar fashion to the way C4.5 handles unknown values. Table 6.2 shows a summary of the datasets after unknown values have been removed. The horse-colic dataset becomes rather useless, as there are only seven instances left, and they are all of the same class. For the sick-euthyroid dataset, the TBG feature was removed prior to filtering instances with unknown values because it contained an unknown value for almost all the instances. In all runs comparing algorithms, the exact same datasets and folds were used (all had unknown values removed).

6.6.1 HOODG-Frwd

The HOODG-Frwd algorithm runs a wrapper starting from the empty list of features. Table 6.3 and Figure 6.14 show a comparison of C4.5 and HOODG-Frwd. The following observations can be made:

1. For the real domains, the results indicate that HOODG-Frwd is significantly better for cleve, DNA, and Pima (at the 90% confidence level), and slightly better on breast

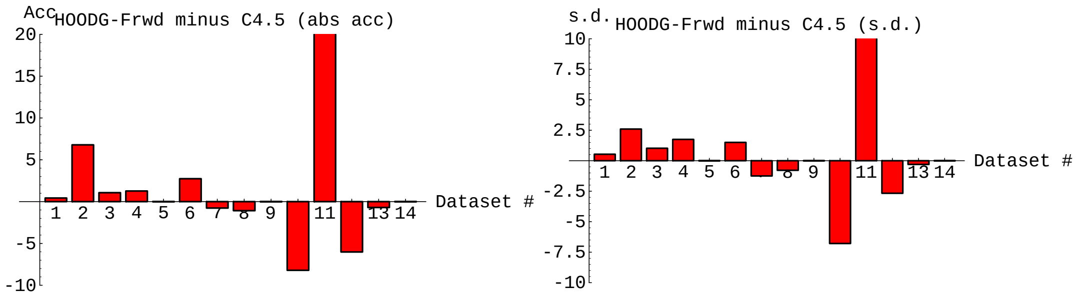

Figure 6.14: HOODG-Frwd minus C4.5: Absolute difference in accuracy (left) and in std-devs (right). The std-devs graph truncated at 5 standard deviations.

cancer and crx. C4.5 is better on sick-euthyroid and soybean-large, but these are not significant at the 90% level. The relative reduction in error is 11.4%.

2. For the artificial domains, HOODG-Frwd is significantly better only for Monk1. It is significantly worse for m-of-n-3-7-10 and Monk2-local, which is disappointing, given that these are two symmetric concepts. The best-first search is not searching the space well enough to find the correct set of features.

The problem of identifying the correct set of features in the artificial problems can be mitigated either by increasing the stale parameter for the wrapper (*i.e.*, the stopping criterion) or by starting the search at a better initial state in the state space. The next section attempts to find a good initial state.

6.6.2 Using Conditional Entropy

One possibility for improving HOODG-Frwd, both in terms of the running times and in terms of finding a good set of features, is to start the search for the feature subset and ordering with a better initial node. A good initial node is algorithm specific, but it is not hard to design a reasonable choice for OODGs. The idea is similar to that of decision trees, namely to minimize the conditional entropy of the label given the tested features. The advantage of OODGs is that any feature subset uniquely defines the conditional entropy (whereas with trees, one has to define all the paths to the leaves).

A greedy algorithm beginning with an empty list of features, finds the best single feature

Table 6.4: A comparison of C4.5, HOODG-Entropy, and HOODG-Middle. Each of the p-val columns indicates the probability that the algorithm in the previous column is better than C4.5.

	Dataset	C4.5	HOODG-Entropy	p-val	HOODG-Middle	p-val
1	breast cancer	95.32± 0.9	95.31± 0.7	0.50	95.32± 0.6	0.50
2	cleve	74.98± 2.5	81.08± 2.5	0.99	82.80± 2.0	1.00
3	crx	84.08± 1.1	86.23± 0.8	0.99	85.31± 1.0	0.88
4	DNA	92.66± 0.8	94.18± 0.7	0.98	94.10± 0.7	0.98
5	horse-colic	100.00± 0.0	100.00± 0.0	0.50	100.00± 0.0	0.50
6	Pima	71.60± 1.9	73.56± 1.4	0.88	74.47± 1.8	0.94
7	sick-euthyroid	97.84± 0.6	96.92± 0.7	0.07	97.07± 0.7	0.11
8	soybean-large	92.54± 1.4	83.63± 2.0	0.00	91.47± 1.1	0.20
9	corral	81.25± 3.5	75.00± 3.8	0.04	81.25± 3.5	0.50
10	m-of-n-3-7-10	85.55± 1.1	100.00± 0.0	1.00	100.00± 0.0	1.00
11	Monk1	75.69± 2.1	100.00± 0.0	1.00	100.00± 0.0	1.00
12	Monk2-local	70.37± 2.2	100.00± 0.0	1.00	100.00± 0.0	1.00
13	Monk2	65.05± 2.3	63.43± 2.3	0.24	64.35± 2.3	0.38
14	Monk3	97.22± 0.8	97.22± 0.8	0.50	97.22± 0.8	0.50
	Average real:	88.63	88.86		90.07	
	Average artif.	79.19	89.27		90.47	

that minimizes the conditional entropy (of the label given the list of features), appends it to the list of features, and reiterates. As with decision trees, we find a longer list than necessary and then prune the trailing features using cross-validation on the oblivious decision-tree formed. If the list is of size n, then we conduct n cross-validation runs and choose the best one. Kohavi & Li (1995) used this idea to build an oblivious decision tree and to convert it to a graph.

The HOODG-Entropy algorithm determines the feature ordering based on the above heuristic. It does not use the wrapper approach. The HOODG-Middle algorithm starts the search at the node found using the entropy heuristic (the wrapper search progresses neither forward nor backward because it starts in the middle of the search space).

Table 6.4 and Figure 6.15 show a comparison of C4.5, the HOODG-Entropy algorithm, and the HOODG-Middle algorithm. The following observations can be made:

1. HOODG-Entropy significantly outperforms C4.5 on three real-world datasets (cleve,

crx, and DNA) and is significantly inferior on two (sick-euthyroid and soybean-large). On the artificial datasets, it is significantly better on m-of-n-3-7-10, Monk1, and Monk2-local where the optimal graphs are found, while inferior only on corral.

Thus on most datasets, the entropy heuristic seems very good.

2. HOODG-Middle, which uses the wrapper approach to search from the initial node found by HOODG-Entropy, is slightly outperforming HOODG-Entropy. The main improvement is for the soybean-large dataset, where the accuracy went up from 83.63% to 91.47%.

3. For the real datasets, the performance of the HOODG-Middle algorithm is similar to that of the HOODG-Frwd. It is slightly higher for some datasets and slightly lower for others, but the overall average accuracy is slightly higher than that of HOODG-Frwd. The relative reduction in error between HOODG-Middle and C4.5 for the artificial datasets was 10.9%.

4. For the artificial datasets, HOODG-Middle improves over HOODG-Entropy and generates a classifier that has the same accuracy as C4.5. The relative reduction in error between HOODG-Middle and C4.5 for the artificial datasets was 54.2%.

The running times for a train-test sequence of HOODG-Middle are very high[6]: 50 hours for DNA, 15 hours for soybean-large, 1.7 hours for cleve, 1.5 hours for Pima, 19 minutes for crx, 15 minutes for sick-euthyroid, 13 minutes for corral, mofn, and Monk2-local, and less than five minutes for breast-cancer, horse-colic, Monk1, Monk2, and Monk3. Decreasing the stale parameter from five to two will speed all runs significantly, with little loss of accuracy.

The running times for HOODG-Entropy are much faster: 47 minutes for DNA, and under three minutes for all the other datasets. One can use HOODG-Middle as an anytime algorithm (Boddy & Dean 1989) and let it run as long as time permits. The initial guess is quite good, but may improve as in the soybean-large dataset.

[6]Speed is always relative. A recent posting by Radford Neal announced software for neural network Bayesian learning, where he writes that "For problems and networks of moderate size (*e.g.*, 200 training cases, 10 inputs, 20 hidden units), full training (to the point where one can be reasonably sure that the correct Bayesian answer has been found) typically takes several hours to a day on our SGI machine.... (Of course, your machine may not be as fast as ours!)" If the scaling up of neural networks were linear in the size of the dataset and the number of features (it is far from that), the DNA dataset, which has 18 times more features and ten times the number of instances, would take more than 180 days.

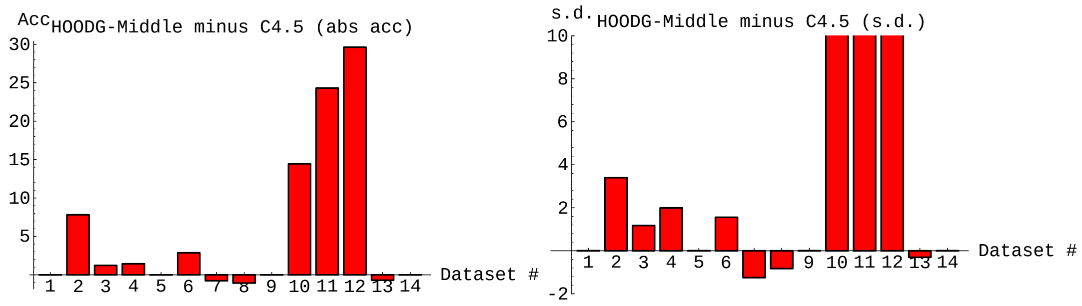

Figure 6.15: HOODG-Middle minus C4.5: Absolute difference in accuracy (left) and in std-devs (right). The std-devs graph is truncated at ten standard deviations.

Table 6.5: The number of features in the dataset, the number used by C4.5, and the number selected by HOODG-Middle.

	Dataset	Original dataset	C4.5	HOODG-Middle
1	breast cancer	10	6.2	3.6
2	cleve	13	8.7	3.1
3	crx	15	9.0	3.2
4	DNA	180	46	11
5	horse-colic	22	0.0	0.0
6	Pima	8	8.0	4.0
7	sick-euthyroid	25	6	3
8	soybean-large	35	16.9	9.7
9	corral	6	4	2
10	m-of-n-3-7-10	10	9	8
11	Monk1	6	5	3
12	Monk2-local	17	12	7
13	Monk2	6	6	3
14	Monk3	6	2	2

Table 6.5 shows the average number of features in the dataset, the number used by C4.5 and in the number used by HOODG-Middle. One can see that HOODG-Middle uses equal or fewer features in all the datasets. A comparison with C4.5-FSS (Table 4.12 on page 114) shows that HOODG-Middle uses equal or fewer features in almost all datasets, with the exception of m-of-n-3-7-10, Monk2-local, and Monk2. (The comparison is not completely fair because instances with unknown values were removed from the datasets in this chapter, which affects all the real datasets except DNA and Pima.) The larger feature subset in m-of-n-3-7-10 and Monk2-local is justified because HOODG-Middle has higher accuracy and finds an optimal feature subset.

The comprehensibility of a classifier's structure (*e.g.*, a decision tree or graph) is a subjective judgment. Appendix D shows the trees and the OODGs generated by C4.5 and HOODG-Middle. To avoid showing ten trees for the cross-validation runs, we used a 1/3 holdout (the training set was two-thirds of the dataset, the test set was one third of the dataset) to generate the trees and graphs.

Not all graphs are more comprehensible than trees, even when the accuracy is comparable or higher. For example, the OODG for DNA seems very complicated. Some of the differences are due to the discretization more than to differences in the algorithm. For example, the discretization algorithm discretized the "FTI" feature in the sick-euthyroid dataset into one interval (essentially removing it), yet C4.5 used it heavily. If we run C4.5 on the discretized dataset, then the ten-fold accuracy degrades to 96.76%, which is lower than the accuracy of the both variants of the basic HOODG algorithm; of course, in some cases, the discretization might help C4.5.

6.7 Related Work

We are like dwarfs on the shoulders of giants, so that we can see more than they, and things at a greater distance, not by virtue of any sharpness of sight on our part, or any physical distinction, but because we are carried high and raised up by their giant size.
—Bernard of Chartres, John of Salisbury Metalogicon, 1159, Book 3

Decision graphs have been rediscovered many times, and their properties were studied under different frameworks. In this section, we give references to the most important papers and survey papers. Very little work, however, has dealt with learning decision graphs; therefore, we will cover the relevant papers in more detail.

6.7.1 Variants of Decision Graphs

Lee (1959) introduced *binary decision programs* that are evaluated by executing a series of instructions that test a variable (feature) and make a two way branch. He showed that it is possible to represent any switching function in $O(2^n/n)$ such instructions.

Akers (1978) described binary decision diagrams and gave a top-down procedure for building them using the Boole-Shannon expansion (Boole 1854, Shannon 1949):

$$f = x_i \cdot f|_{x_i=1} + \overline{x_i} \cdot f|_{x_i=0}$$

where $f|_{x_i=b}$ is the restriction, or cofactor, of the function f

$$f|_{x_i=b}(x_1, \ldots, x_n) = f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$$

Moret (1982) gives an excellent survey of work on decision trees and diagrams with over 100 references.

Bryant (1986) introduced Ordered Binary Decision Diagrams (OBDDs), which spawned a plethora of articles and a whole subcommunity dealing with OBDDs; Bryant (1992) surveys the topic. OBDDs are a restriction of Binary Decision Diagrams (BDDs), where a total ordering is defined over the set of features and all paths must test features in accordance with the given ordering. Note that OBDDs are *not* necessarily levelled. Bryant describes the advantages of OBDDs over the common representations like CNF and DNF (these advantages apply to OODGs too):

- Operations like complementation may yield exponential growth for DNF and CNF, while they do not change the size of OBDDs.
- Common operations, such as reduction, $f_1 < \text{op} > f_2$ (where op is any binary function), restriction, and composition are bounded by the product of the graph sizes for the functions being operated on.
- Satisfiability testing takes constant time (check if the OBDD has a single category node $\mathbf{0}$), while finding a satisfying assignment for n features takes $O(n)$. Counting the number of satisfying assignments if $O(|G|)$ where $|G|$ is the size of the graph, and finding all satisfying assignments is $O(n \cdot |S_f|)$ where $|S_f|$ is the number of such satisfying assignments.

OBDDs have been used for automatically verifying finite state machines, including 64-bit ALUs, with up to 10^{120} states by representing the state space symbolically instead of explicitly (Burch, Clarke, McMillan, Dill & Hwang 1990, Burch, Clarke & Long 1991). These applications show, at least empirically, that many functions occurring in engineering domains seem to be representable in small (polynomial) OBDD structures (and hence in OODGs).

Chakravarty (1993) characterizes BDDs in terms of the complexity of computational problems. He showed that free BDDs, which are read-once decision graphs and hence a superset of OBDDs, can also be used to solve the common problems in the BDD community (*e.g.*, test generation, simulation, and compact testing) in polynomial time. He also showed that the cover problem, *i.e.*, determining whether one function implies the other, is NP-hard even for simple BDDs, in which every input feature labels at most one node (equivalent to μ-branching programs described below).

In the computer science theory community, binary decision graphs have been called **branching programs,** and studied extensively in the hope of separating some complexity classes and for studying the amount of space needed to compute various functions (Boppana, & Sipser 1990). Two important theorems tell us that an algorithm in SPACE($S(n)$) for $S(n) \geq \log n$ has a branching program complexity of at most $c^{S(n)}$ for some constant c (Masek 1976), and that constant-width branching programs are very powerful, being able to accept all NC^1 languages (Barrington 1989).

Krause & Waack (1991) studied decision graphs of linear depth and gave exponential lower bounds on several graph accessibility problems. Meinel, Krause & Waack (1988) showed the equivalence of read-once decision graphs with a $\log n$-space bounded **eraser Turing machine**, which has a special read-once-only input tape. By means of an indexing tape, the machine decides in the course of the computation in what order to read the input. After one input cell has been read, it is erased, and the machine will never ask for it again. The relations between the different models, that is, OBDD, Branching Programs, and Decision Trees, are summarized in Meinel (1992).

An interesting point, first mentioned by Lee (1959) and later by Akers (1978) is that a decision diagram actually represents more than one function. Entering the diagram at a different node allows *sharing* functions. This idea was studied by Minato, Ishiura & Yajima (1990) and might be relevant to multi-task learning problems (Caruana 1993), where the goal is to learn several hard tasks at one time. Because subgraphs may be shared, learning may proceed faster, an idea that was exploited for Explanation-Based Learning by Mitchell & Thrun (1993) and more recently, through the learning of invariants, by Thrun & Mitchell (1995).

6.7.2 Learning Decision Graphs

Oliveira & Sangiovanni-Vincentelli (1995) described an algorithm for learning OODGs, under the name Reduced Ordered Decision Graphs. The algorithm starts from a decision tree and converts it to an OODG using the minimum description length principle (Rissanen 1986, Rissanen 1978). Their algorithm performed extremely well on many artificial domains but rather poorly on the real-domains from UC Irvine repository (Murphy & Aha 1995).

General decision graphs were investigated by Oliver, Dowe & Wallace (1992) and Oliver (1993). The algorithms construct decision graphs top-down, by doing a hill-climbing search through the space of graphs, estimating the usefulness of each graph by Wallace's MMLP (minimum message length principle). At each stage a decision is made whether to split a leaf (and which), or whether to join two leaves. Operations that increase the message-length are never performed, hence the algorithm is guaranteed to terminate.

Dvorak (1992) independently discovered the original bottom-up technique we have used here to minimize OBDDs. His motivation was to minimize Boolean functions with "Don't Cares," as opposed to induce a structure with high predictive power. Because he is only interested in "compressing" a function, he does not have to deal with issues of pruning. He suggested a lookahead scheme to minimize the width of the OBDD a few levels up, but concluded the paper by saying that the method can be applied to Boolean functions up to 20 features.

Bahl, Brown, de Souza & Mercer (1989) suggested a **pylon** structure, shown in Figure 6.16. Pylons are very restricted decision graphs: the graphs are of width two, and tests alternate between the right and left columns. Pylons were suggested as the structure to use in composite nodes *within* a decision tree, and claimed that they can "express certain types of semantic questions as well as grammatical questions."

Chou (1991, 1988) suggested a variant of the standard recursive partitioning algorithm that is suitable for constructing directed acyclic decision graphs called **decision trellises**. The idea is that after two levels of a decision tree have been constructed and there are, say, four leaves, the partitioning algorithm tries to partition the instances directly into the four leaves, thus creating a composite node.

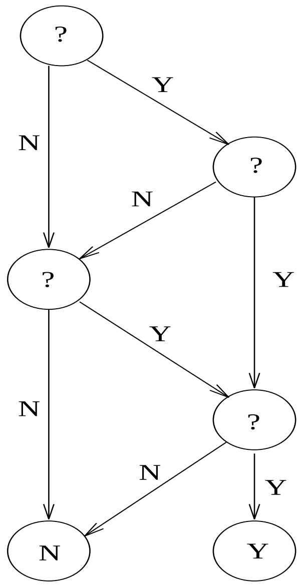

Figure 6.16: A Pylon. Question marks denote tests at the branching nodes.

Michalski & Imam (1994) and Imam (1995) suggested the use of **decision structures**, which are decision trees with complex tests at nodes, disjunctive values on the edges, and probabilistic classifications at the leaves. Polynomially-size decision structures are more powerful than univariate trees because of the wider range of tests allowed (*e.g.*, test equality of two features) but are incomparable to OODGs because they are still trees. The decision structures are constructed from rules (as opposed to construction from instances in the standard supervised machine learning setting), which are in turn derived by the AQ15 system (Michalski, Mozetic, Hong & Lavrac 1986) or one of its variants. The AQDT-2 algorithm selects tests for nodes based on five criteria that measure properties of the decision rules: measuring cost, disjointness of classes, rule importance score, value distribution, and dominance of features.

Results for learning branching programs under Valiant's PAC learning model (Valiant 1984, Angluin 1992) are either negative or still open. Raghavan & Wilkins (1993) showed that even if we restrict ourselves to **μ-branching programs**, where each feature can appear only once in the whole graph, exact PAC learning is impossible either with membership queries alone (membership queries allow the learner to ask whether an instance is in the target class or not) or with equivalence queries alone (equivalence queries allow the learner to ask for a counter example to a given hypothesis). Learning is possible if both types of queries are allowed, but the complexity of their algorithm is $O(n^5)$.

Ergün, Kumar & Rubinfeld (1995) showed how to PAC learn width-two branching programs, and then showed that the existence of an efficient algorithm for learning width-three branching programs would imply the existence of an efficient algorithm for learning DNF, which is still an open problem.

Gavaldà & Guijarro (1995) showed that learning OBDDs is possible in the PAC model if the feature ordering is given and both equivalence and membership queries are allowed. Neither type of query suffices unless P=NP.

6.8 Future Work

Don't be afraid to leave part of the problem for future work.
—desJardins (1994)

The wrapper approach was used to improve the initial ordering selected by the entropy criterion. There is definitely an improvement for some datasets, but one would hope that a better criterion could be found that would be faster than the wrapper.

We project one level at a time, but multi-level projections may be more useful. Specifically, it may be possible to try to project to a level that is highly constrained or that is narrow and then break the problem up into two. The added constraints on the possible placement of incomplete projections might help.

The wrapper swaps the ordering of features and rebuilds the OODG. It may be possible to modify the algorithm to swap features of an OODG that has been built. Minato (1992) and Ishiura, Sawada & Yajima (1991) in the OBDD community showed improvements by swapping features, but because they deal with completely specified functions, swapping is easier.

The discretization we have performed discretizes each feature independently of others. This seems to be working reasonably well, except for the sick-euthyroid dataset. Because we decide on an ordering before building the OODG, it is possible to discretize each feature conditioned on all the previous features, as is done with decision trees. This discretization would be less variable in its discretization than in decision trees because the discretization is done per level and uses all the instances (whereas with decision trees only the instances at a given node are used to discretize the feature tested at the node).

The bias of OODGs may be inappropriate for decomposable problems, where after testing a few features, the features at each subtree are disjoint. A combined approach, which does recursive partitioning and then changes to an OODG, may be a good compromise. At the higher nodes, there are enough instances in the nodes; after a few splits, we can change to an OODG structure to avoid further fragmentation.

The decision tables described in Chapter 5 on page 130 can be converted into oblivious decision trees to create comprehensible structures when many features are used. If we decide on an ordering on the features, as we have done with OODGs, it is possible to avoid the silly prediction of the training set's majority class when a perfect match is not found in the decision table; instead, we can predict the majority class of the parent node. This modification still allows using incremental cross-validation, something important that we have lost in the transition to OODGs. A nice Bayesian interpretation to the use of the parent's majority is that it is equivalent to marginalizing over the last feature in the feature list. This is something that is commonly done for the N-gram model used in speech recognition: the 2-gram model is used to smooth the 3-gram model for combinations that did not appear in the training set (Jelinek 1985). Similar smoothing was done by Buntine (1992) for regular (non-oblivious) decision trees.

6.9 Summary

We have described some properties of oblivious read-once decision graphs (OODGs). Given an ordering on the features, every function has a unique OODG implementing it, which makes the hypothesis space structured and non-redundant. The Kite Theorem shows an envelope bounding the sizes of possible OODGs, and its asymmetric shape was the motivation for the bottom-up construction algorithm.

We described a general bottom-up framework for constructing OODGs, and investigated ways to handle the two main problems: placing (or coloring) incomplete projections and finding a good ordering on the features. The former problem was handled by a greedy algorithm; three solutions were proposed for the latter problem: choosing the feature to minimize the width of the next level, finding an order based on minimizing mutual information, and using the wrapper approach to minimize estimated accuracy. The last two approaches seem to work best, with the wrapper approach performing the best but also the slowest by far.

The HOODG family of algorithms successfully avoids the replication and fragmentation problems, which are two of the main problems with decision trees and the recursive partitioning algorithms that construct them. On the symmetric concepts, such as Monk2-local and m-of-n-3-7-10, the induced OODGs were optimal. On real-world concepts, performance was better than C4.5 on average, but—as expected—there are cases for which OODGs and the learning algorithms are inappropriate, and C4.5's performance is better. In at least one case, the sick-euthyroid dataset, we have identified that the problem is with the discretization algorithm, which removes an important feature by discretizing it into a single interval.

The graphs produced by the HOODG algorithms used less features than C4.5, and were usually smaller, and therefore easier to comprehend but not always. In some cases, as in the DNA dataset, the graph structure seemed more complex, although the accuracy was also higher. We believe that a mixed approach of using decision trees at high levels and switching to OODGs at lower levels, or using OODGs as complex nodes in a tree, may be a promising direction for future work.

Chapter 7

Conclusions

"Where shall I begin, please your Majesty?" he asked.
"Begin at the beginning," the King said, very gravely,
"and go on till you come to the end; then stop."
—Lewis Carroll (Charles Lutwidge Dodgson),
Alice's Adventures in Wonderland

In this dissertation, we dealt with some basic topics in machine learning: accuracy estimation, feature subset selection, and parameter tuning. We also introduced two new hypothesis spaces: decision tables with majority and oblivious decision graphs. In Section 7.1, we briefly summarize the conclusions from this dissertation. In Section 7.2, we make some general remarks about the induction algorithms described in this dissertation and their relation to other algorithms in the field of supervised classification learning. We conclude with general comments about the field in Section 7.3.

7.1 Summary of Results

New ideas sometimes suffer from misinterpretation of original purpose and
overextension of domain of application by impulsive zealots
—Geisser (1975)

In Chapter 3, we reviewed common accuracy estimation methods: holdout, cross-validation, and the .632 bootstrap. We showed examples where each one fails to produce a good estimate and conducted a large-scale experiment comparing cross-validation and the .632 bootstrap on a variety of real-world datasets with differing characteristics. The .632 bootstrap was found to have very large bias in some cases, making it inappropriate for our needs; with cross-validation, one could tradeoff bias for variance by changing the number of folds. While cross-validation has no clear probabilistic interpretation, we showed that it is justified if the induction algorithm is stable for a given dataset. k-fold cross-validation with moderate k values (10-20) reduces the variance while increasing the bias. As k decreases (2-5) and the sample sizes get smaller, there is an increase in the variance due to the instability of the training sets themselves. Repeated cross-validation runs stabilize the estimates for small values of k. In this dissertation, we used a dynamic approach that repeats cross-validation runs based on the estimated variance: if the variance is high, the algorithm executes another run of cross-validation. This approach seems to work well in practice.

In Chapter 4, we defined the wrapper approach, contrasted it with filter approaches, and showed its advantages and disadvantages. We studied the wrapper approach under two settings: feature subset selection and parameter tuning. For feature subset selection, we investigated the relevance and irrelevance of features and concluded that weak and strong relevance are needed to capture our intuition better than just relevance. We then showed that these definitions are mainly useful with respect to an optimal rule, *i.e.*, Bayes rule, and that in practice one should look for optimal features with respect to the specific learning algorithm at hand. Relevance did help motivate compound operators, which are currently the only known practical way to conduct backward searches for feature subsets. For parameter tuning, we have shown that the accuracy of C4.5 can be improved in practice by automatically setting its parameters to achieve high estimated accuracy. The wrapper approach requires a search space, operators, a search engine, and an evaluation function. We investigated all of them in detail. We also showed some problems with the wrapper approach, namely overfitting and the large amounts of CPU times required, and we defined the search problem as an abstract state space search with probabilistic estimates, a formulation that may capture other general problems and that might be studied independently to solve the existing problems.

In Chapter 5, we used a simple hypothesis space, the space of decision tables with a default majority rule (DTMs), to test the conjecture that feature subset selection is a very powerful bias (in the machine learning sense of bias). The accuracy of IDTM was surprisingly high, and we concluded that this bias is extremely important for many real-world datasets. We showed that the resulting decision tables are *very* small and use few features, which can be displayed using General Logic Diagrams (Appendix B on page 203).

It is also possible to convert decision tables into oblivious decision trees or graphs to display them differently and increase their comprehensibility. The ability to incrementally cross-validate the IDTM algorithm and a dataset in time that is linear in the number of instances, the number of features, and the number of label values makes the wrapper approach feasible for large problems that would be impractical with other induction algorithms.

In Chapter 6, we described some properties of oblivious read-once decision graphs (OODGs) and described a general framework for constructing OODGs bottom-up. We proposed three solutions to the main problem of feature ordering: minimizing the width of the next level, finding an order based on minimizing mutual information, and using the wrapper approach to minimize estimated accuracy. The last two approaches seem to work best, with the wrapper approach performing the best but also being the slowest by far. The HOODG family of algorithms successfully avoids the replication and the fragmentation problems, two of the main problems with decision trees and the recursive partitioning algorithms that construct them. The graphs produced by the HOODG algorithms, shown in Appendix D on page 217, used less features than C4.5 and were usually smaller and easier for humans to comprehend.

7.2 Which Algorithm is Best?

> *For every single specific question, you can construct a language or system that is a better answer than C++. C++'s strength comes from being a good answer to many questions rather than being the best answer to one specific question... Thus, the most a general-purpose language can hope for is to be "everybody's second choice."*
> *—Stroustroup (1994)*

There is no algorithm that dominates all others for all problems. For every *single* problem, there is the perfect algorithm that guesses the target function with no input. The best we can hope for is to understand the strengths and limitations of different algorithms, and based on background knowledge for a given domain, make recommendations as to which algorithms to use.

One approach that can work well in practice is to try different algorithms, estimate their accuracy, and pick the one with the highest estimated accuracy. This method will work well as long as there are not too many algorithms being tried and when we have reason to believe that they are appropriate for the domain.

Appendix E on page 244 shows a large experiment comparing the algorithms described in this dissertation with others that are publicly available. On the real-world datasets, the absolute differences are usually small on average, but for many applications they are significant and important (*e.g.*, the relative reduction in error between C4.5 and Naive-Bayes-FSS on the breast cancer dataset is 46.8%). On the real datasets, Naive-Bayes with feature subset selection performed best, followed by simple Naive-Bayes, HOODG, and IDTM. On the artificial datasets, OC1 was the best performer, followed by HOODG and IDTM.

One explanation for the surprising success of Naive-Bayes in this dissertation is the large number of medical datasets used. Although we chose the datasets based on what we believe are rational criteria, it turned out that many of them are related to medical domains, where features are usually well understood and probabilistically independent given the label value.

The other unexpected result in this dissertation is the power of decision tables. In the original paper on decision tables (Kohavi 1995*a*), their power was noted, but they failed for domains with continuous features. In this dissertation, the data has been discretized, and IDTM appears to be a truly powerful algorithm for many real-world datasets.

An interesting observation made by Trevor Hastie is that Naive-Bayes and DTMs represent extreme classifiers in terms of feature interaction. Naive-Bayes assumes that the features are independent given the label and DTMs represent full-interaction models. With the exception of the soybean-large dataset, the differences are surprisingly small given the different models used.

The title of the dissertation has the words "performance enhancement" in it because generic tools such as the wrapper approach work in practice and tend to improve performance on existing algorithms, even though the exact theoretical justification is not yet well understood. The wrapper approach relies on accuracy estimation to guide a search, but all accuracy estimation methods fail sometimes and there is no guarantee that any such meta-level approach will really help.

The power and the weakness of the wrapper approach lies in its generality. For well-understood algorithms, such as linear regression, the wrapper is not very useful because better and faster methods exist for problems of interest such as feature subset selection; however, analyzing complex algorithms such as C4.5 and HOODG is practically impossible, and in these cases the wrapper approach provides an excellent tool.

7.3 Concluding Remarks

Smart companies can be figuring out how they will use a technology, even while its developers are still polishing their prototypes
—Hammer and Champy, Reengineering the Corporation, 1993

An old Kodak advertisement said "you press the button, and we'll do the rest." We wish machine learning could be the same: you gather data, the induction algorithm will do the rest. Regrettably, we know that such a dream is impossible. A lot of effort must be put into choosing features, integrating background knowledge into the induction algorithm, changing the representations, and reiterating. Langley & Simon (to appear) wrote that in their surveyed projects "much of the power comes not from the specific induction method, but from proper formulation of the problems and from crafting the representation to make learning tractable."

Although on a large set of problems induction algorithms tend to perform similarly, there are significant differences for specific datasets. In practice, one is interested in the best algorithm for the data being studied, not an average performance across many domains. This dissertation adds to our understanding of the biases and assumptions made by induction algorithms.

Feigenbaum (1988) wrote: "I have worked in the science and technology of Artificial Intelligence for 20 years and confess to being chronically optimistic about its progress." Herb Simon, at an invited talk in IJCAI-95, said he was only off by about 40 years when he said that in ten years a computer program will beat any human human chess player. We will avoid making concrete predictions about future successes of the machine learning field, but we are optimistic that it will be an influential field in the real world.

Appendix A

Tabulated Results for Accuracy Estimation

The following tables include the detailed experimental results for accuracy estimation.

C4.5 accuracies for cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out	leave 2 out
breast cancer	50/699	89.78±0.66	90.14±0.73	91.19±0.71	91.41±0.71	91.19±0.71	91.59±0.71
chess	900/3196	96.69±0.12	97.79±0.08	97.96±0.08	98.11±0.08	98.16±0.08	98.15±0.08
hypothyroid	400/3163	97.67±0.14	98.23±0.11	98.27±0.12	98.42±0.10	98.35±0.11	98.37±0.12
mushroom	800/8124	98.76±0.06	99.01±0.05	99.08±0.05	99.15±0.06	99.20±0.07	99.21±0.07
soybean	100/683	54.47±0.98	66.36±0.91	68.42±0.78	69.82±0.90	69.82±0.90	71.23±0.94
vehicle	100/846	51.89±1.06	57.62±0.89	59.61±0.79	58.96±0.80	58.96±0.80	59.03±0.97
rand	100/3000	50.12±0.74	49.46±0.77	50.74±0.80	50.00±0.83	50.00±0.83	48.88±0.95

Naive-Bayes accuracies for cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out	leave 2 out
breast cancer	50/699	91.79±0.60	93.91±0.37	94.11±0.35	94.20±0.34	94.11±0.35	94.23±0.35
chess	900/3196	85.61±0.20	86.39±0.20	86.51±0.19	86.61±0.19	86.60±0.18	86.57±0.19
hypothyroid	400/3163	97.31±0.13	97.53±0.10	97.53±0.11	97.55±0.11	97.55±0.10	97.53±0.10
mushroom	800/8124	93.83±0.10	94.32±0.09	94.45±0.08	94.49±0.08	94.52±0.08	94.53±0.08
soybean	100/683	68.99±0.75	76.69±0.61	78.20±0.68	79.03±0.67	79.03±0.67	79.69±0.65
vehicle	100/846	46.66±0.90	47.32±0.75	47.28±0.79	47.27±0.79	47.27±0.79	47.28±0.74
rand	100/3000	49.94±0.84	50.44±0.91	50.14±0.94	50.04±1.00	50.04±1.00	50.40±0.97

C4.5 accuracies for stratified cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out	leave 2 out
breast cancer	50/699	89.90±0.74	91.03±0.66	91.75±0.60	91.52±0.67	91.75±0.60	91.47±0.71
chess	900/3196	96.72±0.11	97.70±0.08	97.88±0.07	98.05±0.08	98.16±0.08	98.14±0.08
hypothyroid	400/3163	97.81±0.12	98.18±0.10	98.33±0.11	98.45±0.10	98.36±0.12	98.39±0.12
mushroom	800/8124	98.81±0.05	99.02±0.05	99.11±0.06	99.18±0.06	99.21±0.07	99.21±0.07
soybean	100/683	61.43±0.91	71.04±0.78	71.83±0.77	71.66±0.90	71.66±0.90	71.44±0.93
vehicle	100/846	54.66±0.93	59.51±0.81	59.94±0.84	59.94±0.90	59.94±0.90	59.50±0.96
rand	100/3000	50.90±0.78	51.28±0.66	50.78±0.62	49.74±0.83	49.74±0.83	49.54±0.93

Naive-Bayes accuracies for stratified cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out	leave 2 out
breast cancer	50/699	91.19±0.63	93.55±0.40	93.95±0.37	93.98±0.39	93.95±0.37	93.91±0.38
chess	900/3196	85.39±0.22	86.22±0.19	86.46±0.18	86.49±0.18	86.58±0.19	86.58±0.19
hypothyroid	400/3163	97.44±0.10	97.52±0.10	97.52±0.10	97.57±0.10	97.54±0.10	97.52±0.10
mushroom	800/8124	94.03±0.09	94.32±0.08	94.47±0.08	94.49±0.08	94.52±0.08	94.53±0.08
soybean	100/683	77.37±0.66	80.08±0.64	80.14±0.63	80.08±0.65	80.08±0.65	79.89±0.67
vehicle	100/846	48.84±0.76	48.16±0.68	47.98±0.59	47.72±0.70	47.72±0.70	47.64±0.73
rand	100/3000	50.44±0.90	50.28±0.92	50.66±0.90	51.40±0.98	51.40±0.98	50.74±0.91

True accuracies for fractions of the data compared with the matching cross-validation estimate (Section 3.4.1 on page 54). 90% corresponds to the sample size used by ten-fold cross-validation 50% corresponds to the sample size used by two-folds.

Dataset	sample-size / total size	C4.5 90% data	5x10 fold	c4.5 50% data	5x2 fold
breast cancer	50/699	91.34±0.33	91.36±0.59	90.13±0.38	90.26±0.56
chess	900/3196	97.86±0.10	97.97±0.07	96.79±0.13	96.75±0.08
hypothyroid	400/3163	98.30±0.09	98.32±0.11	97.88±0.13	97.69±0.07
mushroom	800/8124	99.19±0.08	99.13±0.05	98.71±0.08	98.77±0.04
soybean	100/683	68.59±0.77	68.85±0.74	53.85±1.01	54.40±0.62
vehicle	100/846	58.99±0.49	59.02±0.71	51.71±0.67	51.94±0.63
rand	100/3000	49.92±0.12	50.02±0.63	49.94±0.13	50.12±0.43

True accuracies for fractions of the data compared with the matching stratified estimate. 90% corresponds to the sample size used by ten-fold cross-validation 50% corresponds to the sample size used by two-folds.

Dataset	sample-size / total size	C4.5 90% data	5x10 fold stratified	c4.5 50% data	5x2 fold stratified
breast cancer	50/699	91.34±0.33	91.29±0.64	90.13±0.38	89.82±0.53
chess	900/3196	97.86±0.10	97.96±0.07	96.79±0.13	96.67±0.07
hypothyroid	400/3163	98.30±0.09	98.33±0.10	97.88±0.13	97.87±0.08
mushroom	800/8124	99.19±0.08	99.12±0.05	98.71±0.08	98.75±0.04
soybean	100/683	68.59±0.77	71.74±0.77	53.85±1.01	60.81±0.75
vehicle	100/846	58.99±0.49	60.04±0.74	51.71±0.67	54.20±0.62
rand	100/3000	49.92±0.12	50.27±0.57	49.94±0.13	50.52±0.39

A.1 Five runs

C4.5 accuracies for five times cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out	leave 2 out
breast cancer	50/699	90.26±0.56	90.88±0.57	91.36±0.59	91.35±0.66	91.36±0.59	91.40±0.67
chess	900/3196	96.75±0.08	97.72±0.07	97.97±0.07	98.07±0.07	98.14±0.08	98.15±0.08
hypothyroid	400/3163	97.69±0.07	98.20±0.10	98.32±0.11	98.36±0.11	98.38±0.11	98.38±0.12
mushroom	800/8124	98.77±0.04	99.03±0.04	99.13±0.05	99.17±0.05	99.21±0.07	99.21±0.07
soybean	100/683	54.40±0.62	66.28±0.68	68.85±0.74	70.28±0.81	70.28±0.81	71.25±0.91
vehicle	100/846	51.94±0.63	57.79±0.63	59.02±0.71	59.29±0.79	59.29±0.79	59.31±0.89
rand	100/3000	50.12±0.43	49.74±0.51	50.02±0.63	49.57±0.71	49.57±0.71	49.04±0.88

Naive-Bayes accuracies for five times cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out	leave 2 out
breast cancer	50/699	91.30±0.39	93.74±0.33	94.07±0.34	94.06±0.35	94.07±0.34	94.10±0.35
chess	900/3196	85.63±0.17	86.34±0.19	86.45±0.19	86.52±0.19	86.55±0.19	86.55±0.19
hypothyroid	400/3163	97.33±0.09	97.50±0.10	97.52±0.10	97.54±0.10	97.54±0.10	97.53±0.10
mushroom	800/8124	93.91±0.08	94.32±0.08	94.43±0.08	94.47±0.08	94.51±0.08	94.52±0.08
soybean	100/683	68.52±0.52	76.87±0.60	78.41±0.63	79.20±0.64	79.20±0.64	79.76±0.66
vehicle	100/846	46.92±0.61	47.04±0.66	47.12±0.70	47.37±0.70	47.37±0.70	47.22±0.74
rand	100/3000	50.00±0.61	49.75±0.77	49.81±0.85	50.05±0.91	50.05±0.91	50.00±0.92

C4.5 accuracies for five times stratified cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	Leave 5 out
breast cancer	50/699	89.82±0.53	90.95±0.57	91.29±0.64	91.47±0.64	91.29±0.64
chess	900/3196	96.67±0.07	97.71±0.07	97.96±0.07	98.06±0.07	98.15±0.08
hypothyroid	400/3163	97.87±0.08	98.19±0.10	98.33±0.10	98.37±0.11	98.38±0.11
mushroom	800/8124	98.75±0.04	99.05±0.04	99.12±0.05	99.16±0.06	99.21±0.07
soybean	100/683	60.81±0.75	71.01±0.69	71.74±0.77	71.67±0.83	71.67±0.83
vehicle	100/846	54.20±0.62	58.92±0.62	60.04±0.74	60.10±0.83	60.10±0.83
rand	100/3000	50.52±0.39	50.94±0.42	50.27±0.57	49.66±0.69	49.66±0.69

Naive-Bayes accuracies for five times stratified cross-validation with varying folds

Dataset	size	2 folds	5 folds	10 folds	20 folds	leave 5 out
breast cancer	50/699	91.41±0.36	93.67±0.32	93.96±0.33	94.03±0.35	93.96±0.33
chess	900/3196	85.57±0.16	86.27±0.18	86.47±0.19	86.52±0.18	86.56±0.19
hypothyroid	400/3163	97.34±0.10	97.51±0.10	97.52±0.10	97.53±0.10	97.54±0.10
mushroom	800/8124	93.89±0.09	94.32±0.08	94.44±0.08	94.49±0.08	94.52±0.08
soybean	100/683	76.47±0.57	80.19±0.61	80.08±0.63	80.14±0.65	80.14±0.65
vehicle	100/846	47.61±0.52	47.64±0.65	47.62±0.65	47.57±0.71	47.57±0.71
rand	100/3000	51.16±0.63	50.99±0.82	51.10±0.86	50.92±0.92	50.92±0.92

A.2 Varying Times

C4.5 accuracies for varying times on two-fold cross-validation

Dataset	one time	2 times	3 times	5 times	10 times	20 times	50 times	100 times
breast cancer	89.78±0.66	90.21±0.58	89.98±0.61	90.26±0.56	90.30±0.50	90.06±0.48	89.94±0.48	89.92±0.47
chess	96.69±0.12	96.73±0.09	96.72±0.09	96.75±0.08	96.74±0.07	96.75±0.06	96.74±0.06	96.74±0.06
hypothyroid	97.67±0.14	97.73±0.09	97.73±0.09	97.69±0.07	97.75±0.07	97.70±0.08	97.70±0.07	97.70±0.07
mushroom	98.76±0.06	98.77±0.05	98.79±0.05	98.77±0.04	98.77±0.04	98.78±0.04	98.77±0.04	98.77±0.04
soybean	54.47±0.98	53.83±0.72	53.92±0.58	54.40±0.62	54.72±0.65	54.94±0.64	54.81±0.65	54.70±0.65
vehicle	51.89±1.06	52.53±0.80	52.29±0.71	51.94±0.63	52.37±0.58	52.33±0.53	52.55±0.52	52.64±0.52
rand	50.12±0.74	50.39±0.65	50.21±0.53	50.12±0.43	50.03±0.38	50.04±0.34	50.13±0.31	50.07±0.31

Naive-Bayes accuracies for varying times on two-fold cross-validation

Dataset	one time	2 times	3 times	5 times	10 times	20 times	50 times	100 times
breast cancer	91.79±0.60	91.91±0.46	91.53±0.43	91.30±0.39	91.24±0.32	90.97±0.28	91.00±0.27	91.02±0.27
chess	85.61±0.20	85.62±0.19	85.67±0.18	85.63±0.17	85.53±0.16	85.52±0.16	85.52±0.17	85.51±0.17
hypothyroid	97.31±0.13	97.32±0.11	97.31±0.10	97.33±0.09	97.35±0.09	97.34±0.09	97.35±0.09	97.35±0.09
mushroom	93.83±0.10	93.86±0.09	93.89±0.09	93.91±0.08	93.90±0.08	93.94±0.08	93.94±0.08	93.94±0.08
soybean	68.99±0.75	68.81±0.68	68.43±0.60	68.52±0.52	68.77±0.54	68.89±0.52	68.62±0.54	68.73±0.53
vehicle	46.66±0.90	46.82±0.71	46.87±0.66	46.92±0.61	47.10±0.52	47.06±0.50	47.08±0.49	47.03±0.49
rand	49.94±0.84	49.60±0.73	49.85±0.66	50.00±0.61	49.78±0.60	50.00±0.59	50.01±0.59	49.98±0.60

C4.5 accuracies for varying times on ten-fold cross-validation

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	91.19±0.71	91.35±0.62	91.37±0.61	91.36±0.59	91.29±0.59	91.29±0.59
chess	900/3196	97.96±0.08	97.95±0.07	97.96±0.07	97.97±0.07	97.94±0.07	97.94±0.07
hypothyroid	400/3163	98.27±0.12	98.29±0.11	98.29±0.11	98.32±0.11	98.31±0.10	98.31±0.10
mushroom	800/8124	99.08±0.05	99.12±0.05	99.09±0.05	99.13±0.05	99.13±0.05	99.12±0.05
soybean	100/683	68.42±0.78	68.56±0.76	68.60±0.75	68.85±0.74	68.92±0.75	69.08±0.73
vehicle	100/846	59.61±0.79	59.02±0.76	59.14±0.75	59.02±0.71	58.96±0.72	58.92±0.72
rand	100/3000	50.74±0.80	50.70±0.69	50.34±0.68	50.02±0.63	49.86±0.57	49.94±0.53

Naive-Bayes accuracies for varying times on ten-fold cross-validation

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	94.11±0.35	94.13±0.34	94.15±0.33	94.07±0.34	94.00±0.33	93.95±0.33
chess	900/3196	86.51±0.19	86.50±0.18	86.47±0.18	86.45±0.19	86.44±0.18	86.43±0.18
hypothyroid	400/3163	97.53±0.11	97.54±0.10	97.51±0.10	97.52±0.10	97.52±0.10	97.52±0.10
mushroom	800/8124	94.45±0.08	94.44±0.08	94.44±0.08	94.43±0.08	94.42±0.08	94.43±0.08
soybean	100/683	78.20±0.68	78.38±0.62	78.34±0.62	78.41±0.63	78.40±0.61	78.43±0.62
vehicle	100/846	47.28±0.79	47.16±0.75	47.14±0.73	47.12±0.70	47.20±0.69	47.24±0.67
rand	100/3000	50.14±0.94	49.84±0.86	49.93±0.86	49.81±0.85	49.77±0.88	50.03±0.87

C4.5 accuracies for varying times on two-fold stratified cross-validation

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	89.90±0.74	90.07±0.56	89.78±0.60	89.82±0.53	90.04±0.49	90.20±0.45
chess	900/3196	96.72±0.11	96.65±0.10	96.69±0.08	96.67±0.07	96.75±0.07	96.75±0.06
hypothyroid	400/3163	97.81±0.12	97.87±0.09	97.88±0.08	97.87±0.08	97.82±0.07	97.81±0.07
mushroom	800/8124	98.81±0.05	98.76±0.04	98.75±0.04	98.75±0.04	98.78±0.04	98.77±0.04
soybean	100/683	61.43±0.91	60.37±0.82	60.90±0.75	60.81±0.75	61.01±0.76	61.19±0.76
vehicle	100/846	54.66±0.93	54.41±0.74	54.20±0.65	54.20±0.62	54.30±0.56	54.19±0.52
rand	100/3000	50.90±0.78	50.39±0.56	50.55±0.47	50.52±0.39	50.68±0.38	50.84±0.33

Naive-Bayes accuracies for varying times on two-fold stratified cross-validation

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	91.87±0.59	91.79±0.43	91.43±0.39	91.41±0.36	91.35±0.29	91.38±0.26
chess	900/3196	85.54±0.19	85.58±0.16	85.63±0.15	85.57±0.16	85.54±0.17	85.58±0.17
hypothyroid	400/3163	97.34±0.12	97.34±0.11	97.36±0.11	97.34±0.10	97.36±0.10	97.37±0.10
mushroom	800/8124	93.82±0.10	93.87±0.09	93.90±0.09	93.89±0.09	93.92±0.08	93.93±0.08
soybean	100/683	76.26±0.64	76.44±0.60	76.39±0.59	76.47±0.57	76.59±0.55	76.51±0.56
vehicle	100/846	47.10±0.77	46.92±0.66	47.27±0.56	47.61±0.52	47.65±0.53	47.53±0.51
rand	100/3000	50.84±0.90	51.23±0.78	51.08±0.72	51.16±0.63	50.92±0.59	50.94±0.59

C4.5 accuracies for varying times on stratified ten-fold cross-validation

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	91.75±0.60	91.40±0.65	91.23±0.63	91.29±0.64	91.35±0.62	91.43±0.60
chess	900/3196	97.88±0.07	97.99±0.07	97.99±0.07	97.96±0.07	97.95±0.07	97.96±0.07
hypothyroid	400/3163	98.33±0.11	98.34±0.10	98.33±0.11	98.33±0.10	98.36±0.10	98.34±0.10
mushroom	800/8124	99.11±0.06	99.12±0.05	99.12±0.05	99.12±0.05	99.12±0.05	99.12±0.05
soybean	100/683	71.83±0.77	71.87±0.79	71.61±0.77	71.74±0.77	71.58±0.73	71.60±0.73
vehicle	100/846	59.94±0.84	59.79±0.79	59.62±0.79	60.04±0.74	60.03±0.74	59.93±0.74
rand	100/3000	50.78±0.62	50.52±0.69	50.06±0.65	50.27±0.57	50.45±0.53	50.28±0.51

Naive-Bayes accuracies for varying times on stratified ten-fold cross-validation

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	93.95±0.37	93.95±0.36	93.98±0.34	93.96±0.33	93.94±0.34	93.95±0.33
chess	900/3196	86.46±0.18	86.51±0.19	86.49±0.18	86.47±0.19	86.49±0.19	86.48±0.19
hypothyroid	400/3163	97.52±0.10	97.50±0.10	97.51±0.10	97.52±0.10	97.52±0.10	97.54±0.10
mushroom	800/8124	94.47±0.08	94.46±0.08	94.44±0.08	94.44±0.08	94.43±0.08	94.44±0.08
soybean	100/683	80.14±0.63	80.07±0.62	80.08±0.61	80.08±0.63	80.15±0.63	80.09±0.63
vehicle	100/846	47.98±0.59	47.47±0.68	47.41±0.67	47.62±0.65	47.63±0.66	47.65±0.65
rand	100/3000	50.66±0.90	51.46±0.83	50.94±0.86	51.10±0.86	51.08±0.88	50.89±0.86

A.3 Trimming

C4.5 accuracies for varying times with 10% trimmed two-fold cross-validation. Note that only at five times does trimming kick in because below that we have less than 10 folds.

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	89.78±0.66	90.21±0.58	89.98±0.61	90.75±0.56	90.86±0.53	90.70±0.51
chess	900/3196	96.69±0.12	96.73±0.09	96.72±0.09	96.81±0.08	96.81±0.07	96.82±0.06
hypothyroid	400/3163	97.67±0.14	97.73±0.09	97.73±0.09	97.78±0.08	97.84±0.08	97.79±0.08
mushroom	800/8124	98.76±0.06	98.77±0.05	98.79±0.05	98.77±0.05	98.78±0.04	98.79±0.04
soybean	100/683	54.47±0.98	53.83±0.72	53.92±0.58	54.56±0.67	54.86±0.67	55.14±0.66
vehicle	100/846	51.89±1.06	52.53±0.80	52.29±0.71	52.02±0.64	52.53±0.59	52.48±0.52
rand	100/3000	50.12±0.74	50.39±0.65	50.21±0.53	50.18±0.44	50.05±0.39	50.05±0.34

Naive-Bayes accuracies for varying times with 10% trimmed two-fold cross-validation. Note that only at five times does trimming kick in because below that we have less than 10 folds.

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	91.79±0.60	91.91±0.46	91.53±0.43	91.87±0.38	91.94±0.33	91.88±0.30
chess	900/3196	85.61±0.20	85.62±0.19	85.67±0.18	85.68±0.17	85.58±0.17	85.57±0.16
hypothyroid	400/3163	97.31±0.13	97.32±0.11	97.31±0.10	97.39±0.09	97.40±0.09	97.39±0.09
mushroom	800/8124	93.83±0.10	93.86±0.09	93.89±0.09	93.90±0.08	93.90±0.08	93.93±0.08
soybean	100/683	68.99±0.75	68.81±0.68	68.43±0.60	68.59±0.53	68.78±0.54	68.93±0.52
vehicle	100/846	46.66±0.90	46.82±0.71	46.87±0.66	46.76±0.62	47.04±0.52	47.03±0.50
rand	100/3000	49.94±0.84	49.60±0.73	49.85±0.66	50.08±0.62	49.80±0.58	50.01±0.59

C4.5 accuracies for varying times with 10% trimmed ten-fold cross-validation.

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	92.70±0.74	92.84±0.64	93.06±0.63	93.10±0.60	92.99±0.60	93.04±0.58
chess	900/3196	98.07±0.08	98.08±0.08	98.10±0.07	98.11±0.07	98.08±0.07	98.09±0.07
hypothyroid	400/3163	98.60±0.12	98.60±0.11	98.60±0.11	98.60±0.11	98.61±0.11	98.61±0.10
mushroom	800/8124	99.26±0.06	99.29±0.06	99.27±0.05	99.32±0.05	99.30±0.05	99.30±0.05
soybean	100/683	68.86±0.80	68.93±0.79	68.99±0.79	69.29±0.79	69.42±0.78	69.51±0.75
vehicle	100/846	59.76±0.80	59.21±0.77	59.37±0.77	59.22±0.73	59.20±0.74	59.18±0.75
rand	100/3000	50.63±0.79	50.67±0.68	50.37±0.67	50.03±0.64	49.90±0.58	49.92±0.54

Naive-Bayes accuracies for varying times with 10% trimmed ten-fold cross-validation.

Dataset	size	one time	2 times	3 times	5 times	10 times	20 times
breast cancer	50/699	95.82±0.37	96.06±0.37	96.10±0.36	95.98±0.35	95.88±0.35	95.88±0.35
chess	900/3196	86.60±0.20	86.61±0.19	86.58±0.18	86.58±0.19	86.55±0.19	86.54±0.18
hypothyroid	400/3163	97.78±0.12	97.81±0.11	97.78±0.11	97.79±0.10	97.80±0.10	97.79±0.10
mushroom	800/8124	94.54±0.09	94.52±0.08	94.53±0.08	94.55±0.08	94.53±0.08	94.54±0.08
soybean	100/683	78.59±0.74	78.90±0.67	78.86±0.65	78.93±0.64	78.97±0.63	78.95±0.64
vehicle	100/846	47.40±0.82	47.17±0.77	47.10±0.76	47.05±0.72	47.09±0.70	47.17±0.69
rand	100/3000	50.28±0.92	49.84±0.88	49.83±0.88	49.74±0.86	49.80±0.89	49.99±0.89

A.4 Is Stratification Optimistic?

True accuracies for portions of the data. 90% corresponds to the sample size ten-fold cross-validation is using and 50% corresponds to the sample size two-fold is using.

Dataset	sample-size / total size	C4.5 90% data	Naive-Bayes 90% data	c4.5 50% data	Naive-Bayes 50% data
breast cancer	50/699	91.34±0.33	93.92±0.29	90.13±0.38	91.26±0.68
chess	900/3196	97.86±0.10	86.60±0.22	96.79±0.13	85.91±0.30
hypothyroid	400/3163	98.30±0.09	97.62±0.05	97.88±0.13	97.52±0.06
mushroom	800/8124	99.19±0.08	94.33±0.10	98.71±0.08	93.80±0.12
soybean	100/683	68.59±0.77	78.41±0.47	53.85±1.01	68.34±0.54
vehicle	100/846	58.99±0.49	47.60±0.51	51.71±0.67	47.26±0.47
rand	100/3000	49.92±0.12	50.01±0.13	49.94±0.13	50.08±0.13

Appendix B

General Logic Diagrams for IDTM

A-priori, one might not consider decision-tables as comprehensible structures, unless they are very small. In fact, for all datasets, except for soybean-large and DNA, they *are* very small. The following figures show the General Logic Diagram (GLD) for all the datasets except three: the soybean-large dataset could not be displayed with our software because it has 19 classes; the Monk2 and m-of-n-3-7-10 datasets were not displayed because the accuracy was very low (m-of-n-3-7-10 just predicts the majority class).

A GLD provides a way of displaying up to about ten dimensions in a graphical representation that can be easily understood. GLDs were described in Michalski (1978) and later used in Thrun *et al.* (1991), Wnek & Michalski (1994), and Kohavi (1994*d*).

Each possible instance in the projected space defined by the decision table's schema has exactly one box that is shaded according to the prediction made there (all GLDs shown below have only two shades of grey for the classes, except for DNA which has three classes and thus three shades of grey). Readers familiar with Karnaugh maps may note a resemblance, except that the ordering of feature values does not conform to the hamming distance restriction in Karnaugh maps (GLDs are not restricted to Boolean features).

To avoid showing ten diagrams, we used 1/3 holdout (the training set was two-thirds of the dataset, the test set was one third of the dataset) and IDTM to generate the GLDs. Because the data was pre-discretized, ranges are shown for continuous features, with "inf" denoting infinity. The ordering on the features was chosen to be the same ordering found by the wrappers, *i.e.*, the first two features found are the outermost features on each axis.

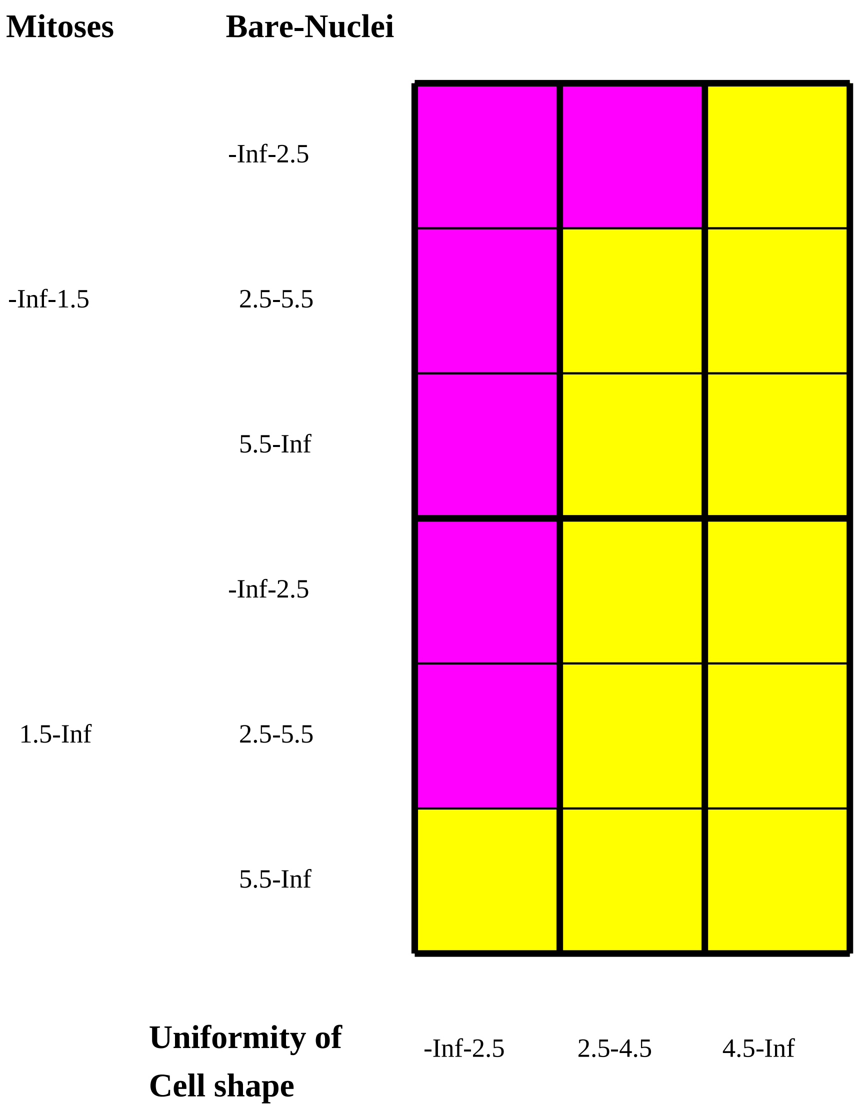

Figure B.1: The GLD for IDTM on dataset breast cancer.
Accuracy: 94.42%

To classify an instance using the GLD, one finds the range (or value) of the outer features and then repeats for inner features. For example, an instance with Mitoses of 2 would fall in the lower half of the GLD because it is in the range 1.5 to infinity. If the Bare-Nuclei is 1, it would then fall in the fourth row of table because it is in the range -infinity to 2.5 and Mitoses indicated that we should be in the lower half. The Uniformity of Cell Shape would determine the unique square within that row, and the shade of grey determines the predicted class.

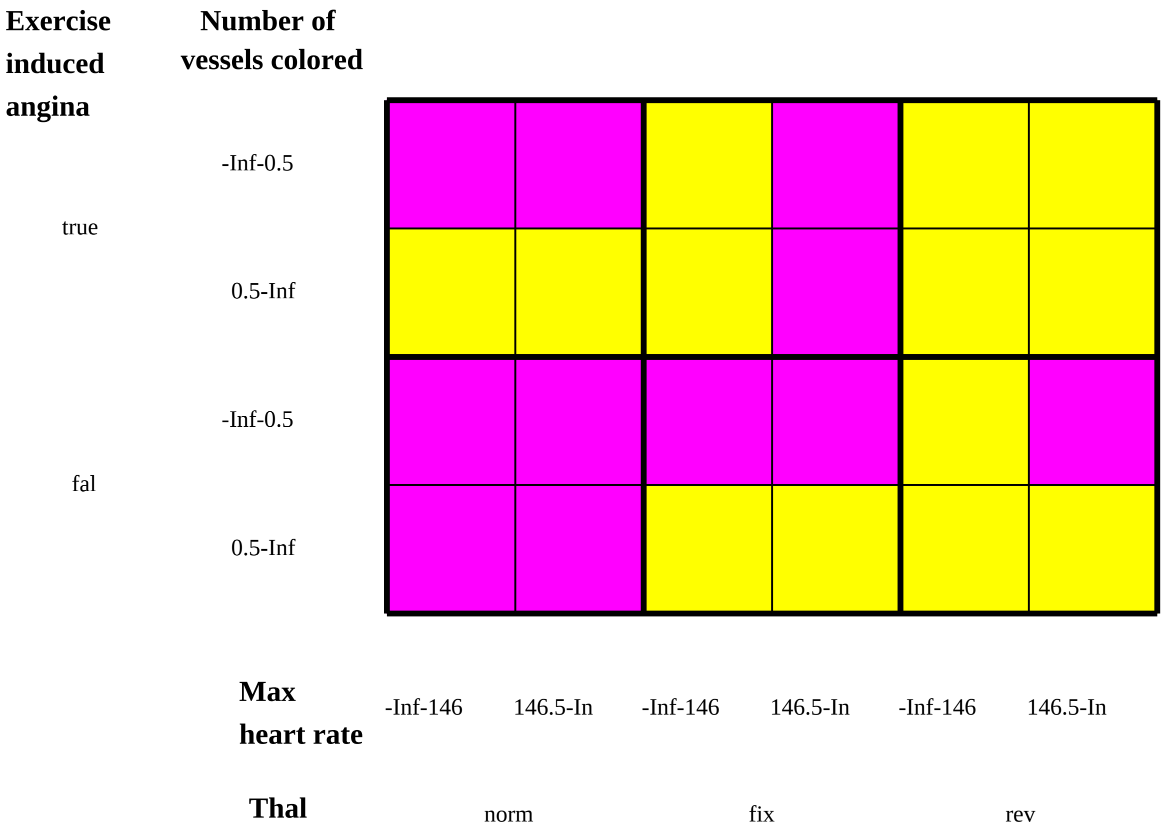

Figure B.2: The GLD for IDTM on dataset cleveland heart disease.
Accuracy: 79.21%

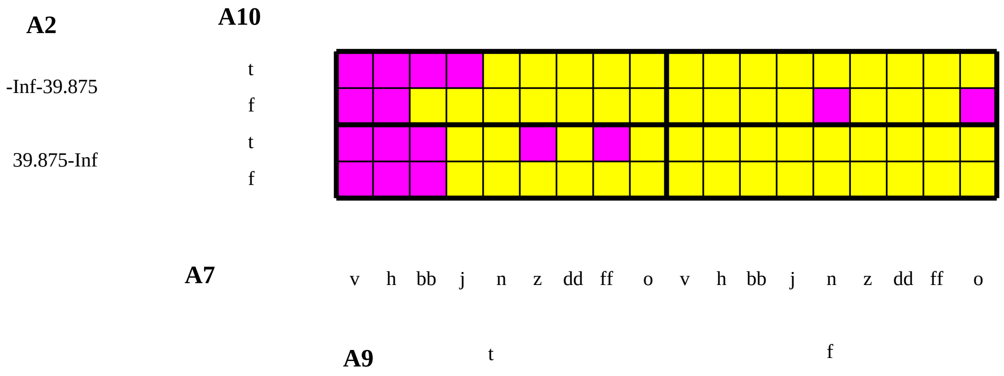

Figure B.3: The GLD for IDTM on dataset australian credit screening (crx). Accuracy: 80.50%

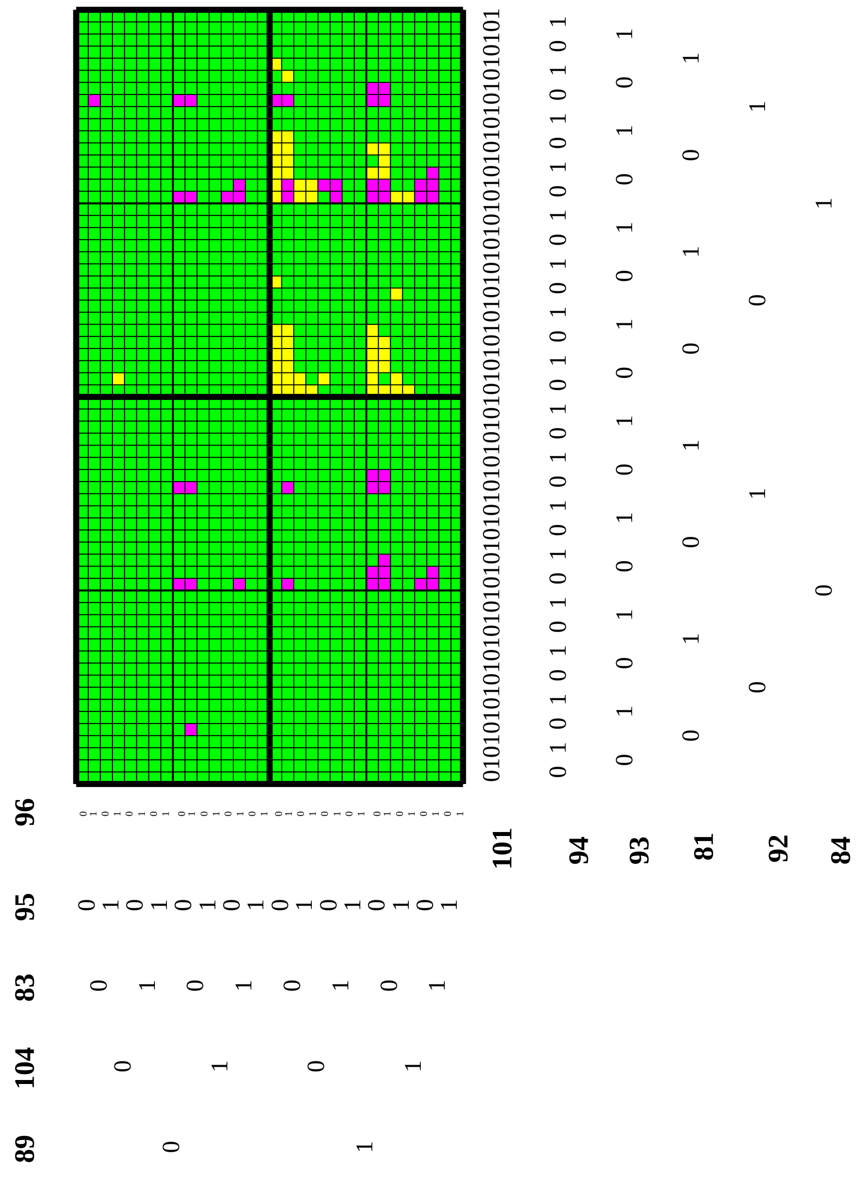

Figure B.4: The GLD for IDTM on dataset DNA split junctions.
Accuracy: 94.60%

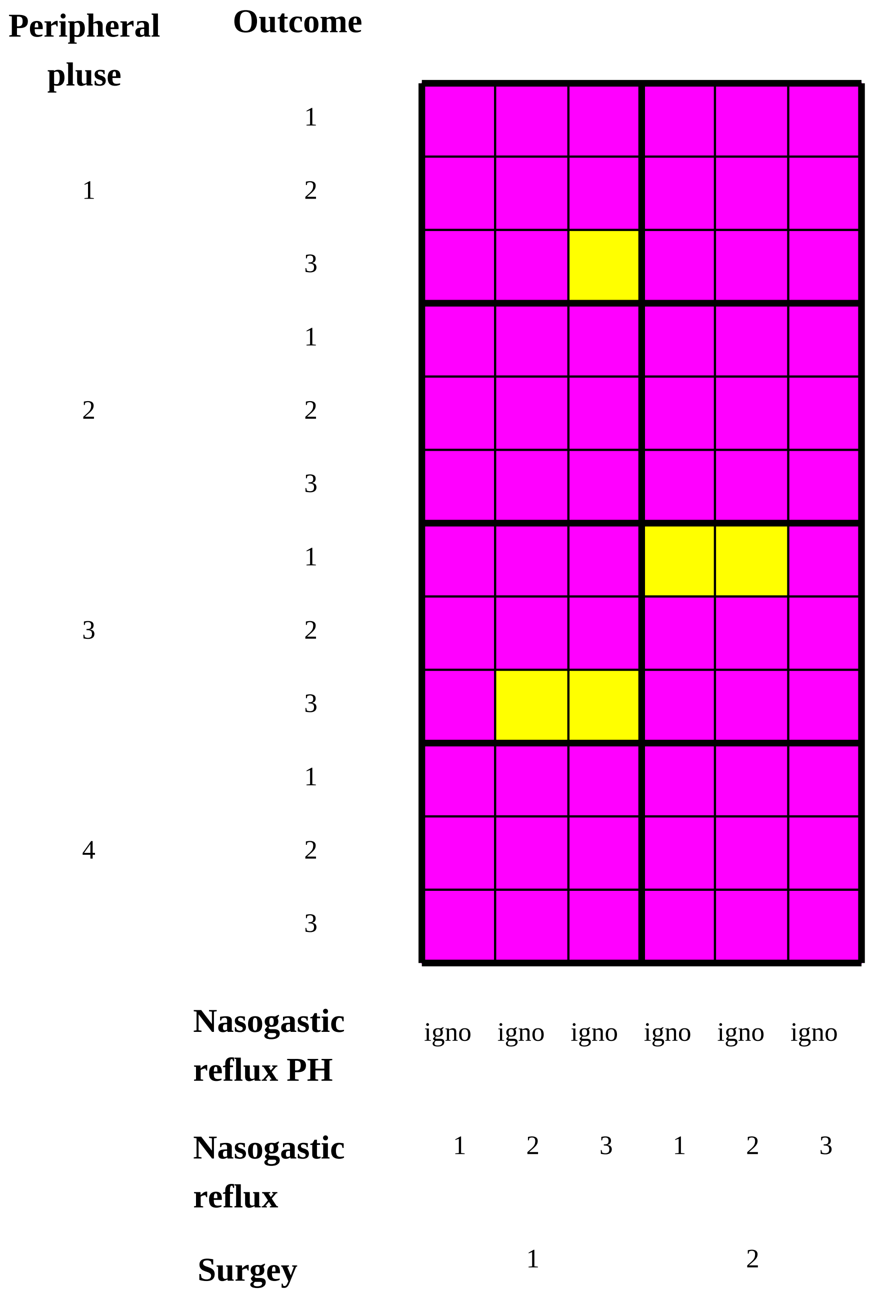

Figure B.5: The GLD for IDTM on dataset horse colic.
Accuracy: 77.94%

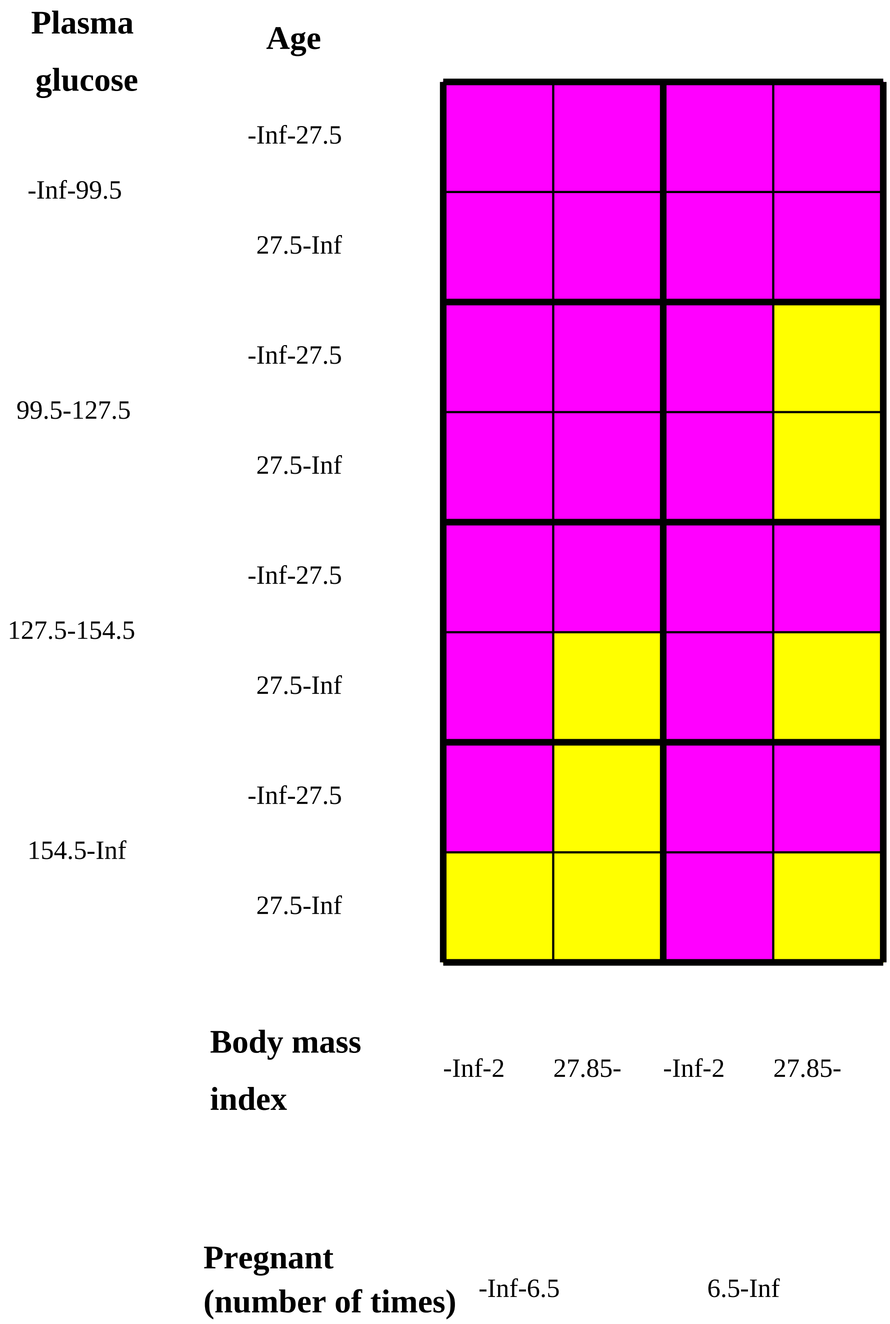

Figure B.6: The GLD for IDTM on dataset Pima Indian diabetes.
Accuracy: 79.30%

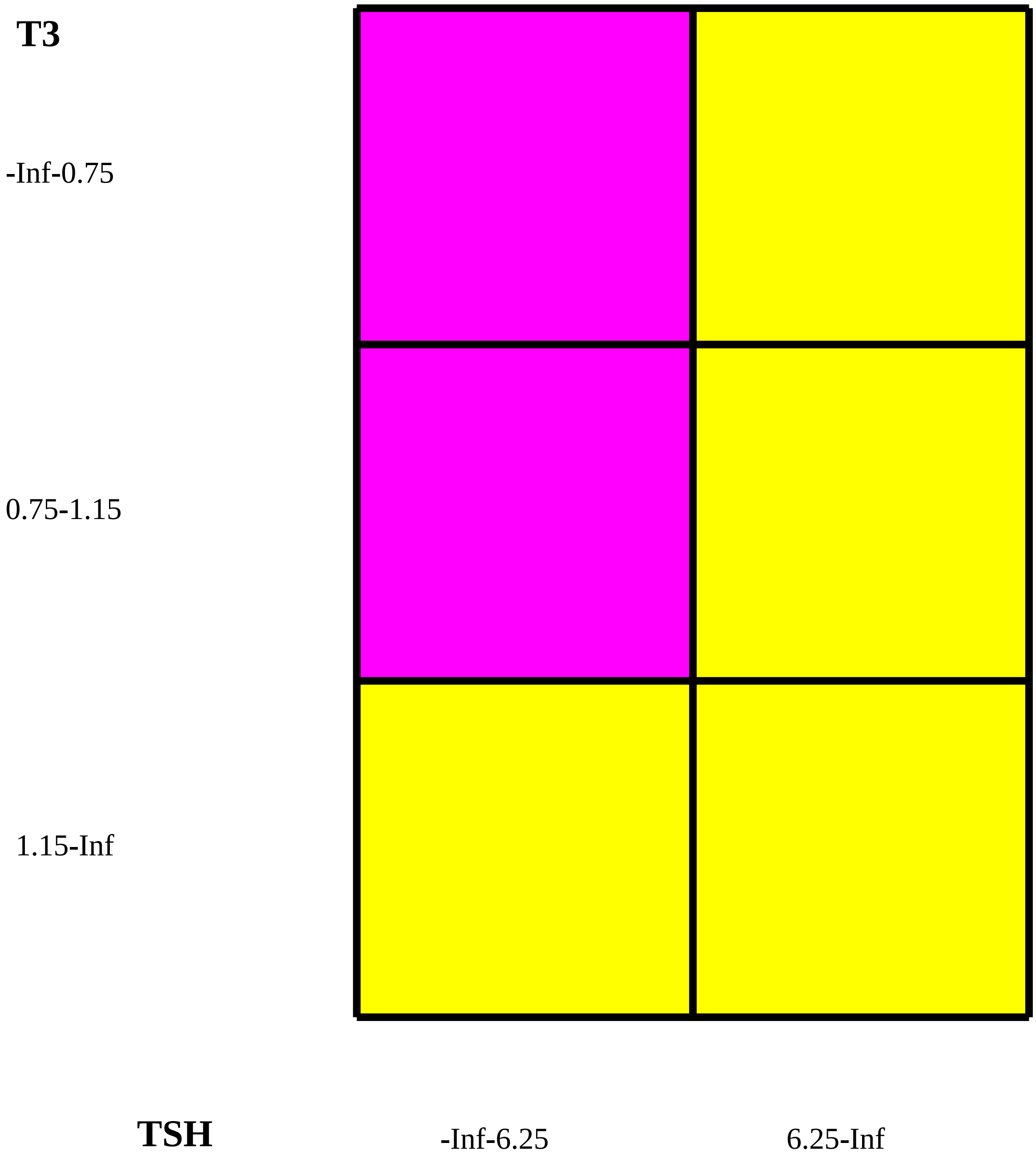

Figure B.7: The GLD for IDTM on dataset sick euthyroid.
Accuracy: 97.35%

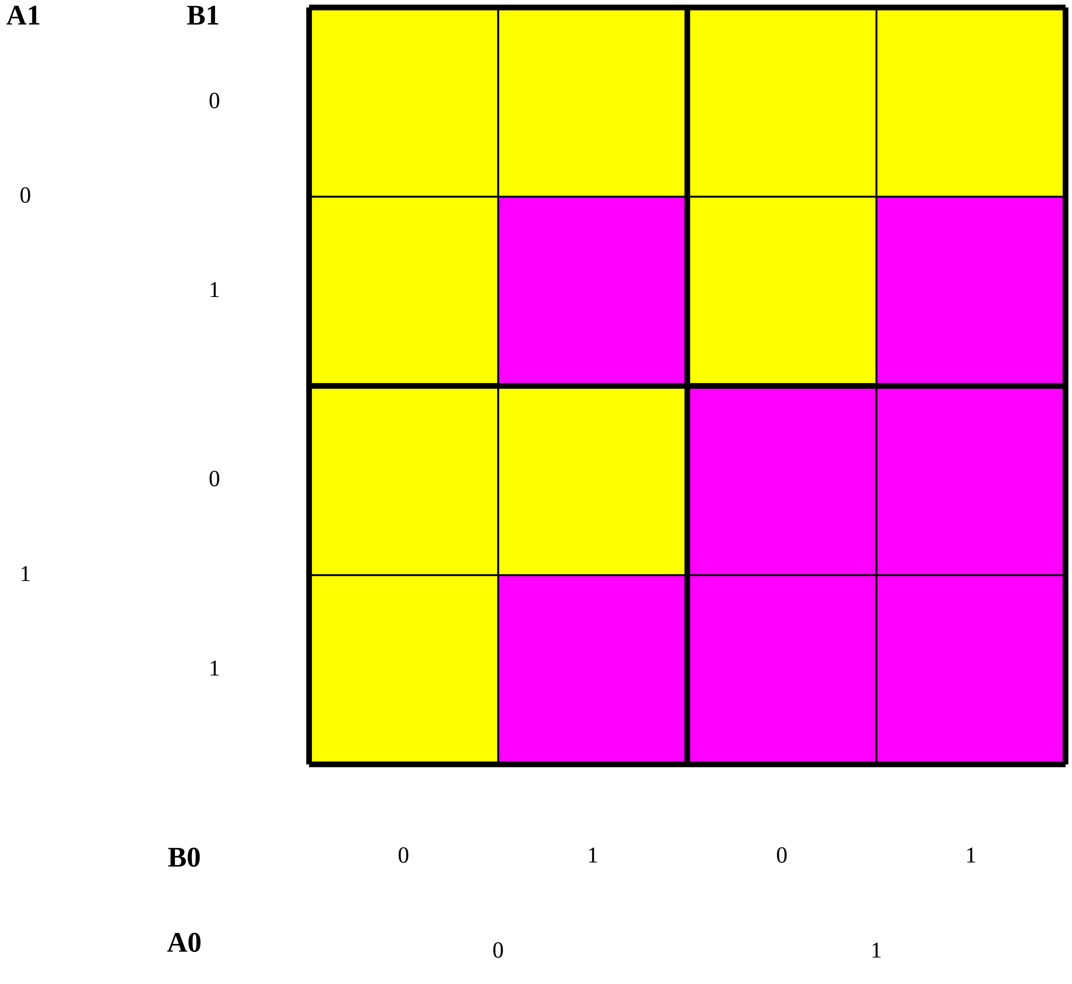

Figure B.8: The GLD for IDTM on dataset corral.
Accuracy: 100.00%

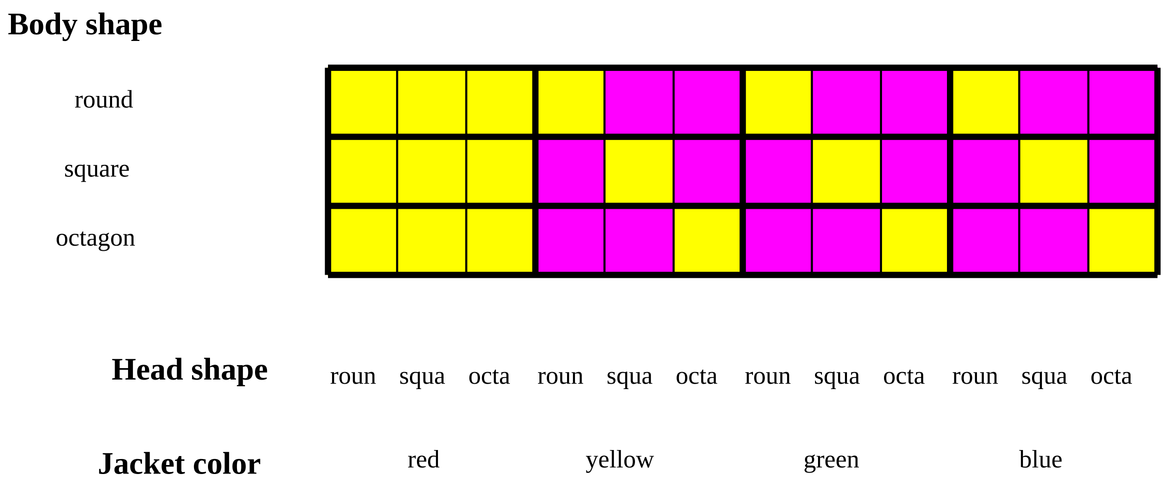

Figure B.9: The GLD for IDTM on dataset Monk1.
Accuracy: 100.00%

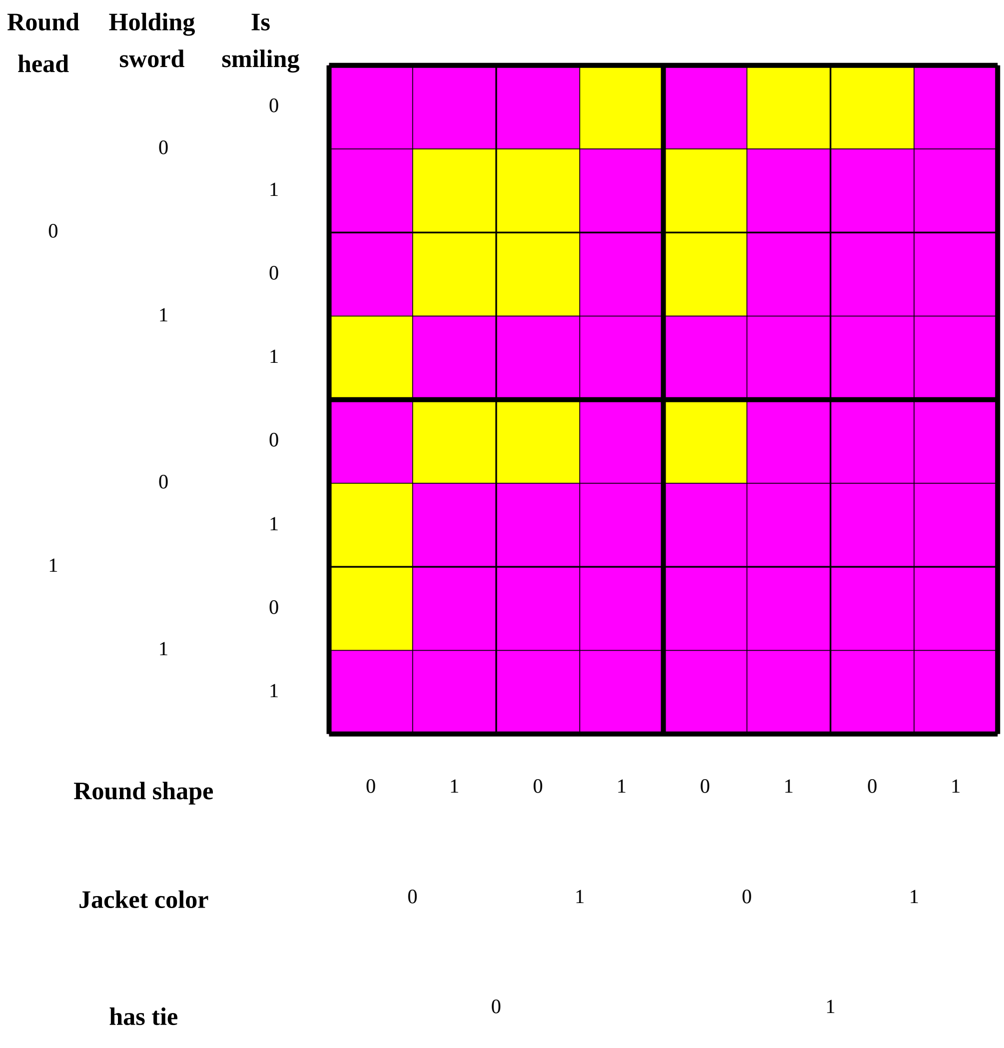

Figure B.10: The GLD for IDTM on dataset Monk2-local.
Accuracy: 100.00%

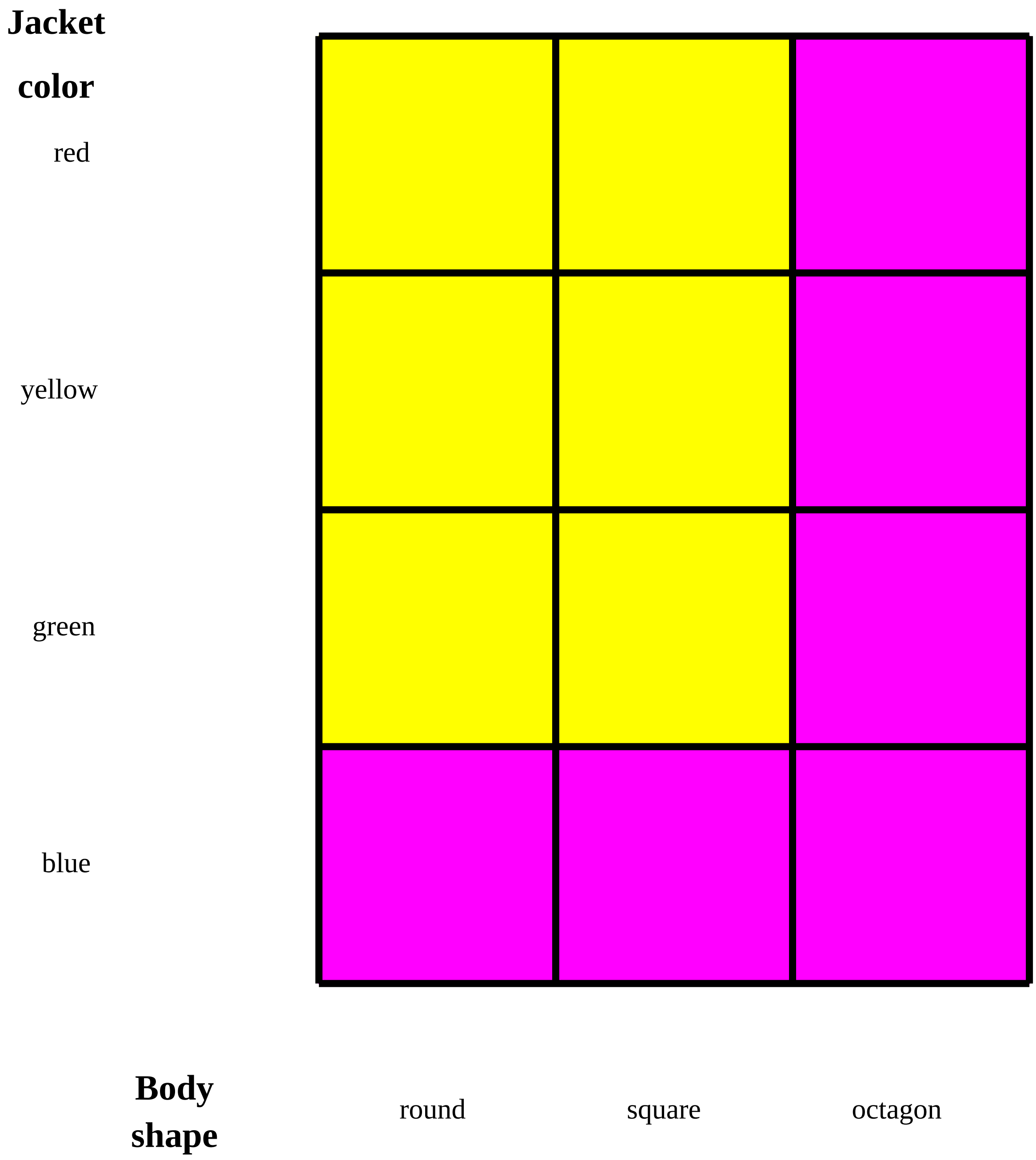

Figure B.11: The GLD for IDTM on dataset Monk3.
Accuracy: 97.22%

Appendix C

Hardness of Minimal Projection

We provide a proof that minimal projection is NP-complete (Theorem 14 on page 172).

Theorem 14

The following decision problem is NP-complete:
Given a set of labelled instances, an ordering on the n features, and two positive integers w and ℓ; is there an OODG that has width $\leq w$ at level ℓ and that correctly classifies all instances?

Proof: To show NP-completeness, we have to show that the problem is in NP, and that it is NP-hard. The problem is in NP because verifying a given solution is easy to do in polynomial time: one only needs to check that the graph is legal, that level ℓ is of width w, and that all instances are correctly classified by the OODG. Note that if there are m instances in the input, the width of a level need never exceed m, and so the overall size of the OODG can be bounded by $m \cdot n$ (polynomial in the input size). To prove that the problem is NP-hard, we show a reduction from k-colorability (chromatic number).

Graph k-colorability is defined as follows (Garey & Johnson 1979, p. 191). Given a graph $G = (V, E)$ and a positive integer $k \leq |V|$, is G k-colorable, *i.e.*, does there exist a function $f : V \mapsto \{1, 2, \ldots, k\}$ such that $f(u) \neq f(v)$ whenever $\langle u, v\rangle \in E$?

Given a graph $G = (V, E)$ for the k-colorability problem, we construct the following projection problem for OODGs. We define w, the desired width, to be k and define ℓ, the desired level to minimize, to be $\lceil \log n \rceil + 1$ (all logs are base two). Without loss of generality, we assume that the ordering of features is X_1 to X_n (otherwise, shuffle the bits in the instances defined below according to the order). The instances we construct will all contain $2\lceil \log n \rceil$ bits that encode two numbers, each one of size $\log n$ bits (*i.e.*, the first number will occupy bits 1 to $\lceil logn \rceil$ and the second number will occupy bits $\lceil \log n \rceil + 1$ to $2\lceil \log n \rceil$). We will denote each instance by $\langle u \cdot v, c \rangle$, where u and v are numbers that can be represented by $\lceil \log n \rceil$ bits each (they make up the unlabelled part of the instance), and c is the Boolean label. We now describe the construction of the training set, which will be of size $2 \cdot |E| + |V|$:

1. For each node v in G, construct an instance $\langle v \cdot v, 0 \rangle$.

2. For each edge in $\langle u, v \rangle \in E$ create two instances: $\langle u \cdot v, 1 \rangle$ and $\langle v \cdot u, 1 \rangle$.

We now have to show that G is k-colorable iff an appropriate OODG can be constructed for the training set defined.

(k-colorability $\rightarrow$ OODG): Given a coloring function f, we show how to construct level ℓ. Create k nodes, 1 to k, and assign each instance $< u \cdot v, c >$ to node $f(u)$. We can clearly build an OODG mapping the instances from the root to level ℓ because by our assignment, there cannot be two different nodes with the same first $\lceil \log n \rceil$ bits at different nodes in level ℓ, hence nodes 1 to ℓ need to implement a deterministic function on $\ell - 1$ features, which is always possible. To show that we can build the bottom half of the OODG, we need to show that no two instances at the same node have their last $\lceil \log n \rceil$ bits the same (*i.e.*, all instances at a node are *consistent*). Once this is shown, it is not hard to see that one can build an oblivious decision tree from each node, and then compress the tree to an OODG. Suppose, to the contrary, that there exist two instances at a given node that are inconsistent, *i.e.*, we have $\vec{x}_1 = \langle u \cdot v, 1 \rangle$ and $\vec{x}_2 = \langle v \cdot v, 0 \rangle$ (all nodes going to 0 are of the form $\langle v \cdot v, 0 \rangle$ and inconsistency means that $\vec{x}_1$ has the same last $\lceil \log n \rceil$ bits as $\vec{x}_2$). According to our construction, $\langle u \cdot v, 1 \rangle$ indicates an edge between nodes u and v and we have a contradiction with the fact that f was a legal coloring.

(OODG$\rightarrow$ k-colorability): Given an OODG, number the nodes at level ℓ in the OODG, and define f, the coloring function, to map each node v to the node number that instance $\langle v \cdot 0 \rangle$ reaches at level ℓ. To show that this is a legal graph coloring, assume to the contrary, that there exist an edge $\langle u, v \rangle \in E$ and that nodes u, v are assigned the same color. In the OODG, the instances $\langle u \cdot v \rangle$ and $\langle v \cdot v \rangle$ must then both pass through the same node at level ℓ, but this implies that the OODG does not correctly classify both, since from level ℓ, the first has to reach leaf 1 and the other leaf 0. ∎

Appendix D

The Graphs for C4.5 and HOODG-Middle

The comprehensibility of a classifier's structure (*e.g.*, a decision tree or graph) is a subjective judgment. The following figures show the trees and the OODGs generated by C4.5 and HOODG-Middle. To avoid showing ten trees for the cross-validation runs, we used a 1/3 holdout (the training set was two-thirds of the dataset, the test set was one third of the dataset) to generate the trees and graphs.

Because the OODG graphs depend on discretized data, each edge is labelled with the discretized interval, where "inf" denotes infinity.

Note: the graphs shown are intended to show the structures induced. Because some graphs contain many nodes, the text at the nodes may not be readable; the exact text is unimportant for our purpose, which is to compare the structures of the induced trees and graphs.

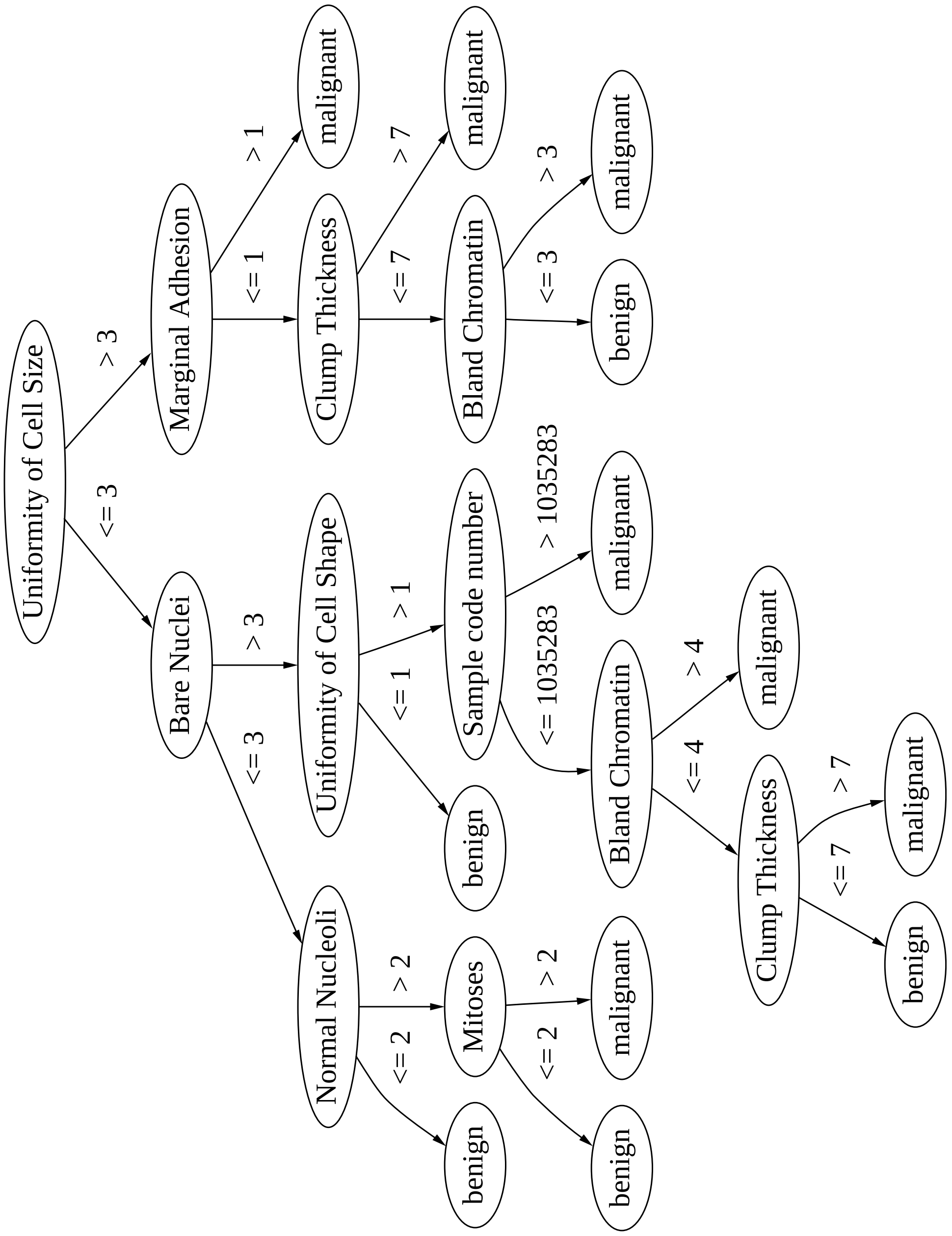

Figure D.1: The decision tree generated by C4.5 for breast cancer.
Accuracy: 94.69%.
Note the split on sample code number.

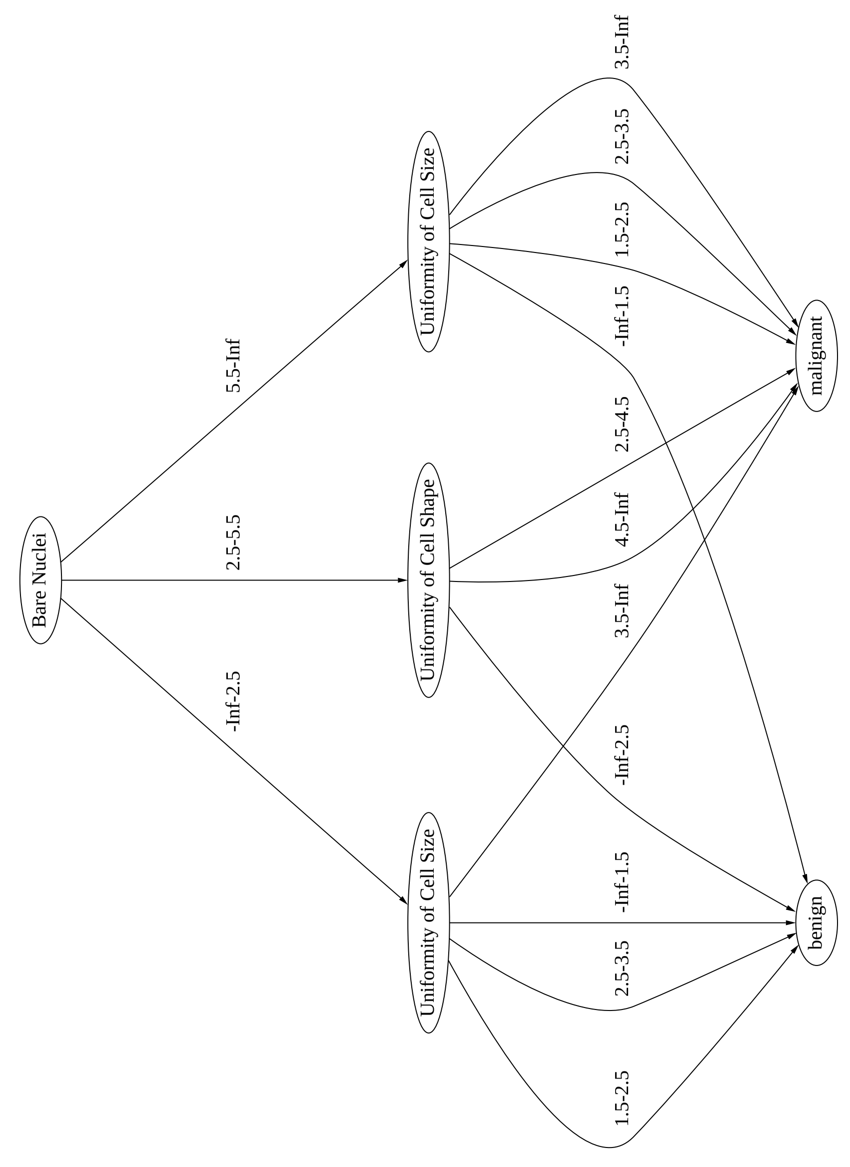

Figure D.2: The OODG generated by HOODG-Middle for breast cancer. Accuracy: 96.02%.

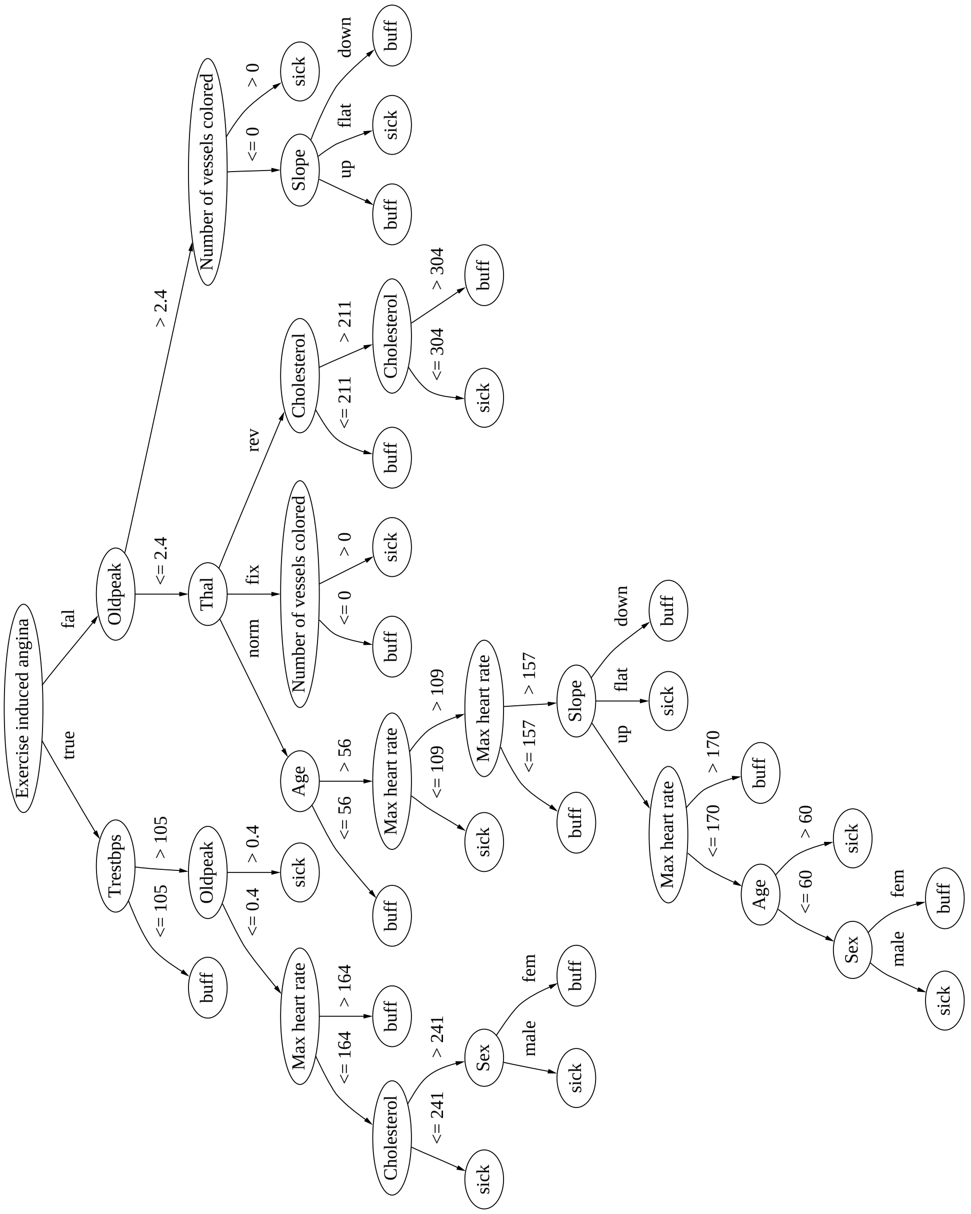

Figure D.3: The decision tree generated by C4.5 for cleveland heart disease. Accuracy: 73.74%.

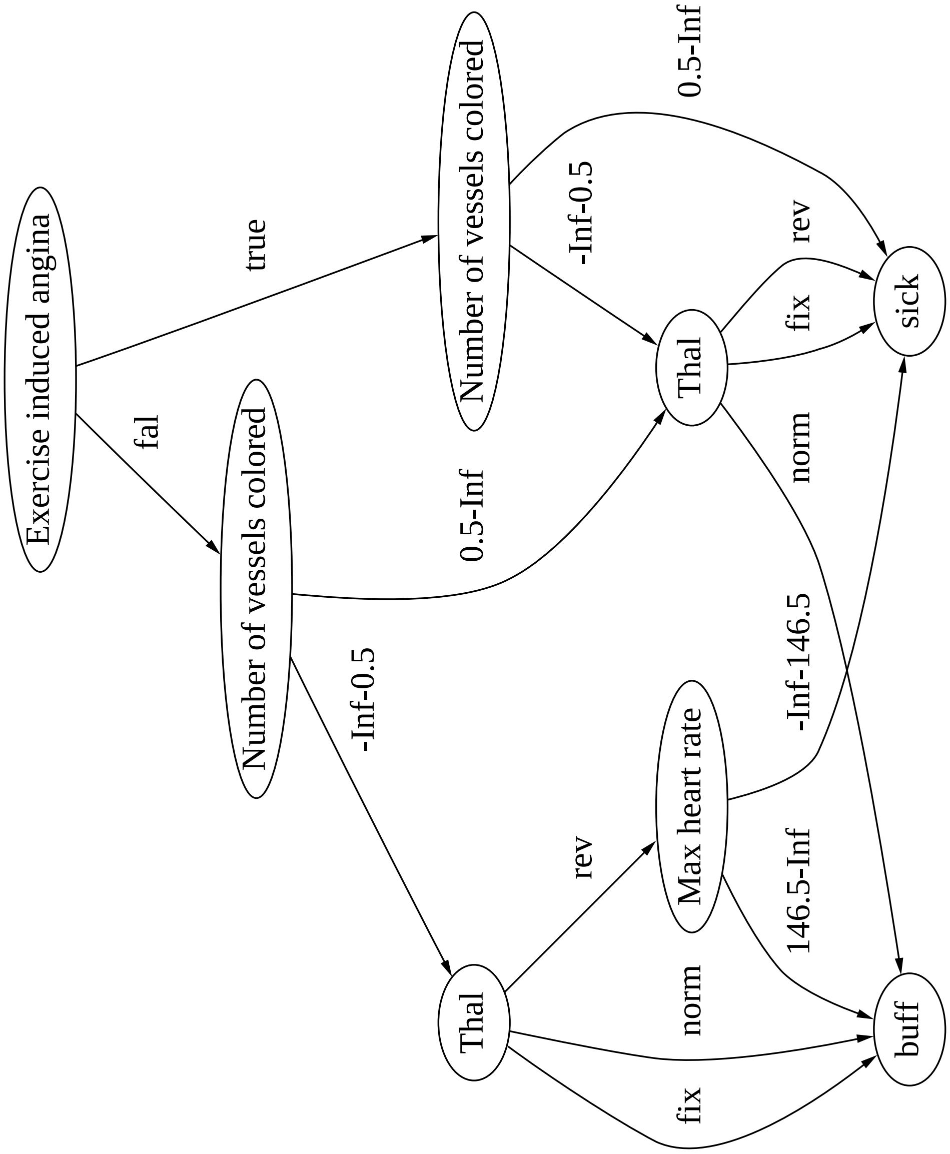

Figure D.4: The OODG generated by HOODG-Middle for cleveland heart disease. Accuracy: 78.79%.

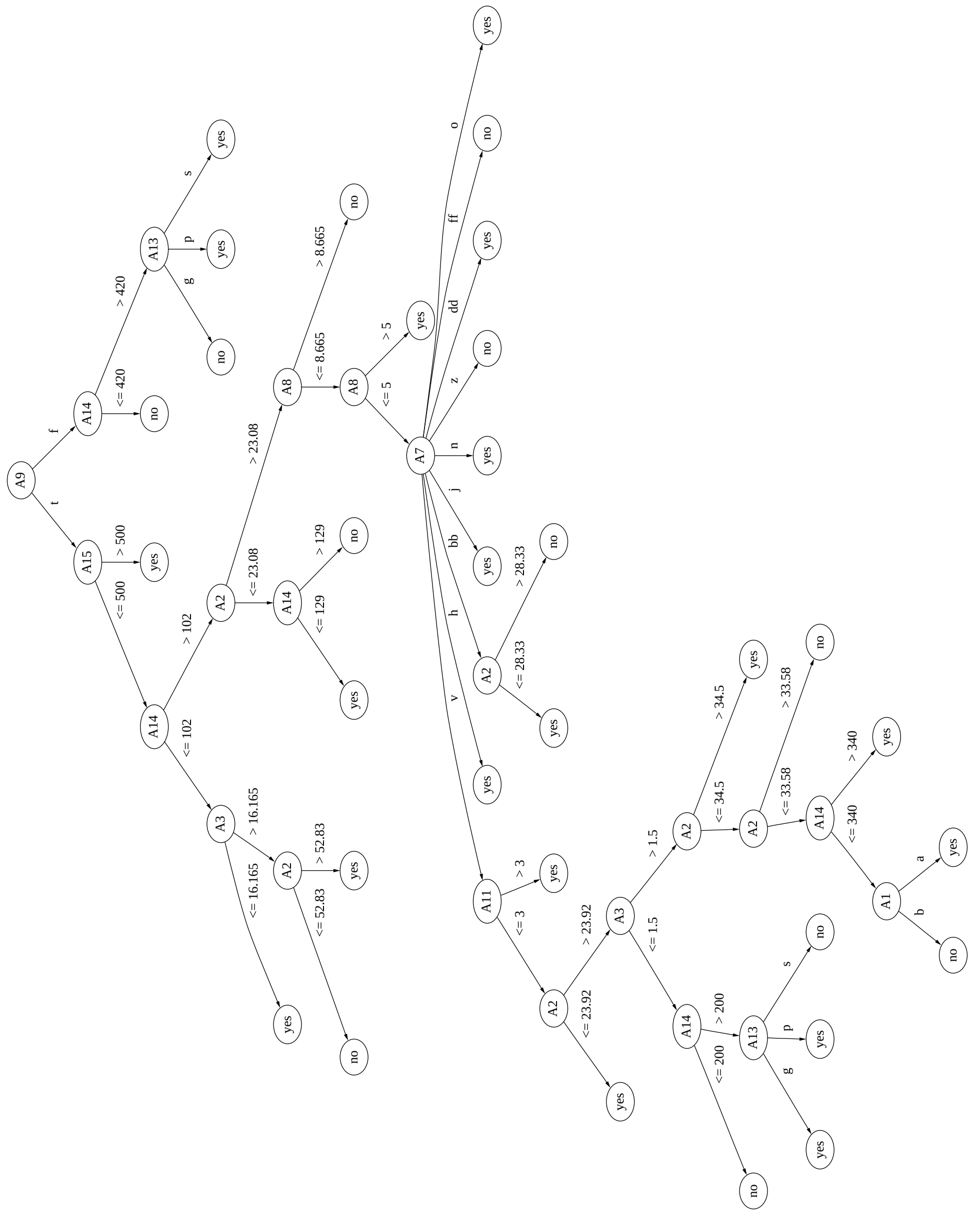

Figure D.5: The decision tree generated by C4.5 for australian credit screening (crx). Accuracy: 84.57%.

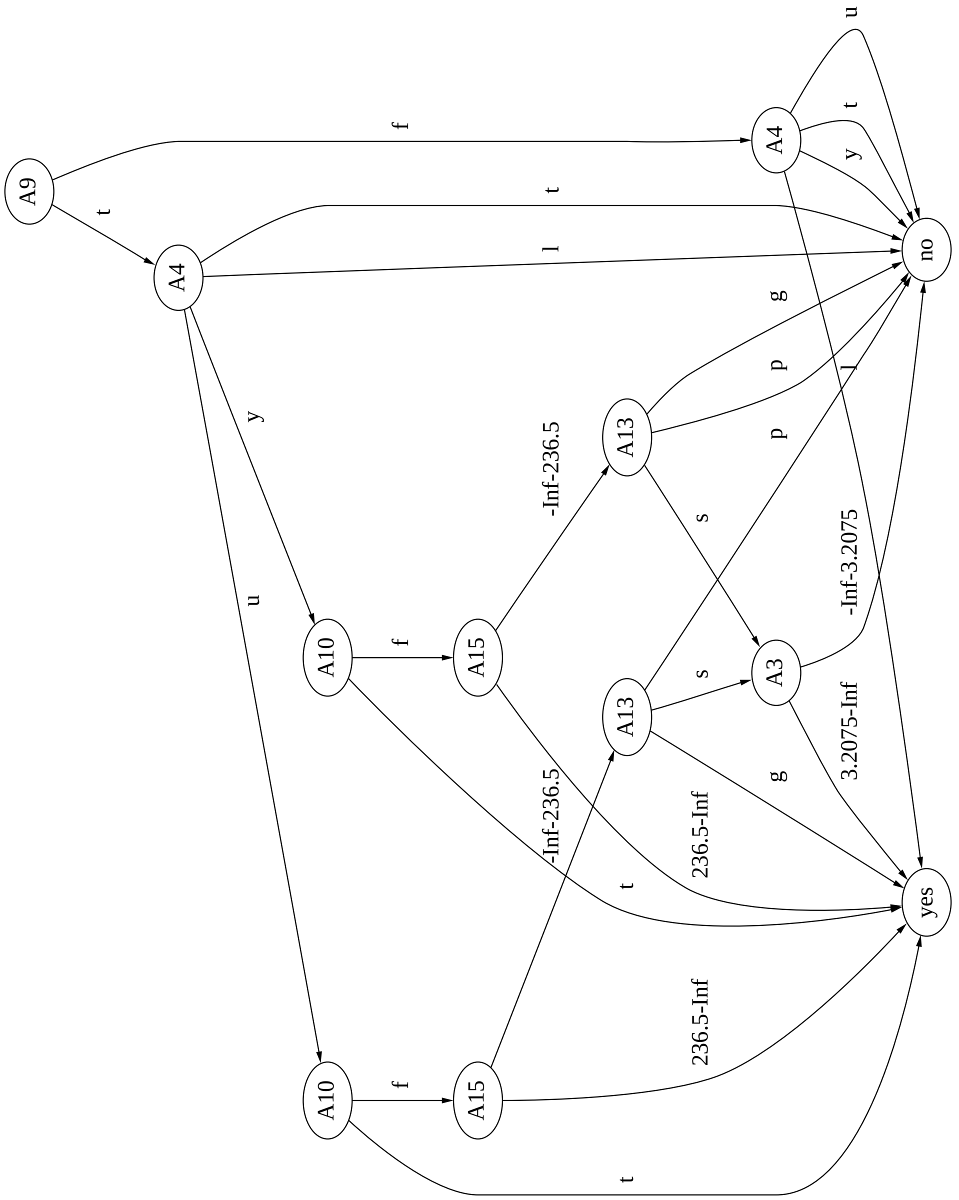

Figure D.6: The OODG generated by HOODG-Middle for australian credit screening (crx). Accuracy: 84.04%.

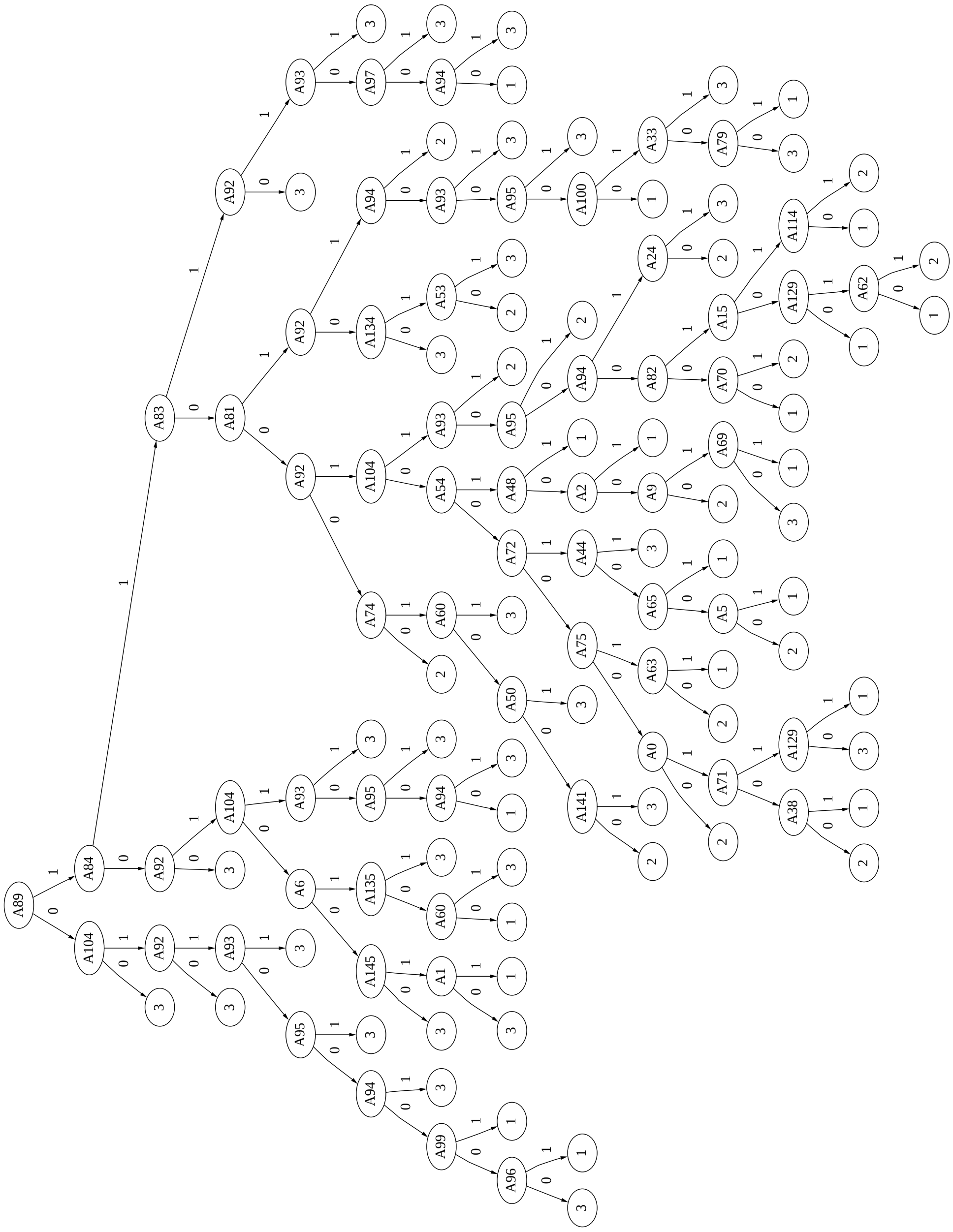

Figure D.7: The decision tree generated by C4.5 for DNA split junctions. Accuracy: 92.66%.

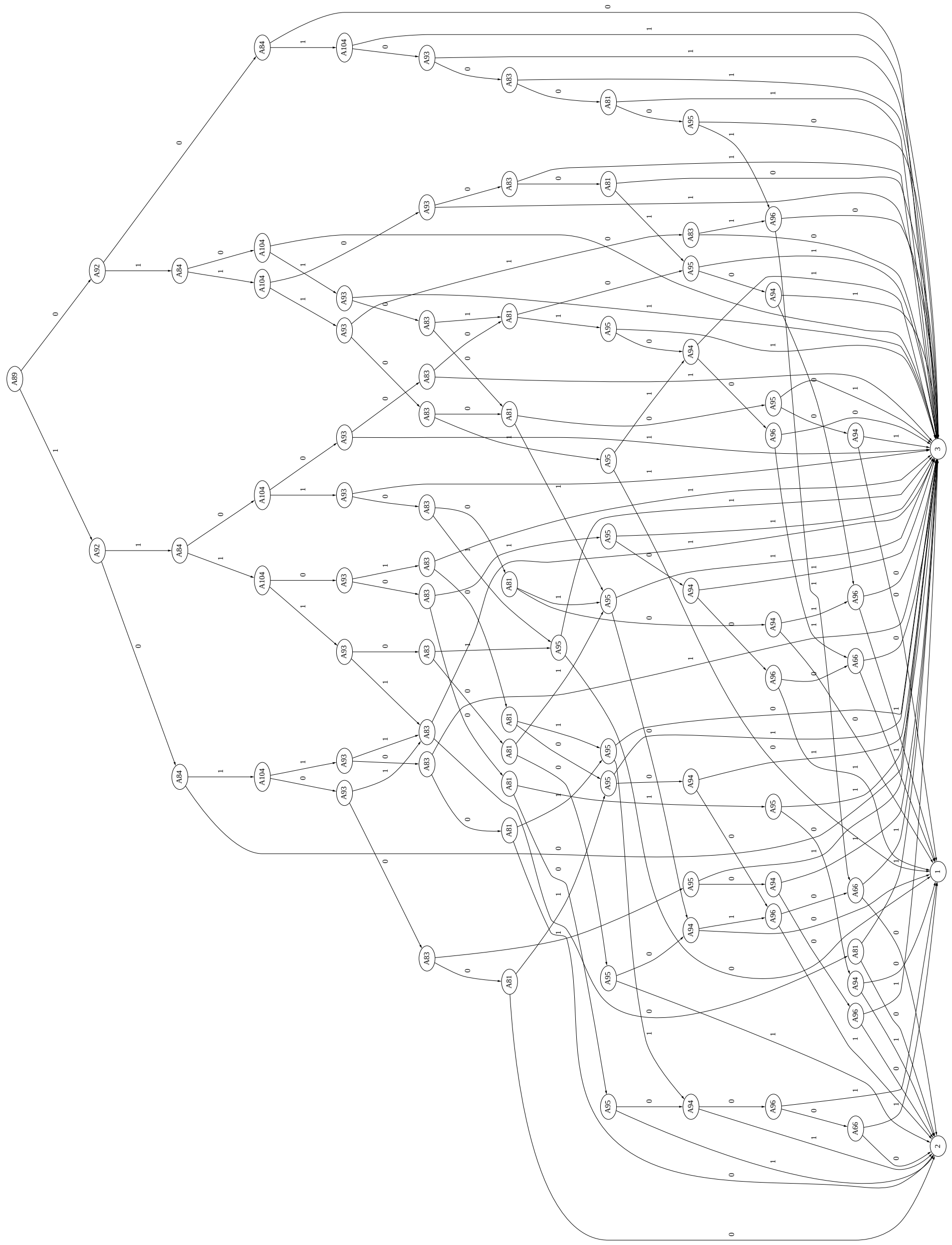

Figure D.8: The OODG generated by HOODG-Middle for DNA split junctions.
Accuracy: 94.10%.
Not very comprehensible, but the accuracy difference is significant: 19.6% relative improvement in accuracy over C4.5

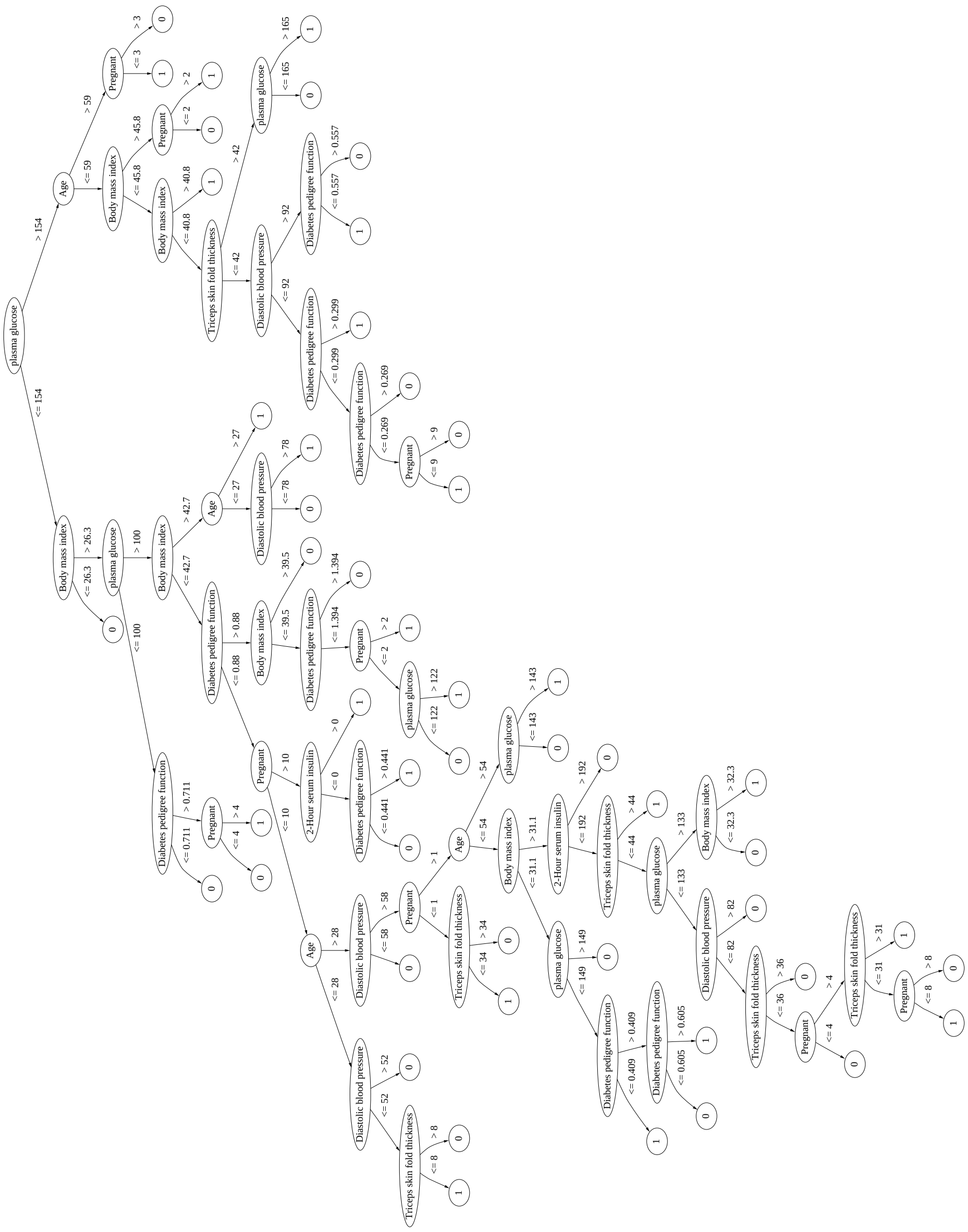

Figure D.9: The decision tree generated by C4.5 for Pima Indian diabetes. Accuracy: 76.17%.

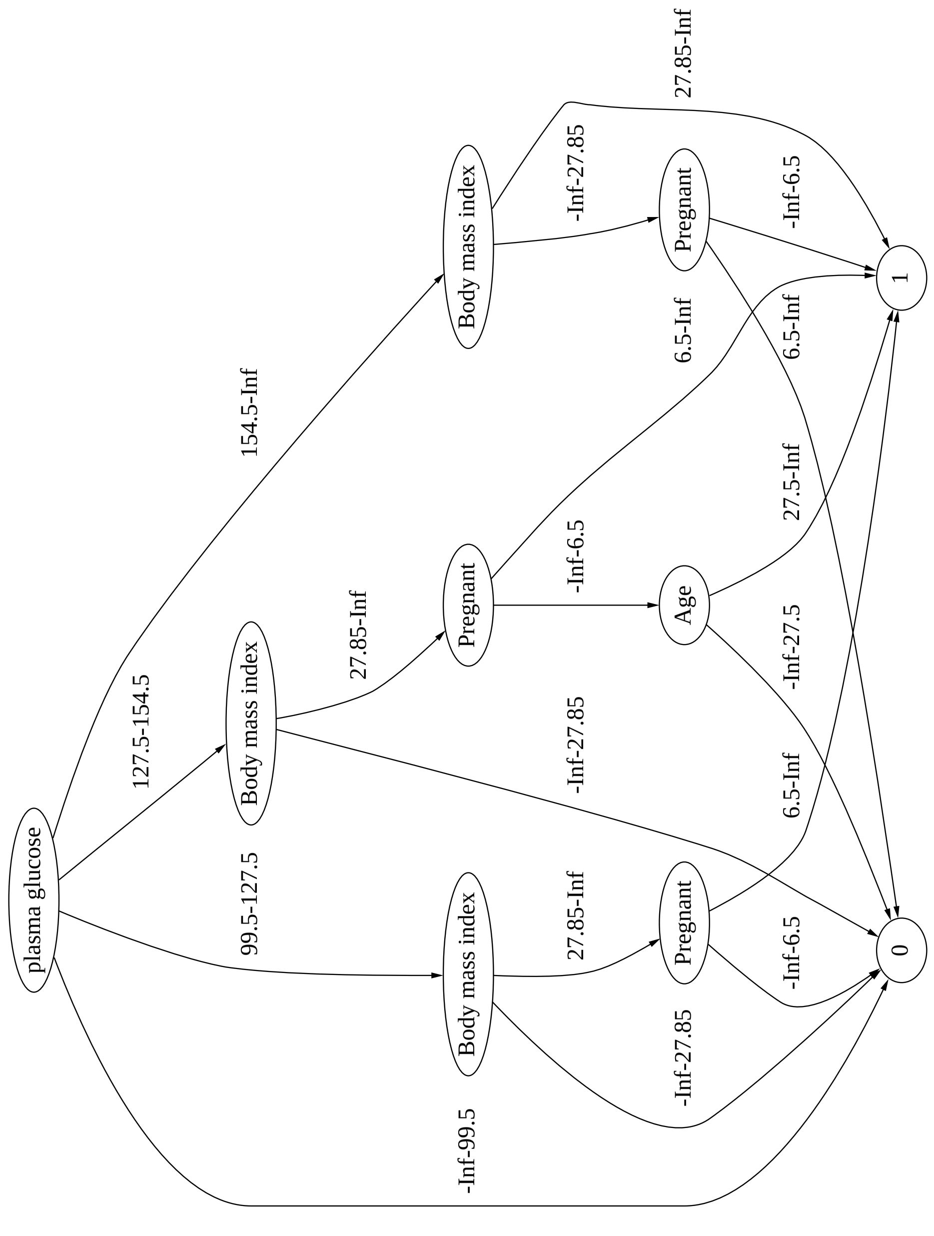

Figure D.10: The OODG generated by HOODG-Middle for Pima Indian diabetes.
Accuracy: 78.52%.
The discretization threshold for number of times pregnant was chosen to be 6.

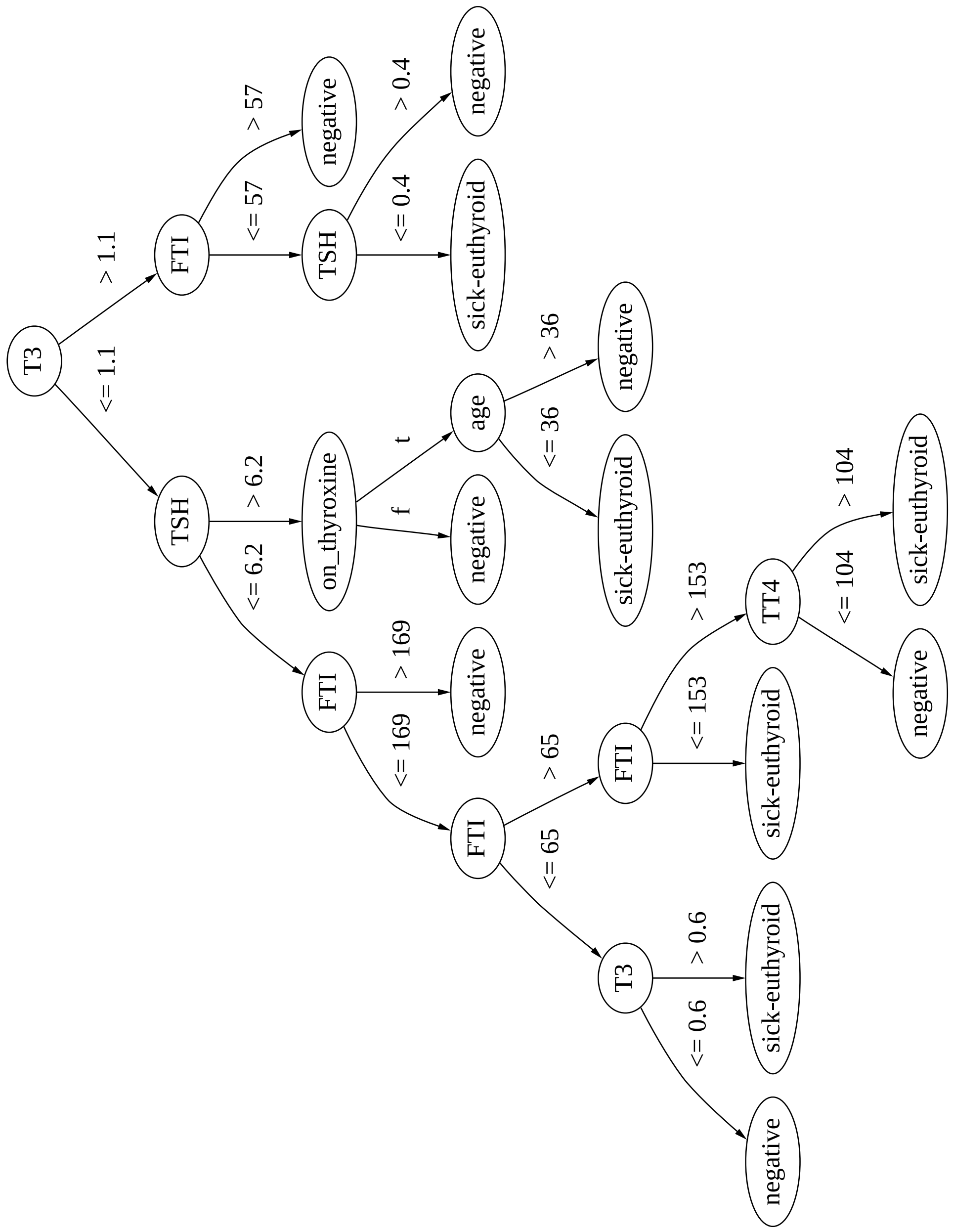

Figure D.11: The decision tree generated by C4.5 for the sick-euthyroid dataset. Accuracy: 97.84%.

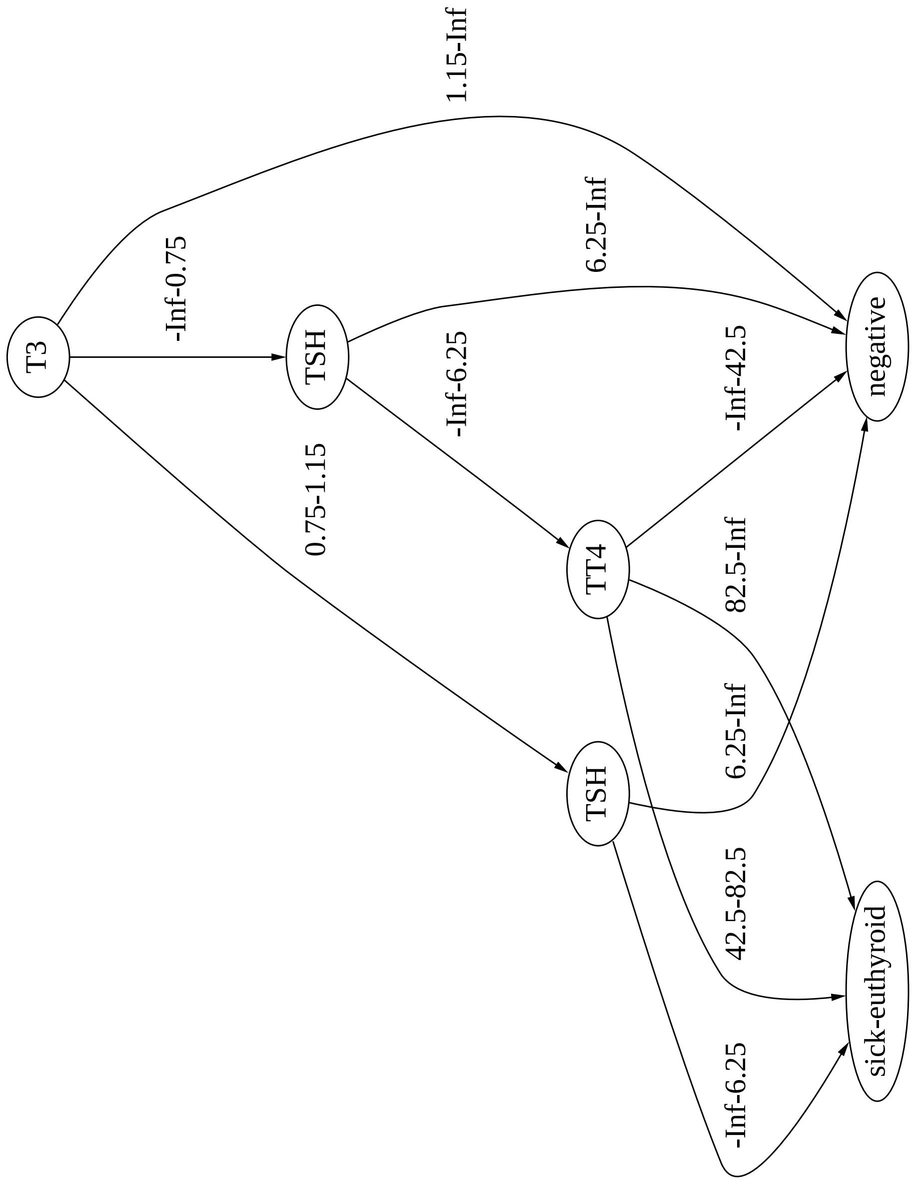

Figure D.12: The OODG generated by HOODG-Middle for the sick-euthyroid dataset. Accuracy: 97.07%.
Note that the graph does not use the feature "FTI", which was heavily used by C4.5. A closer inspection shows that the discretization algorithm discretized this feature into one interval, thus effectively removing it.

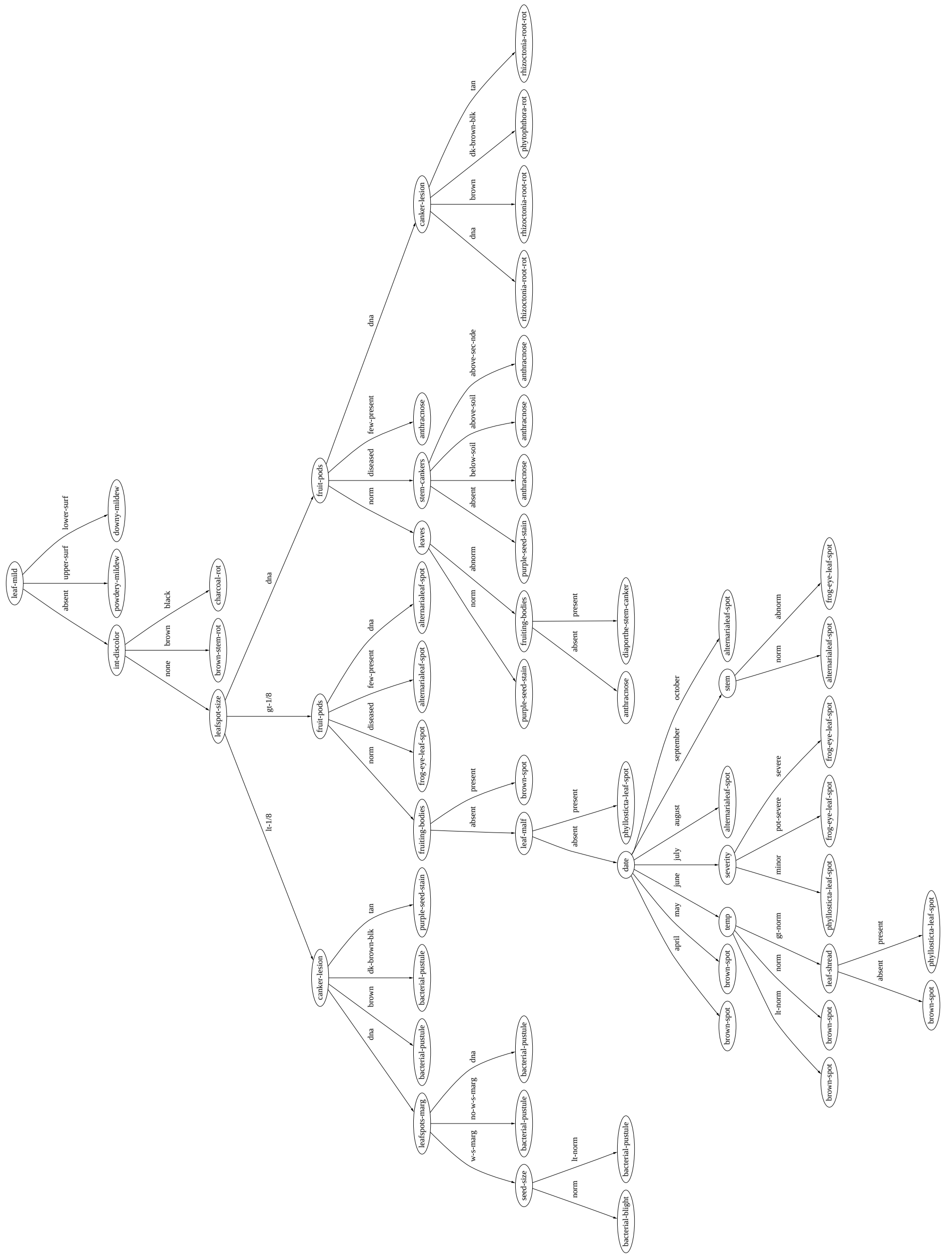

Figure D.13: The decision tree generated by C4.5 for soybean-large. Accuracy: 92.55%.

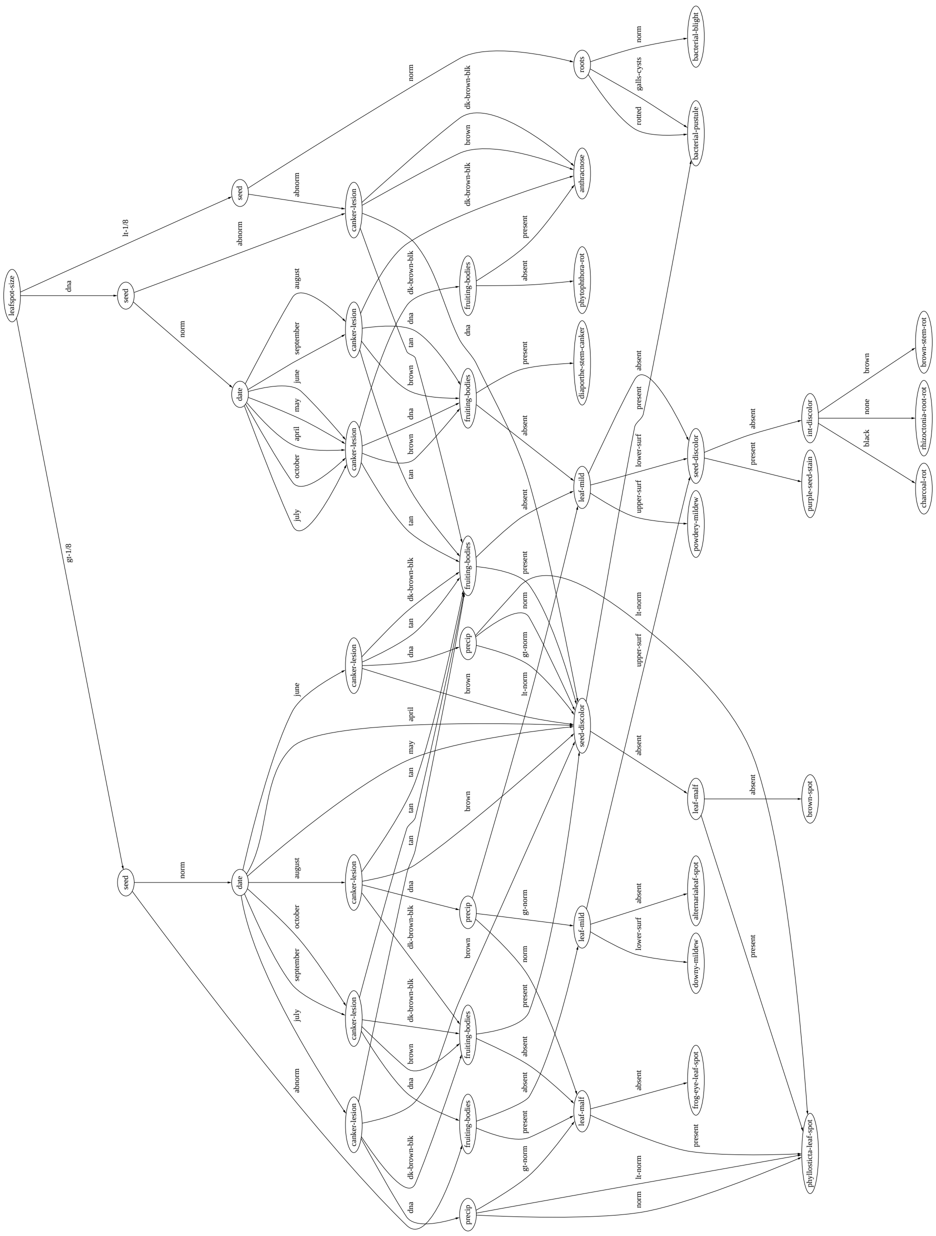

Figure D.14: The OODG generated by HOODG-Middle for soybean-large. Accuracy: 92.02%.

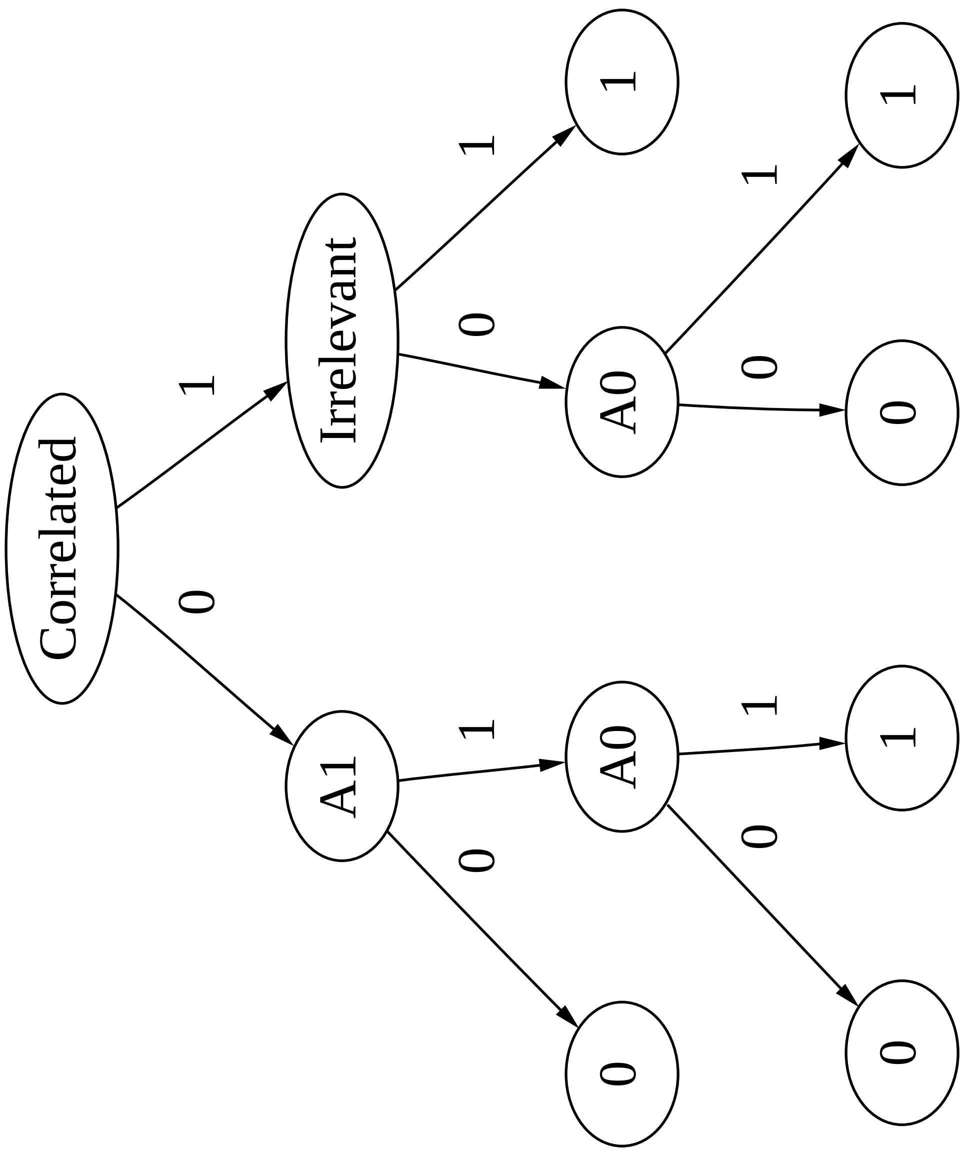

Figure D.15: The decision tree generated by C4.5 for corral.
Accuracy: 81.25%.
Note the split on the "irrelevant" feature and the root split on "correlated."

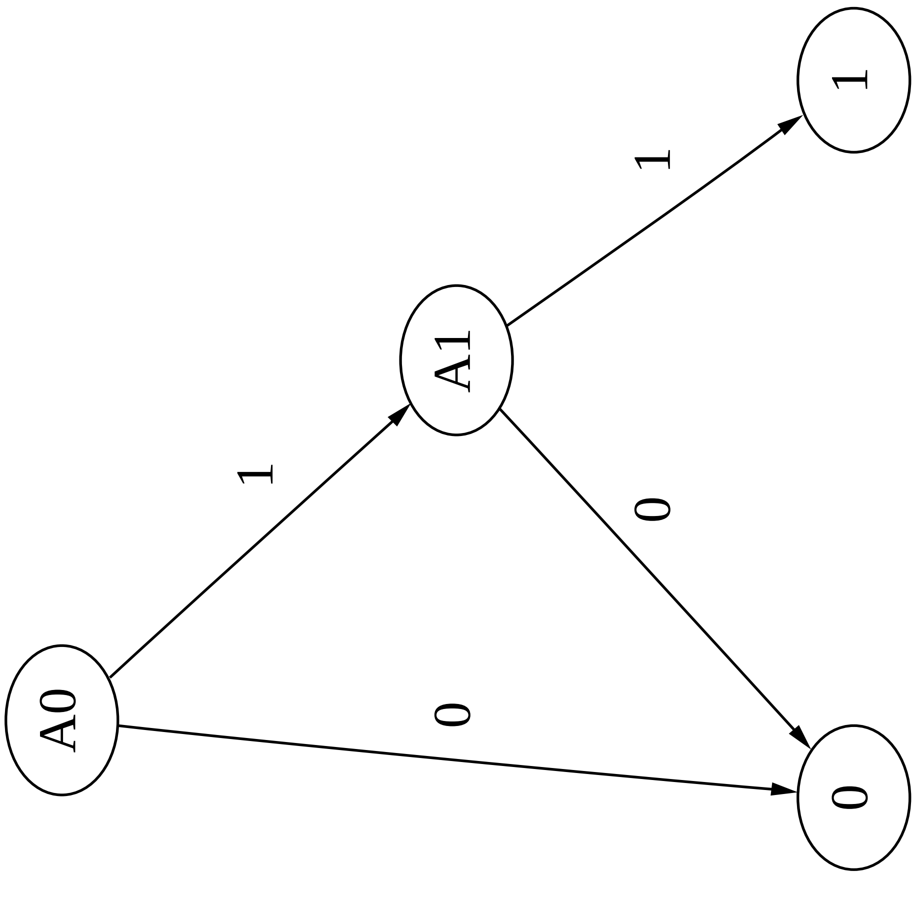

Figure D.16: The OODG generated by HOODG-Middle for corral.
Accuracy: 81.25%.
Simple, but not accurate enough. The wrapper did not find the correct set of features to use.

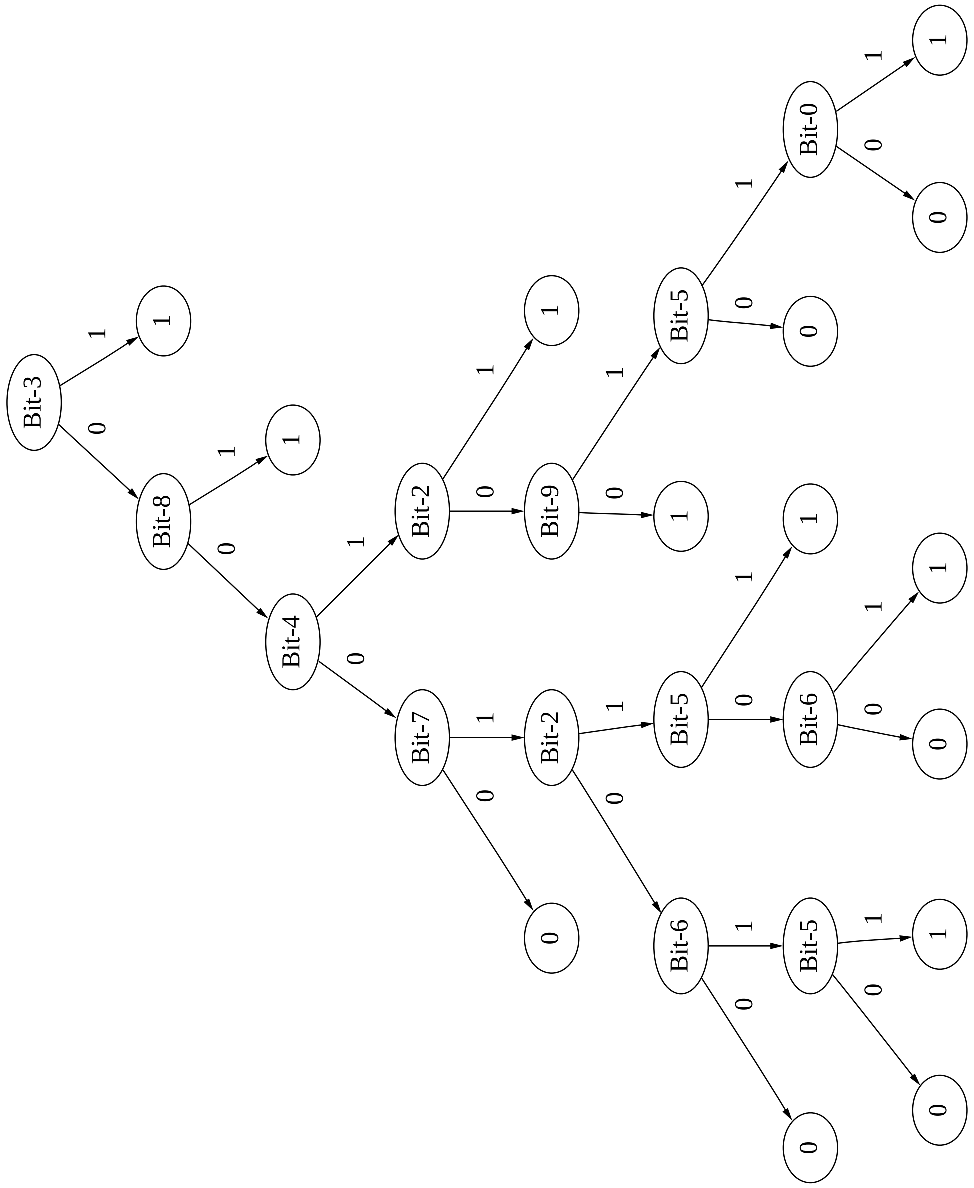

Figure D.17: The decision tree generated by C4.5 for m-of-n-3-7-10.
Accuracy: 85.55%.
Small but terrible.

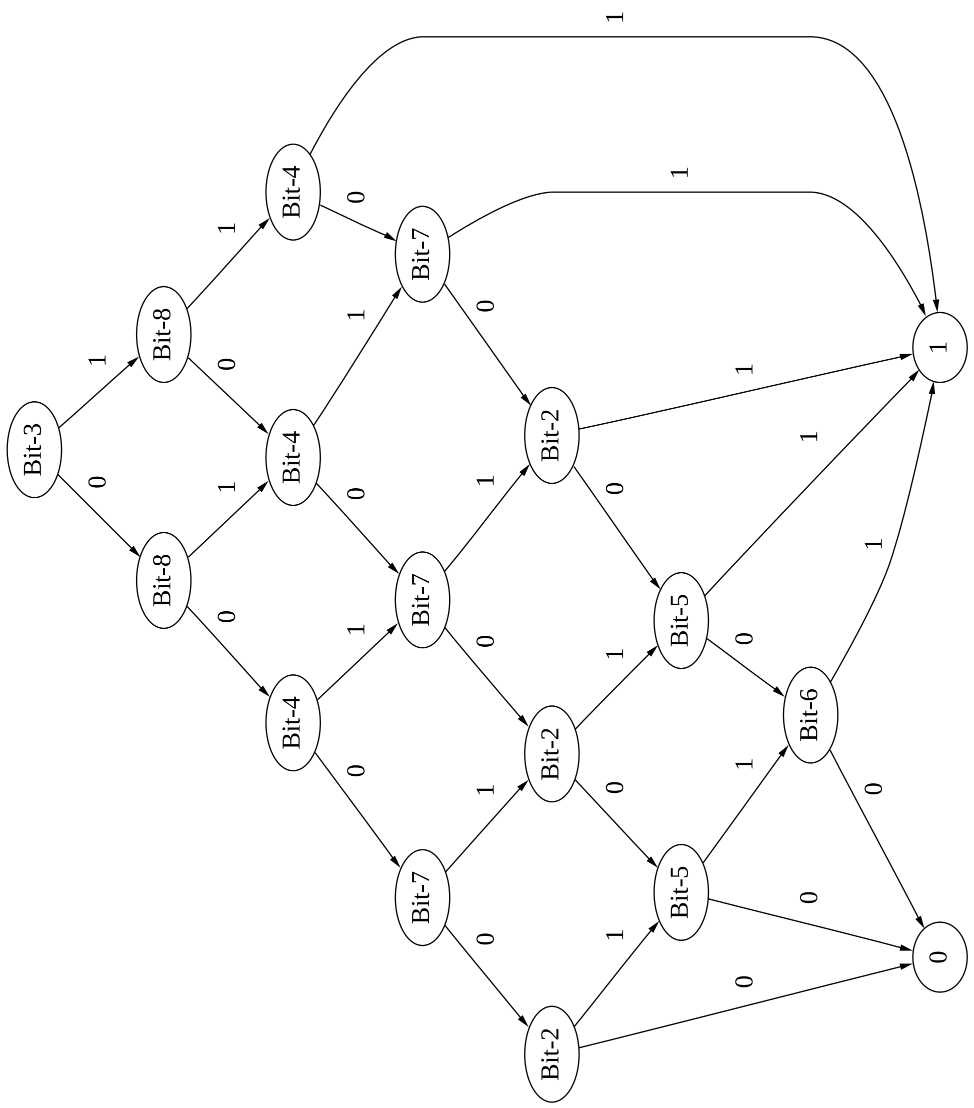

Figure D.18: The OODG generated by HOODG-Middle for m-of-n-3-7-10.
Accuracy: 100.00%.
A perfect graph! Note how the interior levels have three nodes that count the number of ones seen up to now. At the last branching level, only one node is needed because instances with zero or one bits set cannot lead to three bits sets given that we have one test left. Similarly, two levels above the leaves only two nodes are needed.

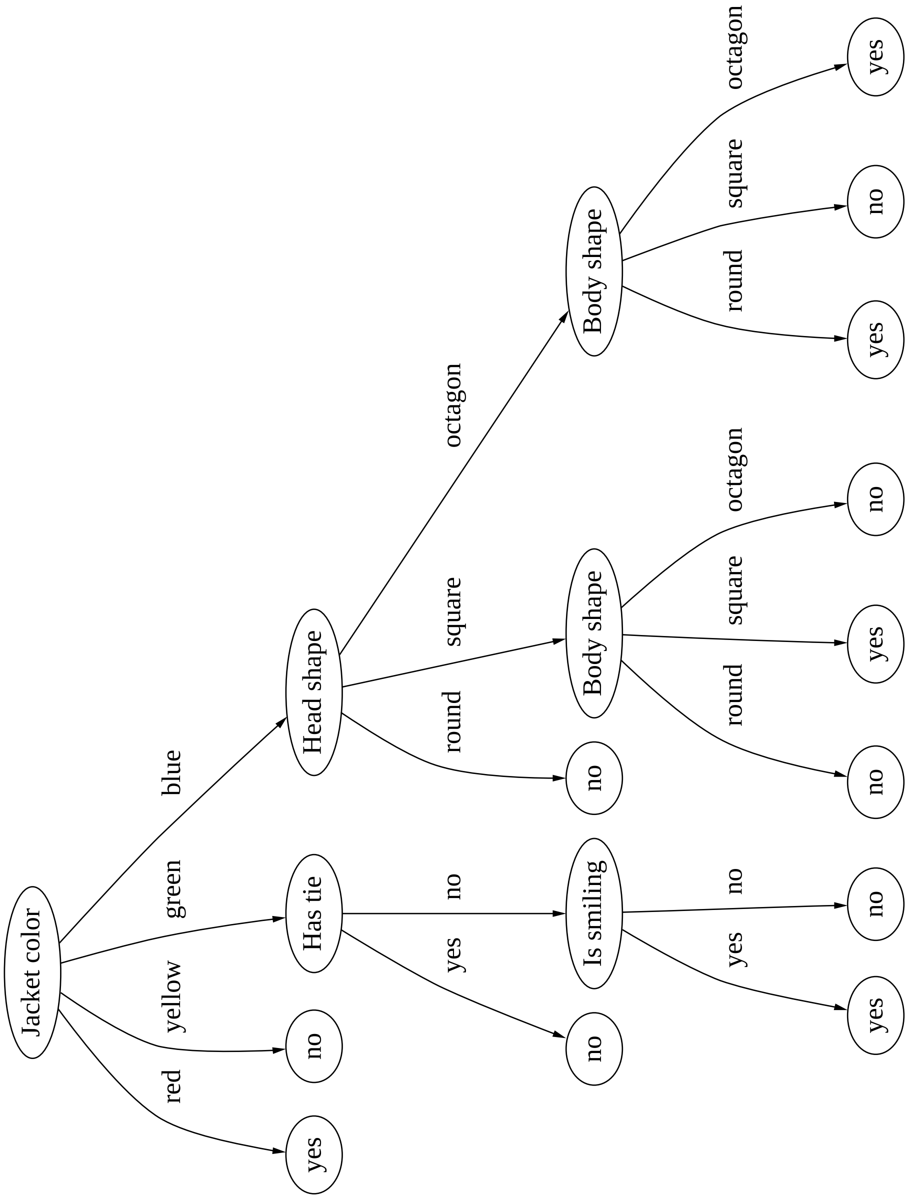

Figure D.19: The decision tree generated by C4.5 for Monk1.
Accuracy: 75.69%.
The test on "is smiling" is redundant. Pruning removes the subtree below "jacket-color = yellow."

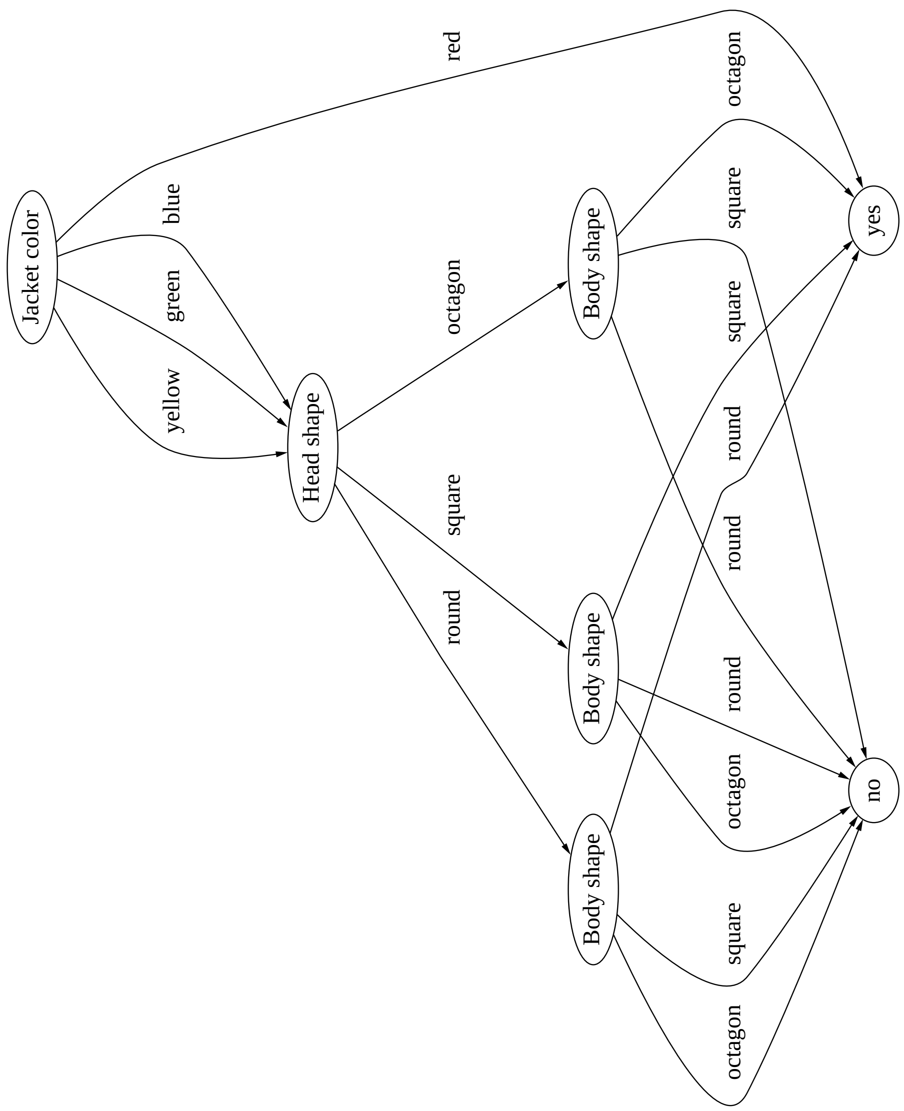

Figure D.20: The OODG generated by HOODG-Middle for Monk1.
Accuracy: 100.00%.
A perfect graph for "jacket-color = red or head-shape = body-shape."

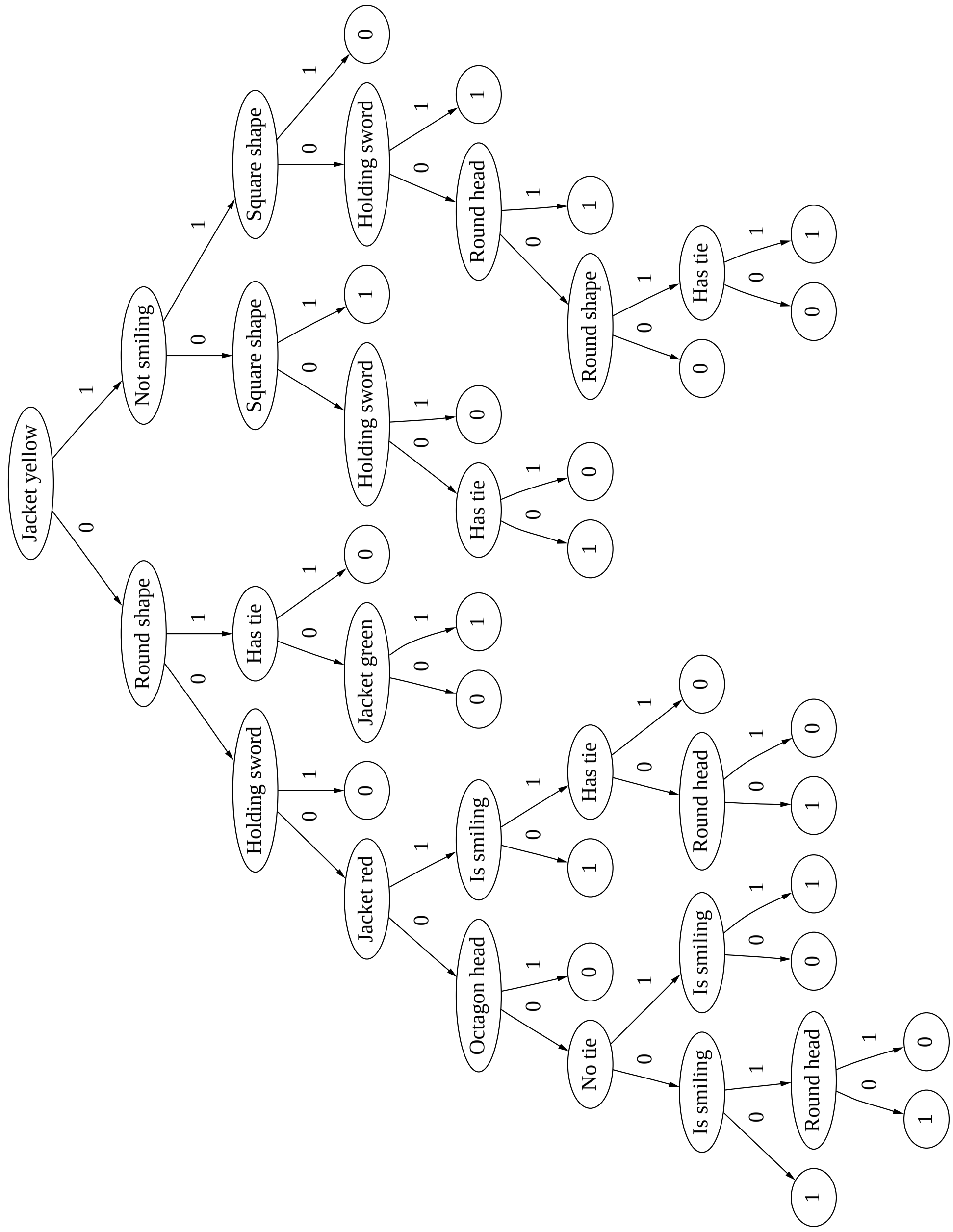

Figure D.21: The decision tree generated by C4.5 for Monk2-local.
Accuracy: 70.37%.

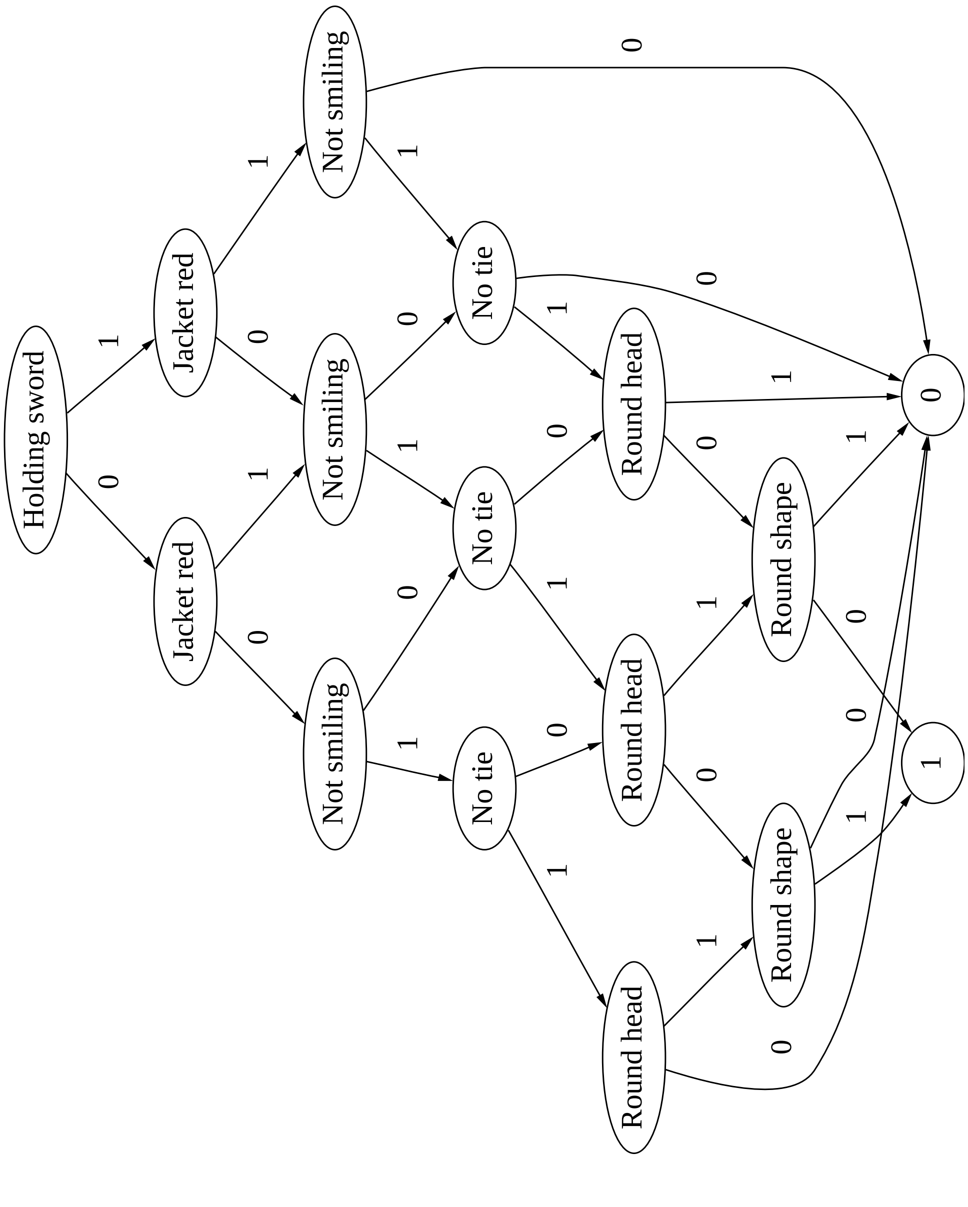

Figure D.22: The OODG generated by HOODG-Middle for Monk2-local.
Accuracy: 100.00%.
A perfect graph. The tests on "no smiling" and "no tie" could be replaced with "smiling" and "tie" and the edges reversed. This might make the concept slightly more comprehensible.

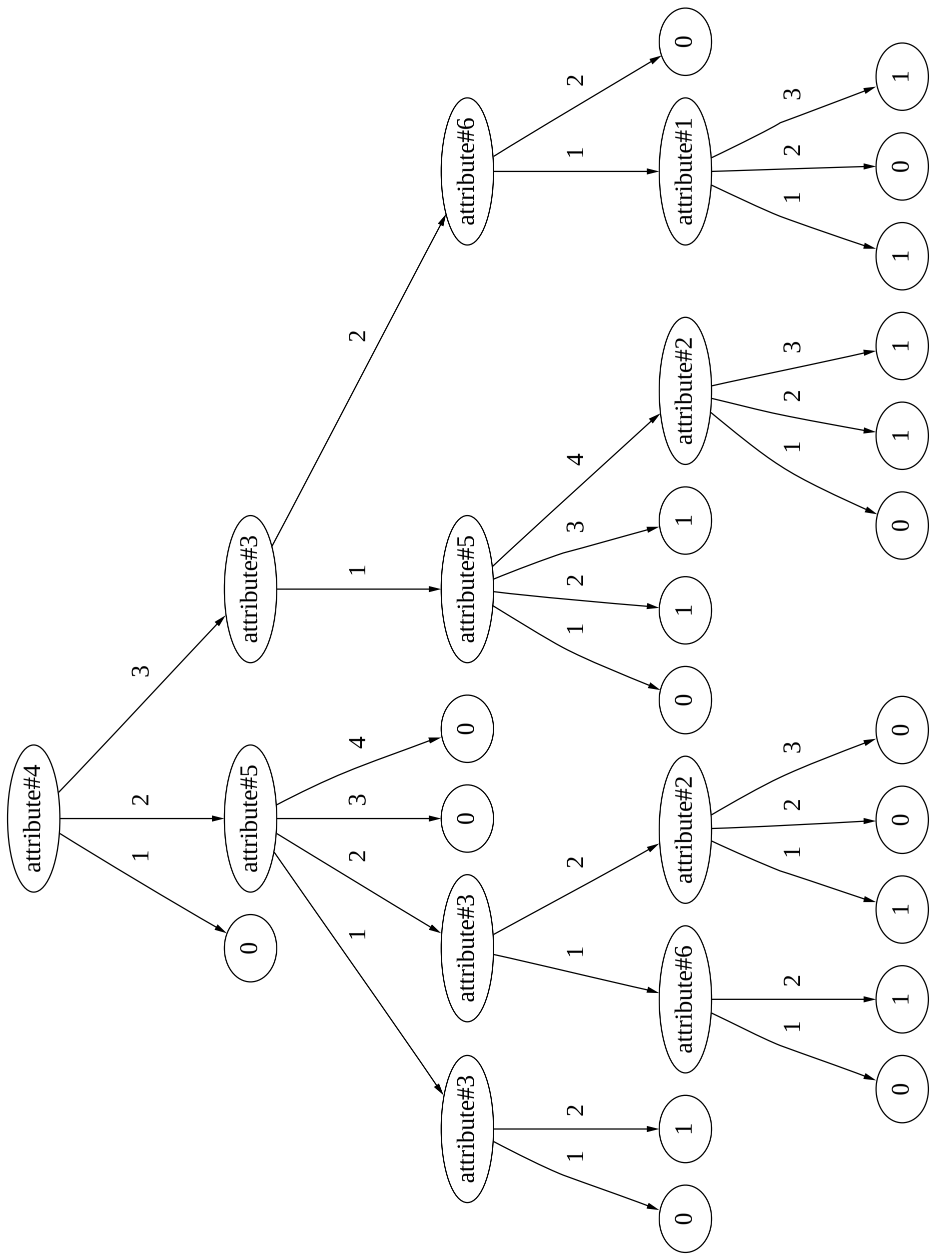

Figure D.23: The decision tree generated by C4.5 for Monk2.
Accuracy: 65.05%.
Lousy tree.

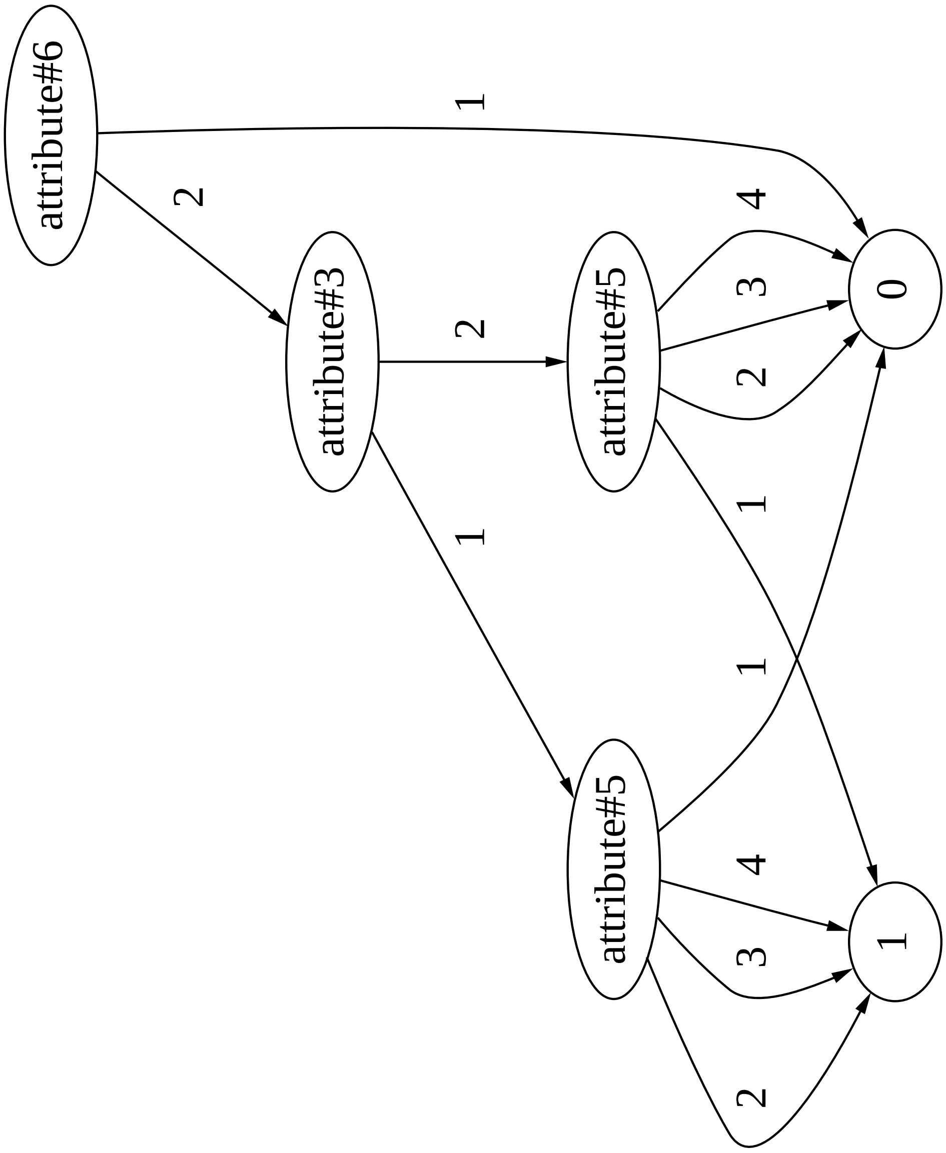

Figure D.24: The OODG generated by HOODG-Middle for Monk2.
Accuracy: 64.35%.
Lousy graph.

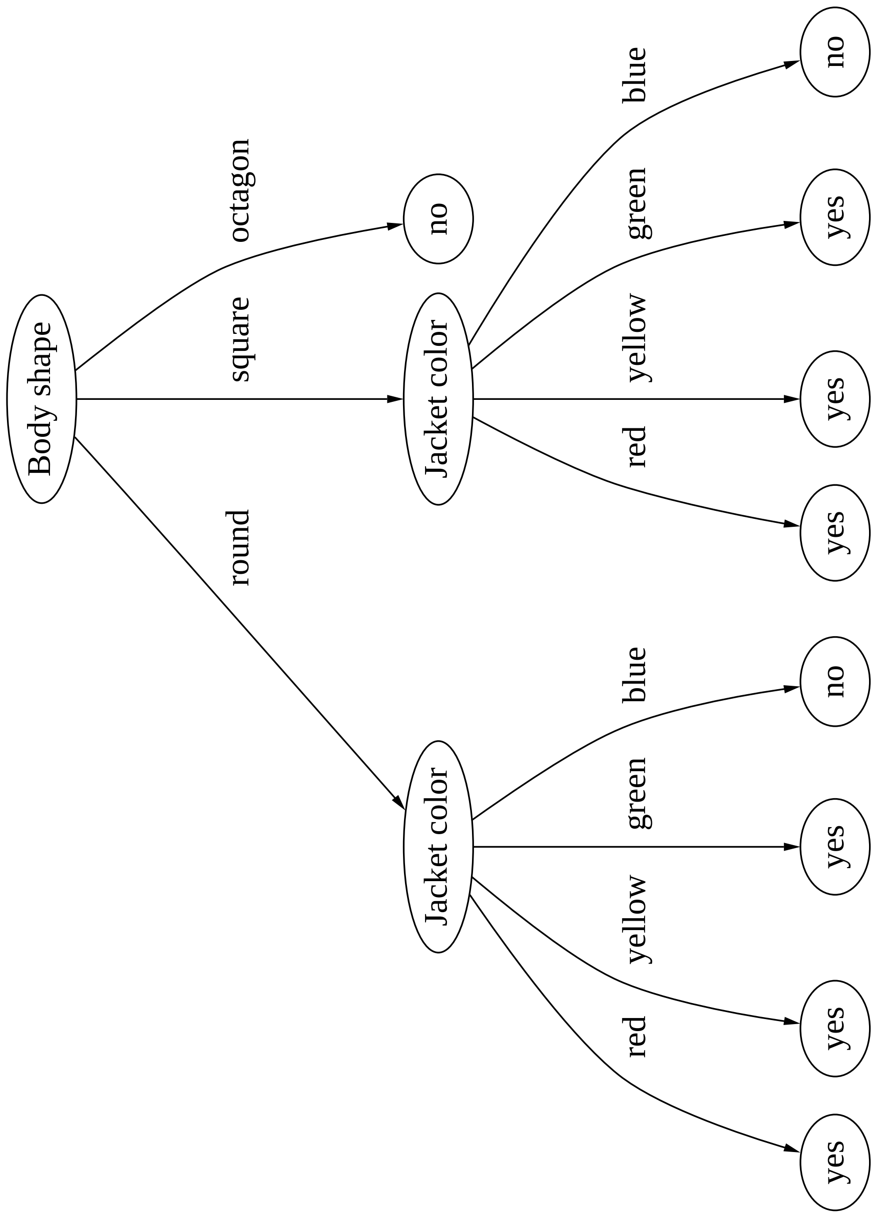

Figure D.25: The decision tree generated by C4.5 for Monk3.
Accuracy: 97.22%.

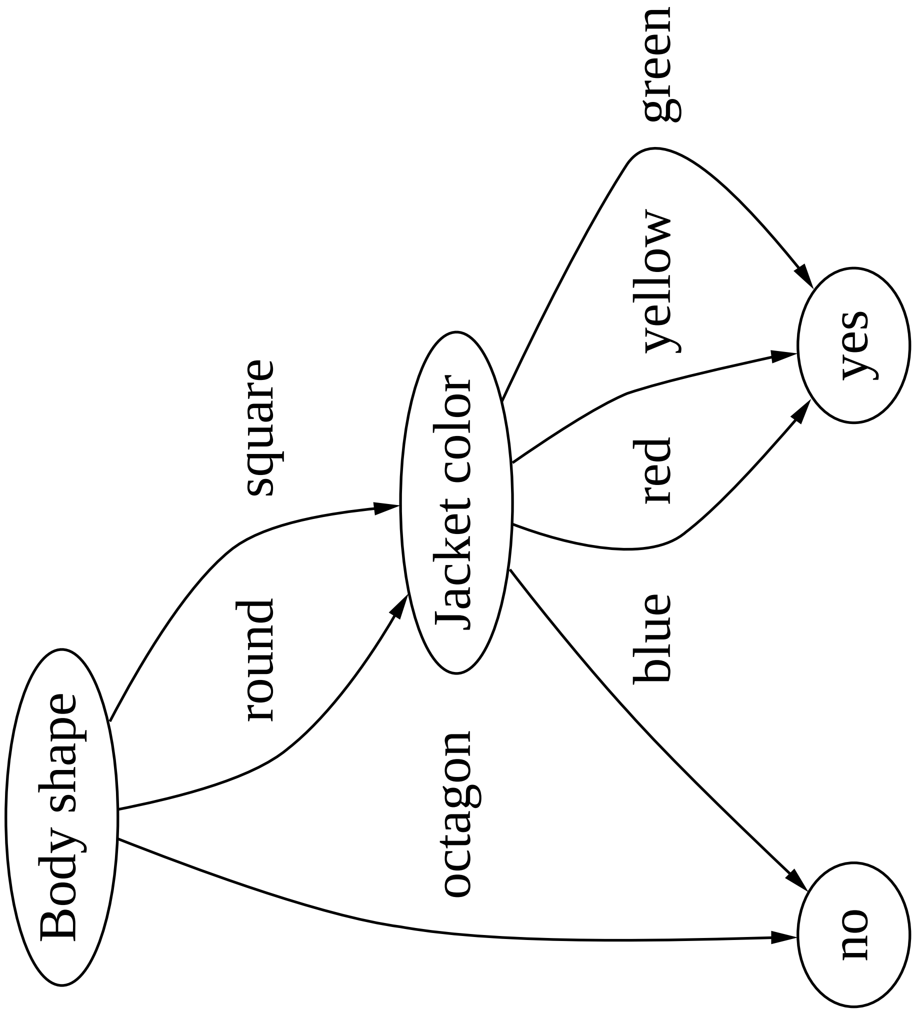

Figure D.26: The OODG generated by HOODG-Middle for Monk3.
Accuracy: 97.22%.
Exactly the same as the C4.5 tree, but compressed into a graph.

Appendix E

Global Comparison

This appendix compares all the main algorithms used in this dissertation and other publicly available algorithms. Instances with unknown values were removed, as in Chapter 6 on page 141 (horse-colic was eliminated because removing instances with unknowns left only one class). See Table 6.2 on page 176 for a description of the datasets and how the accuracy estimate was obtained (ten-fold cross-validation or a single test set). The following algorithms were compared:

Baseline A majority induction algorithm (ignoring all features).

C4.5 The C4.5 algorithm with default parameter settings.

C4.5-AP C4.5 with automatic parameter tuning as described in Section 4.9 on page 119.

C4.5-FSS C4.5 with a backwards best-first search wrapper with compound operators (Section 4.6).

C4.5-rules C4.5 in rule generation mode Quinlan (1993).

CN2 The CN2 V6.1 algorithm (Clark & Niblett 1989, Clark & Boswell 1991), which induces decision rules.

HOODG The HOODG-Middle algorithm described in Chapter 6 on page 141.

IB1,IB4 The instance based algorithms described in Aha (1992) (version from 3/9/94). The algorithms were executed with the storeall option as suggested by David Aha (personal communication).

IDTM The IDTM algorithm described in Chapter 5 on page 130.

NB Naive-Bayes with entropy discretization.

NB-FSS-back Naive-Bayes with entropy discretization and backward best-first search wrapper with compound operators as described in Section 4.5 on page 102.

OC1 The OC1 V3.0 algorithm (Murthy, Kasif, Salzberg & Beigel 1993, Murthy, Kasif & Salzberg 1994), which induces decision trees with oblique splits.

OneR The 1R induction algorithm described in Holte (1993). The implementation was done in $\mathcal{MLC}$++.

Table E.1: A global comparison for real-world datasets.

Algo/Dataset	breast cancer	cleve	crx	DNA	Pima	sick-euth	soy-bean	Average
Baseline	65.00	54.10	54.65	50.84	65.09	87.98	12.81	55.78
C4.5	95.32	74.98	84.08	92.66	71.60	97.84	92.54	87.00
C4.5-AP	95.32	76.70	86.07	92.41	70.18	97.91	91.93	87.22
C4.5-FSS	95.32	78.05	86.37	94.44	70.18	97.38	91.47	87.60
C4.5-rules	95.76	75.34	83.93	93.09	72.78	97.23	90.39	86.93
CN2	95.76	79.47	82.24	86.60	73.67	97.10	91.12	86.57
HOODG	95.32	82.80	85.31	94.10	74.47	97.07	91.47	88.65
IB1	96.07	76.07	81.46	76.60	70.03	88.80	90.76	82.83
IB4	96.94	78.12	86.67	78.40	73.93	87.80	90.24	84.59
IDTM	95.75	82.44	85.92	94.60	76.04	97.07	84.53	88.05
NB	97.10	86.67	87.88	93.34	71.43	94.14	91.23	88.83
NB-FSS	97.51	83.79	85.61	96.12	76.03	95.69	92.71	89.63
OC1	95.03	79.76	85.47	87.02	73.95	95.99	86.84	86.29
OneR	91.66	69.98	86.38	62.14	69.90	93.22	33.81	72.44

Table E.2: A global comparison for the artificial datasets.

Algo/Dataset	corral	m-of-n	Monk1	Monk2-local	Monk2	Monk3	Average
Baseline	56.25	77.34	50.00	67.13	67.13	47.22	60.85
C4.5	81.25	85.55	75.69	70.37	65.05	97.22	79.19
C4.5-AP	100.00	91.99	83.33	83.33	67.13	97.22	87.17
C4.5-FSS	81.25	85.16	88.89	88.43	67.13	97.22	84.68
C4.5-rules	81.25	88.28	91.67	66.20	66.20	96.30	81.65
CN2	100.00	90.50	98.60	74.50	75.70	90.70	88.33
HOODG	81.25	100.00	100.00	100.00	64.35	97.22	90.47
IB1	91.40	89.30	76.60	67.40	67.40	86.30	79.73
IB4	76.60	84.50	70.10	59.70	53.70	58.60	67.20
IDTM	100.00	77.34	100.00	100.00	64.35	97.22	89.82
NB	90.62	86.43	71.30	60.65	61.57	97.22	77.97
NB-FSS	90.62	87.50	72.22	67.13	67.13	97.22	80.30
OC1	89.06	99.22	91.20	84.95	96.30	94.21	92.49
OneR	75.00	77.34	75.00	67.13	67.13	80.56	73.69

Appendix F

Definition of Symbols

You've heard the definition of a drug: any substance which, when injected into a laboratory animal, produces a publication.
—Anonymous

Notation	Page defined	Description
$\mathtt{acc}$	17	The accuracy of a classifier. $$\mathtt{acc} = \mathtt{Pr}(\mathcal{C}(\vec{x}) = y)$$ The probability of correctly classifying a randomly selected instance $\vec{x}$.
$\mathtt{acc}_s$	38	The resubstitution estimate of accuracy.
$\mathtt{acc}_h$	39	The holdout estimate of accuracy.
$\mathtt{acc}_{\mathrm{CV}}$	42	The cross-validation estimate of accuracy.
$\mathcal{C}$	17	Classifier. A mapping from an unlabelled instance $\vec{x} \in \mathcal{X}$ to a label $y \in \mathcal{Y}$.
D	17	The distribution on $\mathcal{X} \times \mathcal{Y}$, the space of labelled instances.
$\mathcal{D}$	17	The dataset, a set of i.i.d. labelled instances from a distribution D.
$\mathrm{Dom}(X_i)$	17	The domain of feature X_i.

Notation	Page defined	Description
$\mathcal{I}$	17	An inducer, or induction algorithm. A mapping from a dataset $\mathcal{D}$ to a classifier $\mathcal{C}$.
$\mathcal{I}(\mathcal{D}, \vec{x})$	17	$(\mathcal{I}(\mathcal{D}))(\vec{x})$, *i.e.*, the label assigned to instance $\vec{x}$ by the classifier $\mathcal{I}(\mathcal{D})$.
m	17	The number of instances in the dataset.
n	17	The number of features in an instance.
$\mathcal{X}$	17	The space of unlabelled instances.
$\vec{X}$	17	An unlabelled instance.
$\vec{x}$	17	The feature values of an unlabelled instance.
X_i	17	A feature i.
x_i	17	A value of a feature X_i.
$\mathcal{Y}$	17	The set of possible label values.
Y	17	The label of an instance.
y	17	A label value.

Bibliography

It is somewhat disturbing to have one's work both unreferenced and dismissed as invalid.
—C. L. Mallows, discussion in Miller (1984)

AAA (1992), *Tenth National Conference on Artificial Intelligence*, MIT Press.

Aha, D. W. (1992), "Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms", *International Journal of Man-Machine Studies* **36**(1), 267–287.

Aha, D. W. & Bankert, R. L. (1994), Feature selection for case-based classification of cloud types: An empirical comparison, *in* "Working Notes of the AAAI-94 Workshop on Case-Based Reasoning", pp. 106–112.

Aha, D. W. & Bankert, R. L. (1995), A comparative evaluation of sequential feature selection algorithms, *in* D. Fisher & H. Lenz, eds, "Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics", Ft. Lauderdale, FL, pp. 1–7.

Aha, D. W., Kibler, D. & Albert, M. K. (1991), "Instance-based learning algorithms", *Machine Learning* **6**(1), 37–66.

Akers, S. B. (1978), "Binary decision diagrams", *IEEE Transactions on Computers* **C-27**(6), 509–516.

Almuallim, H. & Dietterich, T. G. (1991), Learning with many irrelevant features, *in* "Ninth National Conference on Artificial Intelligence", MIT Press, pp. 547–552.

Almuallim, H. & Dietterich, T. G. (1992*a*), Efficient algorithms for identifying relevant features, *in* "Proceedings of the Ninth Canadian Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 38–45.

Almuallim, H. & Dietterich, T. G. (1992*b*), On learning more concepts, *in* ML- (1992), pp. 11–19.

Almuallim, H. & Dietterich, T. G. (1994), "Learning boolean concepts in the presence of many irrelevant features", *Artificial Intelligence* **69**(1-2), 279–306.

Anderson, J. R. & Matessa, M. (1992), "Explorations of an incremental, bayesian algorithm for categorization", *Machine Learning* **9**, 275–308.

Andrews, D. F., Bickel, P. J., Hampel, F. R., Huber, P. J., Rogers, W. H. & Tukey, J. W. (1972), *Robust Estimates of Location*, Princeton University Press.

Angluin, D. (1992), Computational learning theory: Survey and selected bibliography, *in* "Proceedings of the 24th Annual ACM Symposium on the Theory of Computing", ACM Press, pp. 351–369.

Anscombe, F. J. (1967), "Topics in the investigation of linear relations fitted by the method of least squares", *Journal of the Royal Statistical Society B* **29**, 1–52.

Babcock, C. (1994), "Parallel processing mines retail data", *ComputerWorld*. 26 Sep 1994.

Bahl, L. R., Brown, P. F., de Souza, P. V. & Mercer, R. L. (1989), "A tree-based statistical language model for natural language speech recognition", *IEEE Transactions on Acoustics, Speech, and Signal Processing* **37**(7), 1001–1008.

Bai, C. (1988), Asymptotic properties of some samples reuse methods for prediction and classification, PhD thesis, Univesrity of California, San Diego.

Bailey, T. L. & Elkan, C. (1993), Estimating the accuracy of learned concepts, *in* "Proceedings of International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 895–900.

Barrington, D. A. (1989), "Bounded-width polynomial-size branching programs recognize exactly those languages in NC^1", *Journal of Computer and System Sciences* **38**(1), 150–164.

Ben-Bassat, M. (1982), Use of distance measures, information measures and error bounds in feature evaluation, *in* P. R. Krishnaiah & L. N. Kanal, eds, "Handbook of Statistics", Vol. 2, North-Holland Publishing Company, pp. 773–791.

Berliner, H. (1981), The B* tree search algorithm: A best-first proof procedure, *in* B. Webber & N. Nilsson, eds, "Readings in Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 79–87.

Binder, K. & Heerman, D. W. (1988), *Monte Carlo Simulation in Statistical Physics : an Introduction*, Springer-Verlag.

Blum, A. L. & Rivest, R. L. (1992), "Training a 3-node neural network is NP-complete", *Neural Networks* **5**, 117–127.

Blumer, A., Ehrenfeucht, A., Haussler, D. & Warmuth, M. K. (1987), "Occam's razor", *Information Processing Letters* **24**, 377–380.

Boddy, M. & Dean, T. (1989), Solving time-dependent planning problems, *in* N. S. Sridharan, ed., "Proceedings of the Eleventh International Joint Conference on Artificial Intelligence", Vol. 2, Morgan Kaufmann Publishers, Inc., pp. 979–984.

Bollig, B. & Wegener, I. (1994), Improving the variable ordering of OBDDs is NP-complete, Technical Report 542, Universität Dortmund.

Boole, G. (1854), *An investigation of the laws of thought, on which are founded the theories of logic and probabilities*, London, Walton and Maberly; Macmillan and Co. Reprinted by Dover Books, New York, 1954.

Boppana, R. B., & Sipser, M. (1990), The complexity of finite functions, *in* J. v. Leeuwen, ed., "Handbook of Theoretical Computer Science", MIT Press, chapter 14, pp. 758–804.

Brachman, R. J. (1987), "The myth of the one true logic", *Computational Intelligence* **3**, 168–172. Response to Drew McDermott's critique of pure reason.

Brazdil, P., Gama, J. & Henery, B. (1994), Characterizing the applicability of classification algorithms using meta-level learning, *in* F. Bergadano & L. D. Raedt, eds, "Proceedings of the European Conference on Machine Learning".

Breiman, L. (1994*a*), Bagging predictors, Technical Report Statistics Department, University of California at Berkeley.

Breiman, L. (1994*b*), Heuristics of instability in model selection, Technical Report Statistics Department, University of California at Berkeley.

Breiman, L. & Spector, P. (1992), "Submodel selection and evaluation in regression. the x-random case", *International Statistical Review* **60**(3), 291–319.

Breiman, L., Friedman, J. H., Olshen, R. A. & Stone, C. J. (1984), *Classification and Regression Trees*, Wadsworth International Group.

Breitbart, Y., Hunt III, H. B. & Rosenkrantz, D. (1991), The size of binary decision diagrams representing boolean functions, Submitted.

Bryant, R. E. (1986), "Graph-based algorithms for boolean function manipulation", *IEEE Transactions on Computers* **C-35**(8), 677–691.

Bryant, R. E. (1992), "Symbolic boolean manipulation with ordered binary-decision diagrams", *ACM Computing Surveys* **24**(3), 293–318.

Buchanan, B. & Smith, R. (1988), "Fundamentals of expert systems", *Annual Review of Computer Science*.

Buntine, W. (1992), "Learning classification trees", *Statistics and Computing* **2**, 63–73.

Burch, J. R., Clarke, E. M. & Long, D. E. (1991), Representing circuits more efficiently in symbolic model checking, *in* "Proceedings of the 28th ACM/IEEE Design Automation Conference", pp. 403–407.

Burch, J. R., Clarke, E. M., McMillan, K. L., Dill, D. L. & Hwang, L. J. (1990), Symbolic model checking: 10^{20} states and beyond, *in* "Fifth Annual IEEE Symposium on Logic in Computer Science.", IEEE Comput. Soc. Press, pp. 428–439.

Cardie, C. (1993), Using decision trees to improve case-based learning, *in* "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 25–32.

Caruana, R. & Freitag, D. (1994), Greedy attribute selection, *in* W. W. Cohen & H. Hirsh, eds, "Machine Learning: Proceedings of the Eleventh International Conference", Morgan Kaufmann Publishers, Inc.

Caruana, R. A. (1993), Multitask learning: A knowledge-based source of inductive bias, *in* "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 41–48.

Catlett, J. (1991*a*), Megainduction: machine learning on very large databases, PhD thesis, Univeristy of Sydney.
Available from http://www.research.att.com/orgs/ssr/people/catlett/phd.ps.Z.

Catlett, J. (1991*b*), On changing continuous attributes into ordered discrete attributes, *in* Y. Kodratoff, ed., "Proceedings of the European Working Session on Learning", Berlin, Germany: Springer-Verlag, pp. 164–178.

Cendrowska, J. (1987), "PRISM: an algorithm for inducing modular rules", *International Journal of Man-Machine Studies* **27**, 349–370.

Cestnik, B. (1990), Estimating probabilities: A crucial task in machine learning, *in* L. C. Aiello, ed., "Proceedings of the ninth European Conference on Artificial Intelligence", pp. 147–149.

Chakravarty, S. (1993), "A characterization of binary decision diagrams", *IEEE Transactions on Computers* **42**(2), 129–137.

Cheeseman *et al.* (1988), AutoClass: a bayesian classification system, *in* "Proceedings of the Fifth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 54–64. Also appears in Readings in *Machine Learning* by Shavlik and Dietterich.

Chou, P. A. (1988), Application of information theory to pattern recognition and the design of decision trees and trellises, PhD thesis, Stanford University.

Chou, P. A. (1991), "Optimal partitioning for classification and regression trees", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(4), 340–354.

Clark, P. & Boswell, R. (1991), Rule induction with CN2: Some recent improvements, *in* Y. Kodratoff, ed., "Proceedings of the fifth European conference (EWSL-91)", Springer Verlag, pp. 151–163.

Clark, P. & Matwin, S. (1993), Using qualitative models to guide inductive learning, *in* "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 45–56.

Clark, P. & Niblett, T. (1989), "The CN2 induction algorithm", *Machine Learning* **3**(4), 261–283.

Cormen, T. H., Leiserson, C. E. & Rivest, R. L. (1990), *Introduction to Algorithms*, McGraw-Hill.

Cortes, C., Jackel, L. D., Solla, S. A., Vapnik, V. & Denker, J. S. (1994), Learning curves: Asymptotic values and rate of convergence, *in* J. D. Cowan, G. Tesauro & J. Alspector, eds, "Advances in Neural Information Processing Systems", Vol. 6, Morgan Kaufmann Publishers, Inc., pp. 327–334.

Cover, T. M. & Campenhout, J. M. V. (1977), "On the possible orderings in the measurement selection problem", *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-7**(9), 657–661.

Craven, M. W. & Shavlik, J. W. (1993), Learning symbolic rules using artificial neural networks, *in* "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 73–80.

Crawford, S. L. (1989), "Extensions to the CART algorithm", *International Journal of Man-Machine Studies* **31**, 197–217.

Dasarathy, B. V. (1990), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, California.

De Mántaras, R. L. (1991), "A distance-based attribute selection measure for decision tree induction", *Machine Learning* **6**, 81–92.

desJardins, M. (1994), How to be a good graduate student and advisor, Available from marie@erg.sri.com.

Devijver, P. A. & Kittler, J. (1982), *Pattern Recognition: A Statistical Approach*, Prentice-Hall International.

Dietterich, T. G. (1986), "Learning at the knowledge level", *Machine Learning* **1**(3), 287–315. Reprinted in Shavlik and Dietterich (eds.) *Readings in Machine Learning.*

Dietterich, T. G. & Shavlik, J. W., eds (1990), *Readings in Machine Learning*, Morgan Kaufmann Publishers, Inc.

Dietterich, T., Hild, H. & Bakiri, G. (1995), "A comparison of id3 and backpropagation for english text-to-speech mapping", *Machine Learning* **18**(1), 51–80.

Doak, J. (1992), An evaluation of feature selection methods and their application to computer security, Technical Report CSE-92-18, University of California at Davis.

Dougherty, J., Kohavi, R. & Sahami, M. (1995), Supervised and unsupervised discretization of continuous features, *in* A. Prieditis & S. Russell, eds, "Machine Learning: Proceedings of the Twelfth International Conference", Morgan Kaufmann Publishers, Inc.

Draper, N. R. & Smith, H. (1981), *Applied Regression Analysis*, 2nd edn, John Wiley & Sons.

Duda, R. & Hart, P. (1973), *Pattern Classification and Scene Analysis*, Wiley.

Dvorak, V. (1992), An optimization technique for ordered (binary) decision diagrams, *in* P. Dewilde & J. Vandewalle, eds, "Compeuro Proceedings. Computer Systems and Software Engineering", IEEE Comput. Soc. Press, pp. 1–4.

Efron, B. (1979), "Bootstrap methods: another look at the jacknife", *Anals of Statistics* **7**(1), 1–26.

Efron, B. (1983), "Estimating the error rate of a prediction rule: improvement on cross-validation", *Journal of the American Statistical Association* **78**(382), 316–330.

Efron, B. & Tibshirani, R. (1991), "Statistical data analysis in the computer age", *Science* **253**, 390–395.

Efron, B. & Tibshirani, R. (1993), *An Introduction to the Bootstrap*, Chapman & Hall.

Efron, B. & Tibshirani, R. (1995), Cross-validation and the bootstrap: Estimating the error rate of a prediction rule, Technical Report 477, Stanford University.

Ergün, F., Kumar, S. R. & Rubinfeld, R. (1995), On learning bounded-width branching programs, *in* "Proceedings of the Eighth Annual Conference on Computational Learning Theory", ACM, Inc, pp. 361–368.

Esposito, F., Malerba, D. & Semeraro, G. (1995*a*), A further study of pruning methods in decision tree induction, *in* D. Fisher & H. Lenz, eds, "Proceedings of the fifth International Workshop on Artificial Intelligence and Statistics", pp. 211–218.

Esposito, F., Malerba, D. & Semeraro, G. (1995*b*), Simplifying decision trees by pruning and grafting: New results, *in* N. Lavrac & S. Wrobel, eds, "Machine Learning: ECML-95 (Proc. European Conf. on Machine Learning, 1995)", Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, Heidelberg, New York, pp. 287–290.

Fayyad, U. M. (1991), On the induction of decision trees for multiple concept learning, PhD thesis, EECS Dept, Michigan University.

Fayyad, U. M. & Irani, K. B. (1992), The attribute selection problem in decision tree generation, *in* AAA (1992), pp. 104–110.

Fayyad, U. M. & Irani, K. B. (1993), Multi-interval discretization of continuous-valued attributes for classification learning, *in* "Proceedings of the 13th International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 1022–1027.

Fayyad, U. M., Piatetsky-Shapiro, G. & Smyth, P. (to appear), From data mining to knowledge discovery: An overview, *in* U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth & R. Uthurusamy, eds, "Advances in Knowledge Discovery and Data Mining", AAAI/MIT Press, chapter 1.

Fayyad, U. M., Weir, N. & Djorgovski, S. (1993), SKICAT: a machine learning system for automated cataloging of large scale sky surveys, *in* "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 112–119. Longer version to appear in *Advances in Knowledge Discovery and Data Mining*, Fayyad, Piatetsky-Shapiro, Smyth, and Uthurusamy (Eds.).

Feigenbaum, E. A. (1977), The art of artificial intelligence: Themes and case studies of knowledge engineering, *in* "Proceedings of the 5th International Joint Conference on Artificial Intelligence", pp. 1014–1029.

Feigenbaum, E. A. (1988), Knowledge processing: From file servers to knowledge servers, *in* J. R. Quinlan, ed., "Applications of Expert Systems", Vol. 2, Turing Institute Press, chapter 1, pp. 3–11.

Feigenbaum, E. A., McCorduck, P. & Nii, H. P. (1988), *The Rise of the Expert Company*, Times Books.

Finnoff, W., Hergert, F. & Zimmermann, H. G. (1993), Improving model selection by nonconvergent methods, *in* "Neural Networks", Vol. 6, pp. 771–783.

Fisher, R. A. (1936), "The use of multiple measurements in taxonomic problems", *Annals of Eugenics* **7**(1), 179–188.

Freund, Y. (1990), Boosting a weak learning algorithm by majority, *in* "Proceedings of the Third Annual Workshop on Computational Learning Theory", pp. 202–216. To appear in Information and Computation.

Freund, Y. & Schapire, R. E. (1995), A decision-theoretic generalization of on-line learning and an application to boosting, *in* "Proceedings of the Second European Conference on Computational Learning Theory", Springer-Verlag, pp. 23–37.

Friedman, S. J. & Suppowit, K. J. (1987), Finding the optimal variable ordering for binary decision diagrams, *in* "Proceedings of the 24th ACM/IEEE Design Automation Conference", pp. 348–355.

Friedman, S. J. & Suppowit, K. J. (1990), "Finding the optimal variable ordering for binary decision diagrams", *IEEE Transactions on Computers* **39**(5), 710–713.

Fujita, M., Matsunaga, Y. & Kakuda, T. (1991), On variable ordering of binary decision diagrams for the application of multilevel logic synthesis, *in* "Proceedings of the European Conference on Design Automation", IEEE Computing Press, pp. 50–54.

Furnival, G. M. & Wilson, R. W. (1974), "Regression by leaps and bounds", *Technometrics* **16**(4), 499–511.

Gaines, B. R. (1991), The trade-off between knowledge and data in knowledge acquisition, *in* G. Piatetsky-Shapiro & W. Frawley, eds, "Knowledge Discovery in Databases", MIT Press, chapter 29, pp. 491–505.

Garey, M. R. (1972), "Optimal binary identification procedures", *Siam Journal on Applied Mathematics* **23**, 173–186.

Garey, M. R. & Johnson, D. S. (1979), *Computers and Intractability: a Guide to the Theory of NP-completeness*, W. H. Freeman and Company, San Francisco, CA.

Gavaldà, R. & Guijarro, D. (1995), Learning ordered binary decision diagrams, Technical Report NC-TR-95-050, Universitat Politècnica de Catalunya, Barcelona, Spain. NeuroCOLT Technical Report Series.

Geisser, S. (1975), "The predictive sample reuse method with applications", *Journal of the American Statistical Association* **70**(350), 320–328.

Geman, S., Bienenstock, E. & Doursat, R. (1992), "Neural networks and the bias/variance dilemma", *Neural Computation* **4**, 1–48.

Gennari, J. H., Langley, P. & Fisher, D. (1989), "Models of incremental concept formation", *Artificial Intelligence* **40**, 11–61.

Ginsberg, M. L. (1993), *Essentials of Artificial Intelligence*, Morgan Kaufmann Publishers, Inc.

Glick, N. (1978), "Additive estimators for probabilities of correct classification", *Pattern Recognition* **10**, 211–222.

Goldberg, D. E. (1989), *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley Publishing Company, Inc.

Gong, G. (1982), Cross validation, the jackknife, and the bootstrap: excess error estimation in forward logistic regression, PhD thesis, Stanford University. 3781-1982G.

Good, I. J. (1965), *The Estimation of Probabilities: An Essay on Modern Bayesian Methods*, M.I.T. Press.

Greiner, R. (1992), Probabilistic hill climbing : Theory and applications, *in* J. Glasgow & R. Hadley, eds, "Proceedings of the Ninth Canadian Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 60–67.

Hancock, T. R. (1989), On the difficulty of finding small consistent decision trees, Unpublished Manuscript, Harvard University.

Hardy, G. H. & Wright, E. M. (1979), *An introduction to the theory of numbers*, 5th edn, Oxford: claredon Press.

Harrison, D. (1993), "Backing up", *Network Computing* pp. 98–104. 15 Oct.

Hartmann, C. R. P., Varshney, P. K., Mehrotra, K. G. & Gerberich, C. L. (1982), "Application of information theory to the construction of efficient decision trees", *IEEE Transactions on information theory* **IT-28**(4), 565–577.

Haussler, D. (1988), "Quantifying inductive bias: AI learning algorithms and valiant's learning framework", *Artificial Intelligence* **36**(2), 177–221.

Haussler, D. (1992), "Decision theoretic generalizations of the PAC model for neural net and other learning applications", *Information and Computation* **100**(1), 78–150.

Haussler, D., Kearns, M., Seung, H. S. & Tishby, N. (1994), Rigorous learning curve bounds from statistical mechanics, *in* "Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory", ACM Press, pp. 76–87.

Hertz, J. A., Krogh, A. & Thorbergsson, G. I. (1989), "Phase transitions in simple learning", *Journal of Physics A* **22**(12), 2133–2150.

Hertz, J., Krogh, A. & Palmer, R. G. (1991), *Introduction to the Theory of Neural Computation*, Addison Wesley.

Hoaglin, D. C., Mosteller, F. & Tukey, J. W. (1983), *Understanding Robust and Exploratory Data Analysis*, John Wiley & Sons, Inc.

Hoeffding, W. (1963), "Probability inequalities for sums of bounded random variables", *Journal of the American Statistical Association* **58**, 13–30.

Holland, J. H. (1992), *Adaptation in natural and artificial systems : an introductory analysis with applications to biology, control, and artificial intelligence*, MIT Press.

Holte, R. C. (1993), "Very simple classification rules perform well on most commonly used datasets", *Machine Learning* **11**, 63–90.

Hyafil, L. & Rivest, R. L. (1976), "Constructing optimal binary decision trees is NP-complete", *Information Processing Letters* **5**(1), 15–17.

Imam, I. F. (1995), Driving Task-Oriented Decision Structures from Decision Rules, PhD thesis, School of Information Technology and Engineering, George Mason Univeristy.

Ishiura, N., Sawada, H. & Yajima, S. (1991), Minimization of binary decision diagrams based on exchanges of variables, *in* "IEEE International Conference on Computer-Aided Design. Digest of Technical Papers", IEEE Comput. Soc. Press, pp. 472–475.

Jain, A. K., Dubes, R. C. & Chen, C. (1987), "Bootstrap techniques for error estimation", *IEEE transactions on pattern analysis and machine intelligence* **PAMI-9**(5), 628–633.

Jelinek, F. (1985), The development of an experimental discrete dictation recognizer, *in* "Proceedings of the IEEE", Vol. 73, pp. 1616–1624.

John, G. H. (1994), Cross-validated C4.5: Using error estimation for automatic parameter selection, Technical Report STAN-CS-TN-94-12, Computer Science Department, Stanford University.
Available at `ftp://starry.Stanford.EDU/pub/gjohn/papers/cvc45.ps`.

John, G., Kohavi, R. & Pfleger, K. (1994), Irrelevant features and the subset selection problem, *in* "Machine Learning: Proceedings of the Eleventh International Conference", Morgan Kaufmann, pp. 121–129.

Judd, S. (1988), "On the complexity of loading shallow neural networks", *Journal of Complexity* **4**, 177–192.

Kadie, C. M. (1995), Seer: Maximum Likelihood Regression for Learning-Speed Curves, PhD thesis, University of Illinois at Urbana-Champaign.

Kadie, C. M. & Wilkins, D. C. (to appear), "Learning-speed curves: Their use in induction of classification expert systems", *International Journal of Knowledge Acquisition*.

Kaelbling, L. P. (1993), *Learning in Embedded Systems*, MIT Press.

Kira, K. & Rendell, L. A. (1992*a*), The feature selection problem: Traditional methods and a new algorithm, *in* AAA (1992), pp. 129–134.

Kira, K. & Rendell, L. A. (1992*b*), A practical approach to feature selection, *in* ML- (1992).

Kittler, J. (1978), Une généralisation de quelques algorithms sous-optimaux de recherche d'ensembles d'attributs, *in* "Proc. Congrés Reconnaissance des Formes et Traitement des Images".

Kittler, J. (1986), *Feature Selection and Extraction*, Academic Press, Inc, chapter 3, pp. 59–83.

Knoke, J. D. (1986), "The robust estimation of classification error rates", *Computers and Mathematics with Applications* **12A**(2), 253–260.

Knuth, D. E. (1973), *The Art of Computer Programming, fundamental algorithms*, Vol. 1, 2nd edn, Addison-Wesley Publishing Company.

Kohavi, R. (1994*a*), Bottom-up induction of oblivious, read-once decision graphs, *in* F. Bergadano & L. D. Raedt, eds, "Proceedings of the European Conference on Machine Learning", pp. 154–169.

Kohavi, R. (1994*b*), Bottom-up induction of oblivious, read-once decision graphs : strengths and limitations, *in* "Twelfth National Conference on Artificial Intelligence", pp. 613–618.

Kohavi, R. (1994*c*), Feature subset selection as search with probabilistic estimates, *in* "AAAI Fall Symposium on Relevance", pp. 122–126.

Kohavi, R. (1994*d*), A third dimension to rough sets, *in* "Third International Workshop on Rough Sets and Soft Computing", pp. 244–251. Also appeared in *Soft Computing* by Lin and Wildberger.

Kohavi, R. (1995*a*), The power of decision tables, *in* N. Lavrac & S. Wrobel, eds, "Proceedings of the European Conference on Machine Learning", Lecture Notes in Artificial Intelligence 914, Springer Verlag, Berlin, Heidelberg, New York, pp. 174–189.

Kohavi, R. (1995*b*), A study of cross-validation and bootstrap for accuracy estimation and model selection, *in* C. S. Mellish, ed., "Proceedings of the 14th International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc.

Kohavi, R. & Frasca, B. (1994), Useful feature subsets and rough set reducts, *in* "Third International Workshop on Rough Sets and Soft Computing", pp. 310–317. Also appeared in *Soft Computing* by Lin and Wildberger.

Kohavi, R. & John, G. (1995), Automatic parameter selection by minimizing estimated error, *in* A. Prieditis & S. Russell, eds, "Machine Learning: Proceedings of the Twelfth International Conference", Morgan Kaufmann Publishers, Inc.

Kohavi, R. & Li, C.-H. (1995), Oblivious decision trees, graphs, and top-down pruning, *in* C. S. Mellish, ed., "Proceedings of the 14th International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc.

Kohavi, R. & Sommerfield, D. (1995*a*), Feature subset selection using the wrapper model: Overfitting and dynamic search space topology, *in* "The First International Conference on Knowledge Discovery and Data Mining".

Kohavi, R. & Sommerfield, D. (1995*b*), MLC++ utilities,
Available in http://robotics.stanford.edu/users/ronnyk/mlc.html.

Kohavi, R., John, G., Long, R., Manley, D. & Pfleger, K. (1994), MLC++: A machine learning library in C++, *in* "Tools with Artificial Intelligence", IEEE Computer Society Press, pp. 740–743.
Available at `ftp://starry.Stanford.EDU/pub/ronnyk/mlc/toolsmlc.ps`.

Kononenko, I. (1993), "Inductive and bayesian learning in medical diagnosis", *Applied Artificial Intelligence* **7**, 317–337.

Kononenko, I. (1994), Estimating attributes: Analysis and extensions of Relief, *in* F. Bergadano & L. D. Raedt, eds, "Proceedings of the European Conference on Machine Learning".

Kononenko, I. (1995*a*), A counter example to the stronger version of the binary tree hypothesis, *in* "ECML-95 workshop on Statistics, machine learning, and knowledge discovery in databases", pp. 31–36.

Kononenko, I. (1995*b*), On biases in estimating multi-valued attributes, *in* C. S. Mellish, ed., "Proceedings of the 14th International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 1034–1040.

Koutsofios, E. & North, S. C. (1994), Drawing graphs with dot,
Available by anonymous ftp from `research.att.com:dist/drawdag/dotdoc.ps.Z`.

Koza, J. (1992), *Genetic Programming : On the Programming of Computers by Means of Natural Selection*, MIT Press.

Krause, M. & Waack, S. (1991), "On oblivious branching programs of linear length", *Information and Computation* **94**, 232–249.

Krishnaiah, P. R. & Kanal, L. N. (1982), *Classification, Pattern Recognition, and Reduction in Dimensionality*, Amsterdam: North Holland.

Krogh, A. & Vedelsby, J. (1995), Neural network ensembles, cross validation, and active learning, *in* "Advances in Neural Information Processing Systems", Vol. 7, MIT Press.

Kwok, S. W. & Carter, C. (1990), Multiple decision trees, *in* R. D. Schachter, T. S. Levitt, L. N. Kanal & J. F. Lemmer, eds, "Uncertainty in Artificial Intelligence", Elsevier Science Publishers, pp. 327–335.

Lachenbruch, P. A. (1967), "An almost unbiased method of obtaining confidence intervals for the probability of misclassification in discriminant analysis", *Biometrics* **23**, 639–645.

Lachenbruch, P. A. & Mickey, M. R. (1968), "Estimation of error rates in discriminant analysis", *Technometrics* **10**(1), 1–11.

Langley, P. (1994), Selection of relevant features in machine learning, *in* "AAAI Fall Symposium on Relevance", pp. 140–144.

Langley, P. (1995), *Elements of Machine Learning*, Morgan Kaufmann Publishers, Inc., San Francisco. In press.

Langley, P. & Sage, S. (1994*a*), Induction of selective bayesian classifiers, *in* "Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence", Morgan Kaufmann Publishers, Inc., Seattle, WA, pp. 399–406.

Langley, P. & Sage, S. (1994*b*), Oblivious decision trees and abstract cases, *in* "Working Notes of the AAAI-94 Workshop on Case-Based Reasoning", AAAI Press, Seattle.

Langley, P. & Simon, H. A. (to appear), "Applications of machine learning and rule induction", *Communications of the ACM*.

Langley, P., Iba, W. & Thompson, K. (1992), An analysis of bayesian classifiers, *in* "Proceedings of the tenth national conference on artificial intelligence", AAAI Press and MIT Press, pp. 223–228.

Lavrac, N. & Dzeroski, S. (1994), *Inductive logic programming : Techniques and Applications*, E. Horwood, New York.

Lee, C. Y. (1959), "Representation of switching circuits by binary-decision programs", *The Bell System Technical Journal* **38**(4), 985–999.

Lenat, D. B. & Feigenbaum, E. A. (1991), "On the thresholds of knowledge", *Artificial Intelligence* **47**, 185–250.

Linhart, H. & Zucchini, W. (1986), *Model Selection*, John Wiley & Sons.

Lund, C. & Yannakakis, M. (1993), On the hardness of approximating minimization problems, *in* "ACM Symposium on Theory of Computing".

Mallows, C. L. (1973), "Some comments on c_p", *Technometrics* **15**, 661–675.

Mammen, E. (1992), *When does bootstrap work? : asymptotic results and simulations*, Springer-Verlag.

Marill, T. & Green, D. M. (1963), "On the effectiveness of receptors in recognition systems", *IEEE Transactions on Information Theory* **9**, 11–17.

Maron, O. & Moore, A. W. (1994), Hoeffding races: Accelerating model selection search for classification and function approximation, *in* "Advances in Neural Information Processing Systems", Vol. 6, Morgan Kaufmann Publishers, Inc.

Marquardt, D. W. (1963), "An algorithm for least-squares estimation of nonlinear parameters", *Journal of the Society for Industrial and Applied Mathematics* **11**, 431–441.

Masek, W. J. (1976), A fast algorithm for the string editing problem and decision graph complexity, Master's thesis, Massachusetts Institute of Technology.

McLachlan, G. J. (1976), "The bias of the apparent error rate in discriminant analysis", *Biometrika* **63**, 239–244.

McLachlan, G. J. (1992), *Discriminant Analysis and Statistical Pattern Recognition*, Wiley-Interscience.

Mehta, M., Rissanen, J. & Agrawal, R. (1995), MDL-based decision tree pruning, *in* U. M. Fayyad & R. Uthurusamy, eds, "Proceedings of the first international conference on knowledge discovery and data mining", AAAI Press, pp. 216–221.

Meinel, C. (1992), "Branching programs — an efficient data structure for computer-aided circuit design", *Bulletin of the European Association For Theoretical Computer Science* **46**, 149–170.

Meinel, C., Krause, M. & Waack, S. (1988), Separating the eraser turing machine classes $\mathcal{L}_e$, $\mathcal{NL}_e$, co-$\mathcal{NL}_e$, and $\mathcal{P}_e$, *in* "Mathematical Foundations of Computer Science", Lecture Notes in Computer Science, Vol 324, Springer-Verlag, pp. 405–409.

Michalski, R. S. (1978), A planar geometric model for representing multidimensional discrete spaces and multiple-valued logic functions, Technical Report UIUCDCS-R-78-897, University of Illinois at Urbaba-Champaign.

Michalski, R. S. & Imam, I. F. (1994), Learning problem-oriented decision structures from decision rules: The aqdt-2 system, *in* Z. W. Ras & M. Zemankova, eds, "Proceedings of the 8th International Symposium on Methodology for Intelligent Systems (ISMIS-94)", Lecture Notes in Artificial Intelligence 914, Springer Verlag, pp. 416–426.

Michalski, R. S., Mozetic, I., Hong, J. & Lavrac, N. (1986), The multipurpose incremental learning system AQ15 and its testing application to three medical domains, *in* "Proceedings of the Fifth National Conference on Artificial Intelligence", Morgan Kaufmann, pp. 1041–1045.

Michie, D. (1987), Current developments in expert systems, *in* J. R. Quinlan, ed., "Applications of Expert Systems", Vol. 1, Turing Institute Press, chapter 8, pp. 137–156.

Miller, A. J. (1984), "Selection of subsets of regression variables", *Royal Statistical Society A* **147**, 389–425.

Miller, A. J. (1990), *Subset Selection in Regression*, Chapman and Hall.

Minato, S. (1992), "Minimum-width method of variable ordering for binary decision diagrams", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* **E75-A**(3), 392–399.

Minato, S., Ishiura, N. & Yajima, S. (1990), Shared binary decision diagram with attributed edges for efficient boolean function manipulation, *in* "Proceedings of the 27th ACM/IEEE Design Automation Conference", pp. 24–28.

Minsky, M. L. & Papert, S. (1988), *Perceptrons : an Introduction to Computational Geometry*, MIT Press. Expanded ed.

Mitchell, T. M. (1982), "Generalization as search", *Artificial Intelligence* **18**, 203–226. Reprinted in Shavlik and Dietterich (eds.) *Readings in Machine Learning*.

Mitchell, T. M. & Thrun, S. B. (1993), Explanation based learning: A comparison of symbolic and nerual network approaches, *in* "Proceedings of the Tenth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 197–204.

Mitchell, T. M., Keller, R. M. & Kedar-Cabelli, S. T. (1986), "Explanation-based generalization: A unifying view", *Machine Learning* **1**(1), 47–80.

ML- (1992), *Proceedings of the Ninth International Conference on Machine Learning*, Morgan Kaufmann Publishers, Inc.

Mladenić, D. (1995), Automated model selection, *in* "ECML workshop on Knowledge Level Modeling and Machine Learning".

Modrzejewski, M. (1993), Feature selection using rough sets theory, *in* P. B. Brazdil, ed., "Proceedings of the European Conference on Machine Learning", Springer, pp. 213–226.

Moore, A. W. & Lee, M. S. (1994), Efficient algorithms for minimizing cross validation error, *in* W. W. Cohen & H. Hirsh, eds, "Machine Learning: Proceedings of the Eleventh International Conference", Morgan Kaufmann Publishers, Inc.

Moore, A. W., Hill, D. J. & Johnson, M. P. (1992), An empirical investigation of brute force to choose features, smoothers and function approximators, *in* S. Hanson, S. Judd & T. Petsche, eds, "Computational Learning Theory and Natural Learning Systems Conference", Vol. 3, MIT Press.

Moret, B. M. E. (1982), "Decision trees and diagrams", *ACM Computing Surveys* **14**(4), 593–623.

Mosteller, F. & Tukey, J. W. (1968), Data analysis, including statistics, *in* G. Lindzey & E. Aronson, eds, "Handbook of Social Psychology", Vol. 2, Addison Wesley.

Muggleton, S. H. (1990), *Inductive acquisition of expert knowledge*, Addison-Wesley, Wokingham, England.

Murphy, P. M. & Aha, D. W. (1995), UCI repository of machine learning databases, http://www.ics.uci.edu/~mlearn/MLRepository.html.

Murthy, S. K., Kasif, S. & Salzberg, S. (1994), "A system for the induction of oblique decision trees", *Journal of Artificial Intelligence Research* **2**, 1–33.

Murthy, S., Kasif, S., Salzberg, S. & Beigel, R. (1993), OC1: Randomized induction of oblique decision trees, *in* "Eleventh National Conference on Artificial Intelligence", MIT Press, pp. 322–327.

Naeher, S. (1995), *LEDA: A Library of Efficient Data Types and Algorithms*, 3.2 edn, Max-Planck-Institut fuer Informatik, IM Stadtwald, D-66123 Saarbruecken, FRG.
Available by anonymous ftp in `ftp.mpi-sb.mpg.de`.

Narendra, M. P. & Fukunaga, K. (1977), "A branch and bound algorithm for feature subset selection", *IEEE Transactions on Computers* **C-26**(9), 917–922.

Neter, J., Wasserman, W. & Kutner, M. H. (1990), *Applied Linear Statistical Models*, 3rd edn, Irwin: Homewood, IL.

Newell, A. (1982), "The knowledge level", *Artificial Intelligence* **18**, 87–127. Originally appeared in the *AI Magazine*, 2(2), 1981.

Nilsson, N. J. (1990), *The Mathematical Foundations of Learning Machines*, Morgan Kaufmann Publishers, Inc. Previously published as: Learning Machines, 1965.

O'Kane, D. (1994), "Learning to classify in large committee machines", *Physical Review E* **50**(4), 3201–3209.

Oliveira, A. L. & Sangiovanni-Vincentelli, A. (1995), Inferring reduced ordered decision graphs of minimum description length, *in* A. Prieditis & S. Russell, eds, "Machine Learning: Proceedings of the Twelfth International Conference", Morgan Kaufmann Publishers, Inc., pp. 421–429.

Oliver, J., Dowe, D. & Wallace, C. (1992), Inferring decision graphs using the minimum message length principle, *in* A. Adams & L. Sterling, eds, "Proceedings of the 5th Australian Joint Conference on Artificial Intelligence", World Scientific, Singapore, pp. 361–367.

Oliver, J. J. (1993), Decision graphs—an extension of decision trees, *in* "Proceedings of the fourth International workshop on Artificial Intelligence and Statistics", pp. 343–350.

Olshen, R. A., Gilpin, E. A., Henning, H., LeWinter, M. L., Collins, D. & Ross, J. (1985), Twelve-month prognosis following myocardial infarction: Classification trees, logistic regression, and stepwise linear discrimination, *in* L. L. Cam, R. Olshen & C. Chin-Shiu, eds, "Proceedings of the Berkeley conference in honor of Jerzy Neyman and Jack Kiefer", Vol. 1, Wadsworth Advanced Books & Software.

Pagallo, G. & Haussler, D. (1990), "Boolean feature discovery in empirical learning", *Machine Learning* **5**, 71–99.

Pawlak, Z. (1987), "Decision tables — a rough sets approach", *Bull. of EATCS* **33**, 85–96.

Pawlak, Z. (1991), *Rough Sets*, Kluwer Academic Publishers.

Pawlak, Z. (1993), "Rough sets: present state and the future", *Foundations of Computing and Decision Sciences* **18**(3-4), 157–166.

Pawlak, Z., Wong, S. & Ziarko, W. (1988), "Rough sets: Probabilistic versus deterministic approach", *Internation Journal of Man Machine Studies* **29**, 81–95.

Pazzani, M. J. (1995), Searching for dependencies in bayesian classifiers, *in* D. Fisher & H. Lenz, eds, "Proceedings of the fifth International Workshop on Artificial Intelligence and Statistics", Ft. Lauderdale, FL.

Perrone, M. (1993), Improving regression estimation: averaging methods for variance reduction with extensions to general convex measure optimization, PhD thesis, Brown University, Physics Dept.

Pimat, V., Kononenko, I., Janc, T. & Bratko, I. (1989), Medical estimation of automatically induced decision rules, *in* "Proceedings of the 2nd European conference on Artificial Intelligence in Medicine", pp. 24–36.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992), *Numerical Recipes in C: The Art of Scientific Computing*, second edn, Cambridge University Press.

Provan, G. M. & Singh, M. (1995), Learning bayesian networks using feature selection, *in* D. Fisher & H. Lenz, eds, "Proceedings of the fifth International Workshop on Artificial Intelligence and Statistics", Ft. Lauderdale, FL, pp. 450–456.

Quinlan, J. R. (1986), "Induction of decision trees", *Machine Learning* **1**, 81–106. Reprinted in Shavlik and Dietterich (eds.) *Readings in Machine Learning*.

Quinlan, J. R. (1987), "Simplifyng decision trees", *International Journal of Man-Machine Studies* **27**, 221–234.

Quinlan, J. R. (1988), An empirical comparison of genetic and decision-tree classifiers, *in* "Proceedings of the Fifth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 135–141.

Quinlan, J. R. (1989), Unknown attribute values in induction, *in* "Proceedings of the Sixth International Machine Learning Workshop", Morgan Kaufmann, pp. 164–168.

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann Publishers, Inc., Los Altos, California.

Quinlan, J. R. (1994), Comparing connectionist and symbolic learning methods, *in* S. J. Hanson, G. A. Drastal & R. L. Rivest, eds, "Computational Learning Theory and Natural Learning Systems", Vol. I: Constraints and Prospects, MIT Press, chapter 15, pp. 445—456.

Quinlan, J. R. & Rivest, R. L. (1989), "Inferring decision trees using the minimum description length principle", *Information and Computation* **80**, 227–248.

Raghavan, V. & Wilkins, D. (1993), Learning μ-branching programs with queries, *in* "Proceedings of the Sixth Annual Workshop on Computational Learning Theory", ACM Press, New York, NY, pp. 27–36.

Reinwald, L. T. & Soland, R. M. (1966), "Conversion of limited-entry decision tables to optimal computer programs i: Minimum average processing time", *Journal of the ACM* **13**(3), 339–358.

Reinwald, L. T. & Soland, R. M. (1967), "Conversion of limited-entry decision tables to optimal computer programs ii: Minimum storage requirement", *Journal of the ACM* **14**(4), 742–755.

Rendell, L. & Seshu, R. (1990), "Learning hard concepts through constructive induction: framework and rationale", *Computational Intelligence* **6**(4), 247–270.

Rice, J. A. (1988), *Mathematical Statistics and Data Analysis*, Wadsworth & Brooks/Cole.

Rissanen, J. (1978), "Modeling by shortest data description", *Automatica* **14**, 465–471.

Rissanen, J. (1986), "Stochastic complexity and modeling", *Ann. Statist* **14**, 1080–1100.

Rosenblatt, F. (1958), "The perceptron: A probabilistic model for information storage and organization in the brain", *Psychological Review* **65**, 386–408.

Rumelhart, D. E., Hinton, G. E. & Williams, R. J. (1986), *Learning Internal Representations by Error Propagation*, MIT Press, chapter 8.

Russell, S. J. & Norvig, P. (1995), *Artificial Intelligence: A Modern Approach*, Prentice Hall, Inc., Englewood Cliffs, New Jersey 07632.

Schaffer, C. (1993), "Selecting a classification method by cross-validation", *Machine Learning* **13**(1), 135–143.

Schaffer, C. (1994), A conservation law for generalization performance, *in* "Machine Learning: Proceedings of the Eleventh International Conference", Morgan Kaufmann Publishers, Inc., pp. 259–265.

Schapire, R. E. (1990), "The strength of weak learnability", *Machine Learning* **5**(2), 197–227.

Schumacher, H. & Sevcik, K. C. (1976), "The synthetic approach to decision table conversion", *Communications of the ACM* **19**(6), 343–351.

Shannon, C. E. (1949), "The synthesis of two-terminal switching circuits", *The Bell System Technical Journal* **28**(1), 59–98.

Shao, J. (1993), "Linear model selection via cross-validation", *Journal of the American Statistical Association* **88**(422), 486–494.

Shapiro, A. & Niblett, T. (1982), Automatic induction of classification rules for a chess endgame, *in* M. R. B. Clarke, ed., "Advances in Computer Chess", 3, Oxford: Pergamon.

Shawe-Taylor, J., Anthony, M. & Biggs, N. (1993), "Bounding sample size with the vapnik chervonenkis dimension", *Discrete Applied Mathematics* **42**(1), 65–73.

Siedlecki, W. & Sklansky, J. (1988), "On automatic feature selection", *International Journal of Pattern Recognition and Artificial Intelligence* **2**(2), 197–220.

Simon, H. (1983), Why should machines learn?, *in* R. S. Michalski, J. G. Carbonell & T. M. Mitchell, eds, "Machine learning: An Artificial Intelligence Approach", Vol. 1, Morgan Kaufmann Publishers, Inc.

Singh, M. & Provan, G. M. (1995), A comparison of induction algorithms for selective and non-selective bayesian classifiers, *in* "Machine Learning: Proceedings of the Twelfth International Conference".

Slowinski, R. (1992), *Intelligent decision support : handbook of applications and advances of the rough sets theory*, Kluwer Academic Publishers.

Smith, D. E. & Genesereth, M. R. (1985), "Ordering conjunctive queries", *Artificial Intelligence Journal* **26**(3), 171–215.

Spackman, A. K. (1988), Learning categorical criteria in biomedical domains, *in* "Proceedings of the Fifth International Machine Learning Conference", Morgan Kaufmann Publishers, Inc., pp. 36–46.

Spector, P. (1994), *An Introduction to S and S-PLUS*, Duxbury Press.

Staudte, R. G. & Sheather, S. J. (1990), *Robust Estimation and Testing*, John Wiley & Sons, Inc.

Stone, C. J. (1982), "Optimal global rates of convergence for nonparametric regression", *The Annals of Statistics* **10**(4), 1040–1053.

Stone, M. (1974), "Cross-validatory choice and assessment of statistical predictions", *Journal of the Royal Statistical Society B* **36**, 111–147.

Stone, M. (1977), "Asymptotics for and against cross-validation", *Biometrika* **64**(1), 29–35.

Stone, M. (1978), "Cross validation: A review", *Mathematische Operationsforschung und Statistik. Series Statistics* **9**(1), 127–139.

Street, W. N., Mangasarian, O. L. & Wolberg, W. H. (1995), An inductive learning approach to prognostic prediction, *in* "Machine Learning: Proceedings of the Twelfth International Conference".

Stroustroup, B. (1994), *The Design and Evolution of C++*, Addison-Wesley Publishing Company.

Takenaga, Y. & Yajima, S. (1993), NP-completeness of minimum binary decision diagram identification, Technical Report COMP 92-99, IEICE.

Taylor, C., Michie, D. & Spiegalhalter, D. (1994), *Machine Learning, Neural and Statistical Classification*, Paramount Publishing International.

Thrun *et al.* (1991), The Monk's problems: A performance comparison of different learning algorithms, Technical Report CMU-CS-91-197, Carnegie Mellon University.

Thrun, S. & Mitchell, T. M. (1995), Learning one more thing, *in* C. S. Mellish, ed., "Proceedings of the 14th International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 1217–1223.

Towell, G. G. & Shavlik, J. W. (1993), "Extracting refined rules from knowledge-based neural networks", *Machine Learning* **13**(1), 71–101.

Turney, P. D. (1993), Exploiting context when learning to classify, *in* P. B. Brazdil, ed., "Proceedings of the European Conference on Machine Learning (ECML)", pp. 402–407.

Utgoff, P. E. (1994), An improved algorithm for incremental induction of decision trees, *in* "Machine Learning: Proceedings of the Eleventh International Conference", Morgan Kaufmann, pp. 318–325.

Utgoff, P. E. (1995), Decision tree induction based on efficient tree restructuring, Technical Report 95-18, Department of Computer Science, University of Massachusetts.

Vafai, H. & De Jong, K. (1992), Genetic algorithms as a tool for feature selection in machine learning, *in* "Fourth International Conference on Tools with Artificial Intelligence", IEEE Computer Society Press, pp. 200–203.

Vafai, H. & De Jong, K. (1993), Robust feature selection algorithms, *in* "Fifth International Conference on Tools with Artificial Intelligence", IEEE Computer Society Press, pp. 356–363.

Valiant, L. G. (1984), "A theory of the learnable", *Communications of the ACM* **27**, 1134–1142.

van Laarhoven, P. & Aarts, E. (1987), *Simulated annealing : Theory and Applications*, Kluwer Academic Publishers.

Walker, M. G. (1992), Probability Estimation for Classification Trees and DNA Sequences Analysis, PhD thesis, Stanford University. STAN-CS-92-1422.

Wallace, C. & Freeman, P. (1987), "Estimation and inference by compact coding", *Journal of the Royal Statistical Society B* **49**, 240–265.

Wallace, C. & Patrick, J. (1993), "Coding decision trees", *Machine Learning* **11**, 7–22.

Way, J. & Smith, E. A. (1991), "The evolution of synthetic aperture radar systems and their progression to the EOS SAR", *IEEE Transactions on Geoscience and Remote Sensing* **29**(6), 962–985.

Wegener, I. (1987), *The Complexity of Boolean Functions*, B. G. Tuebner, Stuttgart.

Weiss, S. M. (1991), "Small sample error rate estimation for k-nearest neighbor classifiers", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(3), 285–289.

Weiss, S. M. & Indurkhya, N. (1994*a*), Decision tree pruning : Biased or optimal, *in* "Proceedings of the twelfth national conference on artificial intelligence", AAAI Press and MIT Press, pp. 626–632.

Weiss, S. M. & Indurkhya, N. (1994*b*), Small sample decision tree pruning, *in* W. W. Cohen & H. Hirsh, eds, "Proceedings of the Eleventh international conference on machine learning", Morgan Kaufmann Publishers, Inc., pp. 335–342.

Weiss, S. M. & Kulikowski, C. A. (1991), *Computer Systems that Learn*, Morgan Kaufmann Publishers, Inc., San Mateo, CA.

Wettschereck, D. (1994), A Study of Distance-Based Machine Learning Algorithms, PhD thesis, Oregon State University.

Wilson, S. W. (1987), "Classifier systems and the animat problem", *Machine Learning*.

Wnek, J. & Michalski, R. S. (1994), "Hypothesis-driven constructive induction in AQ17-HCI : A method and experiments", *Machine Learning* **14**(2), 139–168.

Wolberg, W. H. & Mangasarian, O. L. (1990), Multisurface method of pattern separation for medical diagnosis applied to breast cytology, *in* "Proceedings of the National Academy of Sciences, U.S.A.", Vol. 87, pp. 9193–9196.

Wolpert, D. H. (1992*a*), "On the connection between in-sample testing and generalization error", *Complex Systems* **6**, 47–94.

Wolpert, D. H. (1992*b*), "Stacked generalization", *Neural Networks* **5**, 241–259.

Wolpert, D. H. (1994*a*), Off-training set error and a priori distinctions between learning algorithms, Technical Report SFI TR 94-12-123, The Sante Fe Institute.

Wolpert, D. H. (1994*b*), The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework, *in* D. H. Wolpert, ed., "The Mathemtatics of Generalization", Addison Wesley.

Wolpert, D. H. (1995), On bias plus variance, Technical Report SFI TR 95-007, Santa Fe Institute.

Xu, L., Yan, P. & Chang, T. (1989), Best first strategy for feature selection, *in* "Ninth International Conference on Pattern Recognition", IEEE Computer Society Press, pp. 706–708.

Yan, D. & Mukai, H. (1992), "Stochastic discrete optimization", *Siam J. Control and Optimization* **30**(3), 594–612.

Yu, B. & Yuan, B. (1993), "A more efficient branch and bound algorithm for feature selection", *Pattern Recognition* **26**(6), 883–889.

Zhang, J. (1992*a*), Selecting typical instances in instance-based learning, *in* "Proceedings of the Ninth International Conference on Machine Learning", Morgan Kaufmann Publishers, Inc., pp. 470–479.

Zhang, P. (1992*b*), "On the distributional properties of model selection criteria", *Journal of the American Statistical Association* **87**(419), 732–737.

Zheng, Z. (1995), Constructing nominal x-of-n attributes, *in* C. S. Mellish, ed., "Proceedings of the 14th International Joint Conference on Artificial Intelligence", Morgan Kaufmann Publishers, Inc., pp. 1064–1070.

Ziarko, W. (1991), The discovery, analysis, and representation of data dependencies in databases, *in* G. Piatetsky-Shapiro & W. Frawley, eds, "Knowledge Discovery in Databases", MIT Press.

Index

Most books have indexes; most technical reports don't. They should. Any nonfiction work of more than twenty or so pages that is worth reading deserves an index.
—Leslie Lamport, LaTeX2ϵ, page 151