On Growing Better Decision Trees from Data

Kolluru Venkata Sreerama Murthy

A dissertation submitted to The Johns Hopkins University in conformity with the requirement for the degree of Doctor of Philosophy.

Baltimore, Maryland
1997

Abstract

This thesis investigates the problem of growing decision trees from data, for the purposes of classification and prediction.

After a comprehensive, multi-disciplinary survey of work on decision trees, some algorithmic extensions to existing tree growing methods are considered. The implications of using (1) less greedy search and (2) less restricted splits at tree nodes are systematically studied. Extending the traditional axis-parallel splits to *oblique* splits is shown to be practical and beneficial for a variety of problems. However, the use of more extensive search heuristics than the traditional greedy heuristic is argued to be unnecessary, and often harmful.

Any effort to build good decision trees from real-world data involves "massaging" the data into a suitable form. Two forms of data massaging, domain-independent and domain-specific, are distinguished in this work. A new framework is outlined for the former, and the importance of the latter is illustrated in the context of two new, complex classification problems in astronomy. Highly accurate and small decision tree classifiers are built for both these problems through a collaborative effort with astronomers.

Dedicated to

Amma, Babaji

Acknowledgements

My present status in life, this Ph.D. included, is a result of the extraordinary will, efforts and sacrifices of my parents.

Steven Salzberg, my thesis adviser, taught me a lot about professional attitude towards work in addition to guiding me through the nitty-gritties of research. Steven translated my wish to write a thesis into a series of concrete, well-planned sub-goals. Simon Kasif introduced me to decision trees and helped throughout my Ph.D work with his broad knowledge of the literature. Working with Holland Ford on the astronomy classification problems was a pleasure. Simon and Holland reassured me that a friendly, good-humoured nature is not an antithesis to professional competence. Eric Brill read early drafts of this thesis with great interest, and his suggestions significantly improved the thesis. I wish Eric had come to Hopkins earlier-on during my stay.

My wife Sudha's patience, encouragement, love and guidance were essential for my joining and smooth-sailing through the Ph.D. program. My brother Dakshinamurthy buffered my ups and downs with unquestioning support. Kala, Maxine and Bradley helped create a home away from home. Abhay helped shape my perspectives about the responsibilities of technology, the nature of research, etc.

Richard White and Rupali Chandar collaborated in building the astronomy classifiers, and were always friendly and helpful. Richard Beigel helped in designing OC1's randomized search. Wray Buntine, Carla Brodley, David Heath and Kristine Bennett provided software systems for experimental comparisons, and assisted in using them. Each and every faculty member, graduate student and staff member in the computer science department helped make my stay at Hopkins enjoyable and educational. In particular, I would like to mention Michael Goodrich, Michael Brent, Elli Angelopoulou, Kumar Ramaiyer, Pisupati Chandrasekhar, Lewis Stiller, John Sheppard, Paul Callahan, Jim Williams, Nancy and Tensie.

While writing this thesis, I was financially supported by National Science Foundation under Grant Nos. IRI-9116843 and IRI-9223591.

Variations of parts of this thesis have been published elsewhere. References to such publications are given below along with the Chapters or Sections in which the material appears in the thesis.

Chapter 3:

SREERAMA K. MURTHY, S. KASIF, S. SALZBERG, AND R. BEIGEL. OC1: Randomized induction of oblique decision trees. In *AAAI-93: Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC, 11–15th, July 1993. AAAI Press / The MIT Press. pages 322–327.

SREERAMA K. MURTHY, SIMON KASIF, AND STEVEN SALZBERG. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, **2**:1–33, August 1994.

Chapter 4:

SREERAMA K. MURTHY AND STEVEN SALZBERG. Lookahead and pathology in decision tree induction. In *IJCAI-95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 16th–21st, August 1995. Morgan Kaufmann Publishers Inc., San Mateo, CA. Editor: Chris Mellish. pages 1025–1031.

Chapter 5:

SREERAMA K. MURTHY AND STEVEN SALZBERG. Decision tree induction: How effective is the greedy heuristic? In *Proceedings of the First International Conference on Knowledge Discovery in Databases*, Montreal, Canada, 20th–21st, August 1995. AAAI Press. pages 222–227.

Section 6.1:

STEVEN SALZBERG, RUPALI CHANDAR, HOLLAND FORD, SREERAMA MURTHY, AND RICK WHITE. Decision trees for automated identification of cosmic-ray hits in Hubble Space Telescope images. *Publications of the Astronomical Society of the Pacific*, **107**:1–10, March 1995.

Chapter 7:

SREERAMA K. MURTHY. Using structure to improve decision trees. In *AI&Stats-95: Fifth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, 4–7th, January 1995. Society for AI and Statistics. pages 403–409.

Contents

Chapter 1

Introduction

This thesis explores a specific method for discovering mathematical regularities underlying sets of observations using a computer program.

Consider the following examples. (1) An astronomer takes a set of images using a telescope, and identifies the objects as stars and galaxies in each image. With a view to automate the identification process, he/she wants to discover, from the existing images, empirical rules governing how images of stars differ from images of galaxies. (2) A cancer center has a set of medical records accumulated over a period of time. A portion of these records belong to patients whose cancer has recurred. It is desirable to know well in advance whether a particular patient's cancer will recur, and empirical regularities in existing patient records may be a key to this information.

We refer to the general process of discovering rules from data as "exploration", "learning", or "classification". Many areas of scientific enquiry address the problem of data exploration. The science of statistics deals with assembling, classifying and tabulating data or facts. Pattern recognition, an area of engineering, attempts to find patterns (commonalities, definitive tendencies etc.) that exist in images or signals. Several areas of mathematics, such as mathematical programming and function approximation, aim at fitting representations to data. Recent disciplines such as machine learning and artificial neural networks largely concentrate on data exploration and classification techniques.

Advances in data collection methods, storage and processing technology provide a unique challenge and opportunity for automated data exploration techniques. Major scientific projects such as the Human Genome Project, the Hubble Space Telescope, and the human brain mapping initiative are generating enormous amounts of data on a daily basis. In addition, researchers and practitioners from more diverse disciplines than ever before are attempting to use automated methods to analyze their data. As the quantity and variety of data available to data exploration methods increases, there is a commensurate need for robust, efficient and versatile data exploration methods.

This thesis considers **decision trees**, which are a way to represent rules underlying data. Decision trees are hierarchical, sequential classification structures that recursively partition the set of observations (data). In this thesis, we are interested in constructing decision trees automatically from data. A hypothetical data set for a star/galaxy classification problem, and a decision tree which could have been constructed from it are given in Fig. 1.1. In the figure, the attributes w,x,y and z might represent any useful information, such as the brightness of the object, its size etc. Each node of the decision tree consists of either a test that partitions the data, or a decision about the object. Once a tree is constructed from data, it can be used to classify objects of unknown category (star or galaxy).

Automatic construction of rules in the form of decision trees has been attempted in almost all disciplines in which data exploration methods have been developed. Though the terminology and emphases differ from discipline to discipline, there are many similarities in the methodology (see Chapter 2). Several advantages of decision tree-based classification have been pointed out in the literature.

- Knowledge acquisition from pre-classified examples circumvents the bottleneck of acquiring knowledge from a domain expert.
- Tree methods are exploratory (non-parametric) as opposed to inferential (parametric). As only a few assumptions are made about the model and the data distribution, trees can model a wide range of data distributions.
- Hierarchical decomposition implies better use of available features and computational efficiency in classification.
- Tree classifiers can treat uni-modal as well as multi-modal data in the same fashion (as opposed to some statistical methods).
- Trees can be used with the same ease in deterministic as well as incomplete problems. (In deterministic domains, the dependent variable (class) can be determined perfectly from the independent variables, whereas in incomplete problems, it cannot.)
- Trees perform classification by a sequence of simple, easy-to-understand tests whose semantics are intuitively clear to domain experts. The decision tree formalism itself is intuitively appealing.

1.1 Contributions

The main contributions of this thesis, in the author's opinion, are the following.

- "Oblique" decision trees are clearly preferable to the standard axis-parallel trees for some domains. So, it is desirable to have a system capable of efficiently inducing oblique trees. This thesis (Chapter 3) proposes and evaluates one such system, OC1. The OC1 software, written by the author, has been available in the public domain since 1993. Hundreds of researchers from multiple disciplines from all over the world have retrieved this software.
- Two new, complex classification problems in astronomy are successfully solved. (Chapter 6)

Figure 1.1: A hypothetical astronomy data set and a tree generated from it. Nodes in bold face are decision nodes.

- A systematic attempt is made to quantify the effect of lookahead search for decision tree induction (Chapter 4). The existence of *pathology* [360] in the context of decision trees is reported for the first time, and several real and synthetic data sets for which lookahead hurts tree quality are presented.
- A comprehensive, multidisciplinary survey of work on decision trees is presented whose coverage is broader than that of the existing surveys in the field. (Chapter 2)

Most of this thesis (except Chapter 2) is from the point of view of machine learning, a subfield of artificial intelligence (AI). Almost all of this thesis has an experimental flavor. The investigations are empirical in nature, where carefully designed and controlled experiments form the basis of observations and conclusions. Experimental analysis has historically been a predominant form of AI research [447].

One of the grand goals of AI has been to emulate and comprehend human intelligence. A reader with such a perspective might expect machine learning systems to imitate human learning. Adaptive processes in humans in particular, and biological organisms in general, are fascinating. However, this thesis neither explores biological learning nor attempts to justify machine learning techniques using biological analogies. It is possible that automated techniques to learn from data enhance our understanding of biological learning, or *vice versa*. Nevertheless, this thesis is based on the belief that biological justification is neither necessary nor sufficient for a successful machine learning system. We believe that an attempt to evaluate aircrafts by comparing them with birds is futile [200]. Throughout this thesis, machine learning techniques are viewed as helpful tools, and not as techniques that attempt to emulate natural intelligence.

1.2 Overview of the thesis

This thesis can be roughly divided into three parts. Chapter 2 is the first part, which presents in detail the existing work on decision trees. The second part deals with one way of building "better" decision trees: through algorithmic extensions. This part comprises of Chapters 3, 4 and 5. The third part addresses another, not necessarily an alternative, way of building better trees: by re-representing or *massaging* the data into an appropriate form. Chapters 6 and 7 form the third part. We outline the contents of each of the three parts below.

Existing work: Chapter 2 presents a concise multi-disciplinary survey of decision tree related work. The main emphasis is on tree construction methodology, but pointers are also given to comparisons of decision trees with other (statistical and neural) data exploration methods, and to some recent, real-world applications. The effort is to trace the directions decision tree work has taken over the years, rather than to provide a tutorial of specific topics. Chapter 2 is intended to accomplish two things: (1) as there exist no comprehensive, multi-disciplinary surveys of decision tree work, it attempts to fill an important gap in the literature; (2) it shows the reader where in the big picture the rest of the results in this thesis fit.

Algorithmic extensions

Less restricted splits: Chapter 3 considers the problem of inducing "oblique" decision trees. Oblique decision trees can contain tests that use linear combinations of features. (Fig. 1.1 is an oblique decision tree.) These are a generalization of the more popular univariate, or axis-parallel, decision trees. (See Section 2.1.2 for decision tree terminology and basics.) In Chapter 3, we discuss the difficulties in inducing oblique trees, present an efficient algorithm and thoroughly evaluate it. We demonstrate that stochastic search is an effective and efficient tool for building oblique trees.

Less greedy gearch: Most existing methods for tree construction use greedy search for determining the splits at tree nodes — splits that are optimal locally (i.e., at a node) according to some criterion are used to partition each node. As greedy search is known to produce necessarily suboptimal trees, the advantages of replacing greedy search by one-level lookahead search are investigated in Chapter 4. In this method, the best split at a tree node is taken to be the one that can eventually produce the best $\leq$ 4-way partitioning. Systematic experimentation with artificial and real world data reveals that limited lookahead search does *not* offer significant advantages over the greedy approach. In addition, limited lookahead search produces *worse* trees than greedy search in several situations. This latter trend is known as "pathology" in the context of game trees, and, to our knowledge, has not been reported in the context of decision trees.

The counter-intuitive result that lookahead search does not improve over greedy search would stand to reason *if* the latter itself constructs near-optimal trees. Chapter 5 describes experiments that attempt to quantify how close the greedily induced trees are to the optimal ones. For hundreds of thousands of data sets, we compare the greedily induced trees to the optimal ones, under varying training data characteristics. The results of these experiments suggest that the greedy heuristic, together with pruning (Section 2.4.1), indeed produces trees with near-optimal expected depth.

Data Massaging

Domain-specific data massaging: Chapter 6 describes two projects in which decision trees are used to solve new astronomical classification problems. The first involves identifying cosmic rays (a type of noise) in Hubble Space Telescope images, and the second involves classifying stars and galaxies in Sloan Digital Sky Survey images. Both problems are characterized by data streams (images) that are directly *not* useful for classification. There are high noise levels and a very large variety of parameters that can be measured. The enormous sizes of the data streams make it essential that the classifiers be highly accurate and efficient. The author worked with a group of four other researchers for approximately 3.5 years to successfully develop decision tree classifiers for both these problems. A large portion of our effort involved re-representing the data in an appropriate form.

Domain-independent data massaging: The basis for choosing splits at decision tree nodes are *goodness measures* (Section 2.3.1), which assign a numeric "goodness" to each split. Most existing goodness measures cannot take into account the distribution or structure of the data in numeric attribute spaces. Because some statistical methods can extract this information, Chapter 7 proposes a framework to massage the data using statistical methods prior to tree induction. As an example, clustering is demonstrated to be a useful preprocessing step for several univariate and multi-variate decision tree methods.

Finally, Chapter 8 provides general conclusions and outlines interesting directions for further research.

Chapter 2

Existing work on decision trees: A multi-disciplinary survey

2.1 Introduction

A decision tree can be used for data exploration in one or more of the following ways: [1]

- To reduce a volume of data by transforming it into a more compact form which preserves the essential characteristics and provides an accurate summary.
- Discovering whether the data contains well-separated clusters of objects, such that the clusters can be interpreted meaningfully in the context of a substantive theory.
- Uncovering a mapping from independent to dependent variables that is useful for predicting the value of the dependent variable in the future.

Work related to automatically constructing and using decision trees for data description, classification and generalization exists in a wide variety of disciplines. It has been traditionally developed in the fields of statistics, engineering (logic synthesis, pattern recognition) and decision theory (decision table programming). Recently renewed interest has been generated by research in artificial intelligence (machine learning) and the neurosciences (neural networks). In spite of a diverse body of literature on automatic construction of decision trees, there exist no comprehensive, multi-disciplinary surveys of up-to-date results on this topic (see Section 2.2 for discussion of existing surveys [379, 338, 417]).

A characteristic of existing decision tree work seems to be a lack of *directed* progress. As most research on this subject is (perhaps needs to be) empirical, researchers and system developers typically try *ad hoc* variations of the basic methodology. However, such a practice can lead to redundant effort, particularly because researchers from very

[1] Adapted from [369], where a similar taxonomy was suggested in the general framework of searching for structure in data.

diverse disciplines are interested in decision trees. It is not unusual to find many papers proposing almost identical heuristics/analyses of tree construction.

As a step towards rectifying the above situation, the current chapter undertakes a concise survey of decision tree related work. We summarize significant results related to automatically constructing decision trees from data, from fields such as logic synthesis, machine learning, mathematical programming, neural networks, pattern recognition, signal processing and statistics. Although it is not the intent of the current chapter to point out specific instances of redundant results, a careful reader may notice several such examples. The main distinctive features of the current chapter, compared to existing surveys are:

- A substantial body of work that has been done after the existing surveys were written (e.g., almost all of the machine learning work) is covered. Many topics that were not discussed in the existing surveys (e.g., multivariate trees) are also covered.
- This chapter considers decision tree work in multiple disciplines in contrast to existing surveys that concentrated on specific disciplines (e.g., Safavin and Langrebe's survey [417] covers work mostly from the pattern recognition literature).
- Our main emphasis is on automatically constructing decision trees for parsimonious descriptions of, and generalization from, data. (In contrast, Moret's [338] main emphasis was on representing Boolean functions as decision trees.)
- A significant portion of this survey is devoted to comparisons of tree-based data exploration with other (e.g., statistical and neural) methods, and to recent real-world applications of decision trees.

Because there is a vast body of work on automatic construction of decision trees, we maintain the conciseness of this survey using the following guidelines and limitations.

- We do not attempt a tutorial overview of any specific topics. Our main emphasis is to trace the directions that decision tree work has taken.
- We avoid repeating many of the references from [379, 338, 417]. This is partly because the above surveys had different emphases than ours, as outlined in Section 2.2.
- We limit our references primarily to refereed journals, published books and recent conferences.
- Our coverage of decision tree applications falls far short of being comprehensive; it is merely illustrative.

Work Not Covered: Work not covered includes automatic construction of hierarchical structures using data in which the categories of objects are not known (*unsupervised* learning), present in fields such as cluster analysis [127] and machine learning (e.g., [141, 165]). A body of work using decision trees as a representational paradigm, existing in fields such as programming languages and analysis of algorithms, is not included. Work on decision trees constructed by hand (prevalent in the medical domain) is also not considered.

2.1.1 Outline of the chapter

Section 2.1.2 is intended to clarify the basics and terminology of decision trees. The next four sections provide pointers to tree construction methods. Section 2.2 gives high level pointers, mentioning existing surveys, text books and historical origins. Section 2.3 discusses work on determining splits at tree nodes. Section 2.4 discusses techniques, such as pruning, used to obtain the right sized trees. Section 2.5 ties into one section several disparate topics relevant to tree construction, such as sample size and dimensionality considerations, work on improving greedy induction, incorporating costs, estimating probabilities from decision trees etc.

Several researchers have analyzed, theoretically and empirically, the process of tree construction itself. Section 2.6 discusses NP-completeness results in the context of tree construction, work analyzing biases and assumptions, etc. Section 2.7 gives pointers to work comparing tree based data exploration to alternatives, such as multivariate statistical methods, neural networks, etc. Section 2.8 lists some recent, real-world applications of decision trees, to give the reader a feel for the versatality of the decision tree paradigm. Section 2.9 concludes the chapter.

2.1.2 Basics of tree construction

This section gives a quick (and dirty) overview of the process of decision tree construction. We assume that the reader is familiar with some form of decision trees, and only try to clarify terminology. Readers not familiar with decision trees are directed to ([417], Section II) for a good summary of basic definitions.

A decision tree is induced on a *training set*, which consists of *objects*. Each object is completely described by a set of *attributes* and a *class* label. Attributes can have *ordered* or *unordered* values. For example, integer values are ordered whereas Boolean values are not. The *concept* underlying a data set is the true mapping between the attribute set and the class label. A *noise-free* training set is one in which all the objects are generated using the underlying concept.

A decision tree contains zero or more *internal* nodes and one or more *leaf* nodes (see Fig. 1.1). All internal nodes have two or more *child* nodes. [2] All non-terminal nodes contain *splits*, which *test* the value of a mathematical or logical expression of the attributes. *Edges* from an internal node T to its children are labelled with distinct outcomes of the test at T. Each leaf node has a class label associated with it. [3] The number of classes is finite. A leaf node t is said to be *pure* if all the training samples at t belong to the same class.

The task of constructing a tree from the training set is called tree *induction*. Most existing tree induction systems proceed in a greedy top-down fashion (see Section 2.5.10 for

[2] Lubinsky [300] considered trees that can have internal nodes with just one child. At these nodes, the data are not split, but residuals are taken from a single variable regression.

[3] While converting decision tables to trees, it is common to have leaf nodes that have a "no decision" label.

exceptions). Starting with an empty tree and the entire training set, the following algorithm is applied until no more splits are possible.

- If all the training examples at the current node t belong to category c, create a leaf node with the class c and halt.
- Otherwise, score each one of the set of possible splits $\mathcal{S}$, using a *goodness measure* (Section 2.3.1).
- Choose the best split s^* as the test at the current node, and create as many child nodes as there are distinct outcomes of s^*.
- Label edges between the parent and child nodes with outcomes of s^*, and partition the training data using s^* into the child nodes.

There are several reasons why one might construct a decision tree from data, such as concise data description, discrimination or classification. *Discrimination* is the process of deriving classification rules from samples of classified objects, and *classification* is applying the rules to new objects of unknown class [190]. In other words, classification is computing the class label of an object given its attribute values. [4] An object X is classified by passing it through the tree starting at the root node. The test at each internal node along the path is applied to the attributes of X, to determine the next edge along which X should go down. The label at the leaf node at which X ends up is outputted as its classification. An object is *misclassified* by a tree if the classification outputted by the tree is not the same as the object's class label. The proportion of objects correctly classified by a decision tree is known as its *accuracy*, whereas the proportion of misclassified objects is the *error*.

A *univariate* decision tree is one in which the test at each internal node uses a single attribute. A *multivariate* decision tree may use as splits expressions containing multiple attributes. A special case of multivariate trees that we are particularly interested in is *oblique* decision trees. The tests in oblique trees use linear combinations of attributes.

Alternative Terminology

Structures very similar to decision trees have been referred to as classification trees, branched testing sequences, discriminant trees, tree structured vector quantizers and identification keys. Tree induction is also referred to as tree construction, building or growing. Training sets or samples consists of objects (also known as observations, examples or instances). Attributes are also known as features, predictors or independent variables. A decision tree imposes a partitioning in an ordered attribute space that can be geometrically represented as a collection of hyperplanes and regions. For this reason, splits are often referred to as hyperplanes, attributes as dimensions and objects as points.

[4] More precisely, a decision tree is said to perform classification if the class labels are discrete values, and *regression* if the class labels are continuous. We restrict almost entirely to classification trees in this chapter.

A class label is also referred to as category or dependent variable. Ordered domains are equivalent to or comprise of continuous, integer, real-valued and monotonous domains. Unordered domains have categorical, discrete or free variables. Internal nodes are the same as non-terminals or test nodes. Leaf nodes are the terminal nodes or decision nodes. Splits are the same as cuts or tests. Goodness measures are also known as feature evaluation criterion, feature selection criterion, impurity measure or splitting rule. A leaf node is pure when it is homogeneous. A univariate tree is also axis-parallel, and an oblique tree is the same as a linear tree.

2.2 High-level pointers

A decision tree performs mutistage hierarchical decision making. For a general rationale for multistage classification schemes and a categorization of such schemes, see [237].

2.2.1 Treatises

The work on decision tree construction in statistics has its origins in methods for exploring survey data. For a review of earlier statistical work on hierarchical classification, see [139]. Statistical programs such as AID [454], MAID [170], THAID [339] and CHAID [240] built binary segmentation trees aimed towards unearthing the interactions between predictor and dependent variables. A standard reference for the current work on decision trees from a statistical perspective is Breiman *et al.*'s excellent monograph on classification and regression trees [44].

Pattern recognition work on decision trees was motivated by the need to interpret images from remote sensing satellites such as LANDSAT in the 1970s [464]. An overview of work on decision trees in the patter recognition literature can be found in [106]. A high level comparative perspective on the classification literature in pattern recognition and artificial intelligence can be found in [76].

Decision trees in particular, and induction methods in general, arose in machine learning to avoid the knowledge acquisition bottleneck [137] for expert systems. A majority of work on decision trees in machine learning is an offshoot of Breiman *et al.*'s CART work [44] and Quinlan's ID3 algorithm [391]. Quinlan's book on C4.5 [398], although specific to his tree building program, is perhaps the best available overview of tree methodology from a machine learning perspective.

In sequential fault diagnosis, a set of possible tests with associated costs and a set of system states with associated prior probabilities are given. One of the states is a "fault-free" state and the other states represent distinct faults. The aim is to build a test algorithm that unambiguously identifies the occurrence of any system state using the given tests, while minimizing the total cost. The testing algorithms normally take the form of decision trees or AND/OR trees [488, 378]. Many heuristics used to construct decision trees are used for test sequencing also.

Vector quantization (VQ) [167] is a data compression technique that has proved useful for image coding. Tree structured vector quantizers (TSVQ) [65] are structures very similar to decision trees. A lot of work exists in the speech and signal processing literature, on building and analyzing TSVQs.

A Binary Decision Diagram (BDD) represents a Boolean function as a rooted, directed acyclic graph [279, 441]. Ordered binary decision diagrams (OBDD) [52, 53] impose restrictions on the ordering of variables at the nodes of a BDD. OBDDs have been used for digital system design, verification and testing. OBDDs are similar to decision trees, [5] and there exist several issues of common concern such as finding the minimal-sized representations.

2.2.2 Surveys

Payne and Preece [379] surveyed results on constructing identification keys, in a paper that attempted "a synthesis of a large and widely-dispersed literature" from fields such as biology, pattern recognition, decision table programming, machine fault location, coding theory and questionnaire design. Taxonomic identification keys are tree structures that have one object per leaf and for which the set of available tests (splits) is prespecified. The problem of constructing identification keys is not the same as the problem of constructing decision trees from data, but many common concerns exist (e.g: optimal key construction, choosing good tests at tree nodes etc.).

Moret [338] provided a tutorial overview of the work on representing Boolean functions as decision trees and diagrams. He summarized results on constructing decision trees in discrete variable domains. Though Moret does mention some pattern recognition work on constructing decision trees from data, this was not his main emphasis. Safavin and Landgrebe [417] more recently summarized decision tree construction methodology, almost entirely from a pattern recognition perspective. Bryant [53] surveyed the methodology and applications of ordered binary decision diagrams.

2.3 Finding splits

To build a decision tree, it is necessary to find, at each internal node, a split for the data. In case of univariate trees, finding a split amounts to finding an attribute which is the most "useful" in discriminating the input data, and finding a decision rule using the attribute. In case of multivariate trees, finding a split can be seen as finding a "composite" feature, a combination of (some of the) existing attributes that has good discriminatory power. In either of these cases, a basic task in tree building is to rank features (single or composite) according to their usefulness in discriminating the classes in the data.

[5] Oblivious decision trees [257] from the machine learning literature are nearly identical in structure to OBDDs.

The manner of growing a tree differs slightly from discipline to discipline, but several underlying concerns are the same. In pattern recognition and statistics literature, features are typically ranked using *feature evaluation rules*, and the single best feature *or* a good feature subset are chosen from the ranked list. In the context of ordered binary decision diagrams (OBDDs), the order in which variables are chosen at tree nodes determines the complexity of the OBDD, and many heuristics have been evaluated for variable order selection (eg., [457, 154]). In machine learning, feature evaluation rules are used mainly for picking the single best feature at every node of the decision tree. Methods used for selecting a good subset of features are typically quite different and are used as preprocessing steps to tree induction. (We will discuss feature subset selection methods separately in Section 2.5.1.) Tree structure vector quantizers, when they were proposed [65], were grown one layer at a time, by splitting all nodes in the previous layer. Makhoul *et al.*[306] introduced an unbalanced tree algorithm that grew the tree a node at a time. Riskin and Gray [409] proposed a greedy method for TSVQs, which is directly related to decision tree growing. [6]

2.3.1 Feature evaluation rules

When used for classification or generalization, decision trees are essentially probability estimators. Feature evaluation rules are heuristics whose aim is to produce as reliable probability estimates from training data as possible. A taxonomy, proposed by Ben-Bassat [23], is helpful in understanding the large number of existing feature evaluation criteria. Ben-Basset divides feature evaluation rules into three, not necessarily distinct, categories: rules derived from information theory, rules derived from distance measures and rules derived from dependence measures.

- **Rules derived from information theory:** Examples of this variety are rules based on Shannon's entropy. [7] Tree construction by maximizing global *mutual information*, i.e., by expanding tree nodes that contribute to the largest gain in average mutual information of the whole tree, is explored in pattern recognition [172, 437, 465]. [8] Tree construction by locally optimizing *information gain*, the reduction in entropy due to splitting each individual node, is explored in pattern recognition [197, 493, 70, 192], in sequential fault diagnosis [488] and in machine learning [391]. Mingers [323] suggested the G-statistic, an information theoretic measure that is a close approximation to χ^2 distribution, for tree construction as well as for deciding when to stop. De Merckt

[6] Chou *et al.*[85] suggested a pruning method, based on [44], for optimally pruning a balanced TSVQ. The TSVQ growing procedure suggested by Riskin and Gray [409] can be viewed as an inverse to Chou's pruning procedure.

[7] The desirable properties of a measure of entropy include symmetry, expandability, decisivity, additivity and recursivity. Shannon's entropy [439] possesses all of these properties [2]. For an insightful treatment of entropy reduction as a common theme underlying several pattern recognition problems, see [498].

[8] Goodman and Smyth [174] report that the idea of using the mutual information between features and classes to select the best feature was originally put forward by Lewis [285].

[486] suggested an attribute selection measure that combined geometric distance with information gain, and argued that such measures are more appropriate for numeric attribute spaces.

- **Rules derived from distance measures:** "Distance" here refers to the distance between class probability distributions. The feature evaluation criteria in this class measure separability, divergence or discrimination between classes. A popular distance measure is the Gini index of diversity, named after the Italian economist Corrado Gini (1884–1965). Gini index has been used for tree construction in statistics [44], pattern recognition [162] and sequential fault diagnosis [378]. Breiman *et al.* pointed out that the Gini index has difficulty when there are a relatively large number of classes, and suggested the *twoing rule* [44, 351] as a remedy. Taylor and Silverman [470] pointed out that the Gini index emphasizes equal sized offspring and purity of both children. They suggested a splitting criterion, called mean posterior improvement (MPI), that emphasizes exclusivity between offspring class subsets instead.

  Bhattacharya distance [290], Kolmogorov-Smirnoff distance [152, 413, 198] and the χ^2 statistic [21, 195, 323, 515, 503] are some other distance-based measures that have been used for tree induction. Class separation-based metrics developed in the machine learning literature [133, 514] are also distance measures. A relatively simplistic method for estimating class separation, which assumes that the values of each feature follow a Gaussian distribution in each class, was used for tree construction in [302].

- **Rules derived from dependence measures:** These measure the statistical dependence between two random variables. All dependence-bassed measures can be interpreted as belonging to one of the above two categories [23].

There exist several attribute selection criteria that do not clearly belong to any category in Ben-Basset's taxonomy. Gleser and Collen [172] and Talmon [465] used a combination of mutual information and χ^2 measures. They first measured the gain in average mutual information $I(T_i)$ due to a new split T_i, and then quantified the probability $P(I(T_i))$ that this gain is due to chance, using χ^2 tables. The split that minimized $P(I(T_i))$ was chosen by these methods. A permutation statistic was used for univariate tree construction for 2-class problems in [286]. The main advantage of this statistic is that, unlike most of the other measures, its distribution is independent of the number of training instances. As will be seen in Section 2.4, this property provides a natural measure of when to stop tree growth.

Measures that use the *activity* of an attribute have been explored for tree construction [337, 329]. The activity of a variable is equal to the testing cost of the variable times the *a priori* probability that it will be tested. The computational requirements for computing activity are the same as those for the information-based measures. Quinlan and Rivest [402] suggested the use of Risannen's minimum description length [408] for deciding which splits to prefer over others and also for pruning. Kalkanis [235] pointed out that measures like information gain and Gini index are all concave (i.e., they never report a worse goodness value after trying a split than before splitting), so there is no natural way of assessing where to stop further expansion of a node. As a remedy, Kalkanis suggested the use of the upper bounds in the confidence intervals for the misclassification error as an attribute selection criterion. [9]

Heath *et al.*[204] used the simplest possible attribute selection criteria, based on the number of misclassified objects, for oblique decision tree induction. Their measures were called *max minority* and *sum minority*, respectively denoting the maximum and the sum of the number of misclassified points on either side of a binary split. Max minority has the theoretical advantage that the depth of the tree constructed using this measure is at worst logarithmic in the number of examples. Lubinsky [297, 298] also used the number of misclassified points as a splitting criterion, calling it *inaccuracy*. The performance of these measures does not seem to be in general as good as the information theory or distance based measures, and additional tricks are needed to make these measures robust [297, 351]. Another measure suggested by Heath *et al.*, called the *sum of impurities*, assigns an integer to each class and measures the variance between class numbers in each partition [204, 351]. An almost identical measure was used earlier in the Automatic Interaction Detection (AID) program [139].

Most of the above feature evaluation criteria assume no knowledge of the probability distribution of the training objects. The optimal decision rule at each tree node, a rule that minimizes the overall error probability, is considered in [270, 271, 272] assuming that complete probabilistic information about the data is known.

Comparisons

Given the large number of feature evaluation rules, a natural concern is to decide their relative effectiveness in constructing "good" trees. Evaluations in this direction, in statistics, pattern recognition and machine learning, have been predominantly empirical in nature, though there have been a few theoretical evaluations. We will discuss the empirical comparisons here, and defer the discussion of the latter to Section 2.6.

In spite of a large number of comparative studies, very few so far have concluded that a particular feature evaluation rule is significantly better than others. A majority of studies have concluded that there is not much difference between different measures. This is to be expected as induction *per se* can not rigorously justify performance on unseen instances. Any strategy that results in superior generalization accuracy on some problems is bound to have inferior performance on some other problems. [10] Of course, comparisons

[9] Quinlan's C4.5 [398] uses a naive version of the confidence intervals for doing pessimistic pruning.

[10] Schaffer [429] stated and proved a conservation theorem that states, essentially, that positive performance in some learning situations must be offset by an equal degree of negative performance in others. To clarify the, sometimes non-intuitive, consequences of the conservation theorem, Schaffer [430] gave an example of a concept for which information *loss* gives better generalization accuracy than information gain.

of individual methods are still interesting because they throw light on which method can be used in what situations.

Baker and Jain [18] reported experiments comparing eleven feature evaluation criteria and concluded that the feature rankings induced by various rules are very similar. Several feature evaluation criteria, including Shannon's entropy and divergence measures, are compared using simulated data in [22], on a sequential, multi-class classification problem. The conclusions are that no feature selection rule is consistently superior to the others, *and* that no specific strategy for alternating different rules seems to be significantly more effective. Breiman *et al.* [44] conjectured that decision tree design is rather insensitive to any one from a large class of splitting rules, and it is the stopping rule that is crucial. Mingers [325] compared several attribute selection criteria, and concluded that tree quality doesn't seem to depend on the specific criterion used. He even claimed that random attribute selection criteria are as good as measures like information gain [391]. This later claim was refuted in [292], where the authors argued that random attribute selection criteria are prone to overfitting, and also fail when there are several noisy attributes.

Babic *et al.* [15] compared ID3 [391] and CART [44], for two clinical diagnosis problems. Miyakawa [329] compared three activity-based measures, Q, O and *loss*, both analytically and empirically. He showed that Q and O do not chose non-essential variables at tree nodes, and that they produce trees that are 1/4th the size of the trees produced by *loss*. Fayyad and Irani [133] showed that their measure C-SEP, performs better than Gini index [44] and information gain [391] for specific types of problems.

Several researchers [195, 391] pointed out that information gain is biased towards attributes with a large number of possible values. Mingers [323] compared information gain and the χ^2 statistic for growing the tree as well as for stop-splitting. He concluded that χ^2 corrected information gain's bias towards multivalued attributes, however to such an extent that they were never chosen, and the latter produced trees that were extremely deep and hard to interpret. Quinlan suggested *gain ratio* [398] as a remedy for the bias of information gain. Mantaras [310] argued that gain ratio had its own set of problems, and suggested using information theory-based *distance* between partitions for tree construction. He formally proved that his measure is not biased towards multiple-valued attributes. However, White and Liu [503] present experiments to conclude that information gain, gain ratio *and* Mantaras' measure are worse than a χ^2 based statistical measure, in terms of their bias towards multiple-valued attributes. A hypergeometric distribution is proposed as a means to avoid the biases of information gain, gain ratio and χ^2 metrics in [312]. Kononenko recently pointed out that [260] Minimum Description Length based feature evaluation criteria have the least bias towards multi-valued attributes.

2.3.2 Multivariate splits

Decision trees most commonly are univariate, i.e., they use splits based on a single attribute at each internal node. Multivariate decision trees can use splits that contain more than one attribute at each internal node. Though several methods have been developed in the literature for constructing multivariate trees, this body of work is not as well-known as that on univariate trees. We summarize below the directions work on automatically constructing multivariate trees has taken.

Most of the work on multivariate splits considered linear (oblique) trees. These are trees which have tests based on a linear combination of the attributes at some internal nodes. The problem of finding an optimal linear split (optimal with respect to any of the feature evaluation measures in Section 2.3.1) is more difficult that that of finding the optimal univariate split. In fact, finding optimal linear splits is known to be intractable for some feature evaluation rules (see Section 2.6.1 for pointers), so heuristic methods are required for finding good, albeit suboptimal, linear splits. Methods used in the literature for finding good linear tests include linear discriminant analysis, hill climbing search, linear programming, perceptron training and others.

Linear Discriminant Trees: Several authors have considered the problem of constructing tree-structured classifiers that have linear discriminants [117] at each node. You and Fu [511] used a linear discriminant at each node in the decision tree, computing the hyperplane coefficients using the Fletcher-Powell descent method [144]. Their method requires that the best set of features at each node be prespecified by a human. Friedman [152] reported that applying Fisher's linear discriminants, instead of atomic features, at some internal nodes was useful in building better trees. Qing-Yun and Fu [387] also describe a method to build linear discriminant trees. Their method uses multivariate stepwise regression to optimize the structure of the decision tree as well as to choose subsets of features to be used in the linear discriminants. More recently, use of linear discriminants at each node is considered by Loh and Vanichsetakul [294]. Unlike in [511], the variables at each stage are appropriately chosen in [294] according to the data and the type of splits desired. Other features of the tree building algorithm in [294] are: (1) it yields trees with univariate, linear combination or linear combination of polar coordinate splits, and (2) allows both ordered and unordered variables in the same linear split. Use of linear discriminants in a decision tree is considered in the remote sensing literature in [218]. A method for building linear discriminant classification trees, in which the user can decide at each node what classes need to be split, is described in [472]. John [229] recently considered linear discriminant trees in the machine learning literature.

An extension of linear discriminants are linear machines [364], which are linear structures that can discriminate between multiple classes. In the machine learning literature, Utgoff *et al.* explored decision trees that used linear machines at internal nodes [49, 115].

Locally Opposed Clusters of Objects: Sklansky and his students developed several piecewise linear discriminants based on the principle of locally opposed clusters of objects. Wassel and Sklansky [496, 450] suggested a procedure to train a linear split to minimize the error probability. Using this procedure, Sklansky and Michelotti [449] developed a system to induce a piece-wise linear classifier. Their method identifies the closest-opposed pairs of clusters in the data, and trains each linear discriminant locally. The final classifier produced by this method is a piecewise linear decision surface, not a tree. Foroutan [147] discovered that the resubstitution error rate of optimized piece-wise linear classifiers is nearly monotonic with respect to the number of features. Based on this result, Foroutan and Sklansky [148] suggest an effective feature selection procedure for linear splits that uses zero-one integer programming. Park and Sklansky [375, 376] describe methods to induce linear tree classifiers and piece-wise linear discriminators. The main idea in these methods is to find hyperplanes that cut a maximal number of *Tomek* links. Tomek links of a data set connect opposed pairs of data points for which the circle of influence between the points doesn't contain any other points.

Hill Climbing Methods: CART's use of linear combinations of attributes ([44], Chapter 5) is well-known. This algorithm uses heuristic hill climbing and backward feature elimination to find good linear combinations at each node. Murthy *et al.* [350, 351] described significant extensions to CART's linear combinations algorithm, using randomized techniques. (See Chapter 3)

Perceptron Learning: A perceptron is a linear function neuron [326, 188] which can be trained to optimize the sum of distances of the misclassified objects to it, using a convergent procedure for adjusting its coefficients. *Perceptron trees*, which are decision trees with perceptrons just above the leaf nodes, were discussed in [480]. Decision trees with perceptrons at all internal nodes were described in [482, 438].

Mathematical Programming: Linear programming has been used for building adaptive classifiers since late 1960s [216]. Given two possibly interesecting sets of points, Duda and Hart [117] proposed a linear programming formulation for finding the split whose distance from the misclassified points is minimized. More recently, Mangasarian and Bennett used linear and quadratic programming techniques to build machine learning systems in general and decision trees in particular [309, 28, 25, 307, 26]. Use of zero-one integer programming for designing vector quantizers can be found in [289]. Brown and Pittard [51] also employed linear programming for finding optimal multivariate splits at classification tree nodes. Almost all the above papers attempt to minimize the distance of the misclassified points from the decision boundary. In that sense, these methods are more similar to perceptron training methods [326], than to decision tree splitting criteria. Mangasarian [308] describes a linear programming formulation to minimize the number of misclassified points instead of the geometric distance.

Neural Trees: In the neural networks community, many researchers have recently considered hybrid structures between decision trees and neural nets. Though these techniques were developed as neural networks whose structure could be automatically determined, their outcome can be interpreted as decision trees with nonlinear splits. Examples of this work include [173, 448, 46, 87, 207, 425, 102]. Techniques very similar to those used in tree construction, such as information theoretic splitting criteria and pruning, can be found in neural tree construction also. In addition to these methods, there exist other hybrid techniques between decision trees and neural networks. Sethi [435] described a method for converting a univariate decision tree into a neural net and then retraining it, resulting in tree structured *entropy nets* with sigmoidal splits. An extension of entropy nets, that converts linear decision trees into neural nets was described in [374]. Decision trees with small multilayer networks at each node, implementing nonlinear, multivariate splits, were described in [184]. Jordan and Jacobs [233] described hierarchical parametric classifiers with small "experts" at internal nodes. Training methods for tree structured Boltzmann machines are described in [427].

Other Methods: Use of polynomial splits at tree nodes is explored in decision theory in [432]. In information theory, Gelfand and Ravishanker [161] describe a method to build a tree structured filter that has linear processing elements at internal nodes. Heath *et al.* [204, 202] used simulated annealing to find the best oblique split at each tree node. Lubinsky [299, 298] attempted bivariate trees, trees in which some functions of two variables can be used as tests at internal nodes. Lubinsky considered the use of linear cuts, corner cuts and rectangular cuts, using ordered and unordered variables.

2.3.3 Ordered vs. unordered attributes

The fields of pattern recognition and statistics historically have considered ordered or numeric attributes as the default. This seems natural considering application domains such as spectral analysis and remote sensing [464]. In these fields, special techniques [436] were developed to accommodate discrete attributes into what are primarily algorithms for ordered attributes. Fast methods for splitting multiple valued categorical variables are described in [83].

In machine learning, a subfield of Artificial Intelligence, which in turn has been dominated by symbolic processing, many tree induction methods (e.g. [388] were originally developed for categorical attributes. The problem of incorporating continuous attributes into these algorithms is considered subsequently. The problem of meaningfully discretizing a continuous dimension is considered in [134, 245, 486, 343]. Methods of discretization that operate on a single continuous attribute at a time can be said to be "local" discretization methods. In contrast, "global" discretization methods simultaneously convert all continuous attributes [81].

Fast methods for splitting a continuous dimension into more than two ranges is considered in the machine learning literature [135, 157]. Trees in which an internal node can have more than 2 children, have also been considered in the vector quantization literature [431]. An extension to ID3 [391] that distinguishes between attributes with unordered domains and attributes with linearly ordered domains is suggested in [88].

2.4 Obtaining the right sized trees

One of the main difficulties of inducing a recursive partitioning structure is knowing when to stop. Obtaining the "right" sized trees may be important for several reasons, which depend on the size of the classification problem [162]. For moderate sized problems, the critical issues are generalization accuracy, honest error rate estimation [11] and gaining insight into the predictive and generalization structure of the data. For very large tree classifiers, the critical issue is optimizing structural properties (height, balance etc.) [493, 71].

Breiman *et al.* [44] pointed out that tree quality depends more on good stopping rules than on splitting rules. Effects of noise on generalization are discussed in [363, 253]. Overfitting avoidance as a specific bias is studied in [507, 428]. Effect of noise on classification tree construction methods is studied in the pattern recognition literature in [468].

Several techniques have been suggested for obtaining the right sized trees. The most popular of these is *pruning*, whose discussion we will defer to Section 2.4.1. The following are some alternatives to pruning that have been attempted in the literature.

- Restrictions on minimum node size: A node is not split if it has smaller than k objects, where k is a parameter to the tree induction algorithm. This strategy, which is known to be not robust, is used in some early methods [152].
- Two stage search: In this variant, tree induction is divided into two subtasks: first, a good structure for the tree is determined; then splits are found at all the nodes. [12] The optimization method in the first stage may or may not be related to that used in the second stage. Lin and Fu [290] use K-means clustering for both stages, whereas Qing-Yun and Fu [387] use multi-variate stepwise regression for the first stage and linear discriminant analysis for the second stage.
- Thresholds on Impurity: In this method, a threshold is imposed on the value of the splitting criterion, such that if the splitting criterion falls below (above) the threshold, tree growth is aborted. Thresholds can be imposed on local (i.e., individual node) goodness measures or on global (i.e., entire tree) goodness. The former alternative is used in [172, 413, 390, 312] and the latter in [437]. A problem with the former method is that the value of most splitting criteria (Section 2.3.1) varies with the size of the training sample. Imposing a single threshold that is meaningful at all nodes in the tree is not easy and may not even be possible. Some feature evaluation rules, whose distribution does *not* depend on the number of training samples (i.e., a goodness value of k would have the same significance anywhere in the tree) have been suggested in the literature [286, 515, 235].

[11] For a general discussion about the relationship between complexity and predictive accuracy of classifiers, see [380].

[12] Techniques that start with a sufficient partitioning and then optimize the structure (e.g., [318]) can be thought of as being a converse to this approach.

- Trees to rules conversion: Quinlan [393, 398] gave efficient procedures for converting a decision tree into a set of production rules. Simple heuristics to generalize and combine the rules generated from trees can act as a substitute for pruning for Quinlan's univariate trees.
- Other: Cockett and Herrera [90] suggested a method to reduce an arbitrary binary decision tree to an "irreducible" form, using discrete decision theory principles. Every irreducible tree is optimal with respect to some expected testing cost criterion, and the tree reduction algorithm has the same worst-case complexity as most greedy tree induction methods. In the context of ordered binary decision diagrams, tree compaction has been attempted using operations that merge, delete and exchange nodes [53].

2.4.1 Pruning

Pruning, the method most widely used for obtaining right sized trees, was proposed by Breiman *et al.* ([44], Chapter 3). They suggested the following procedure: build the complete tree (a tree in which splitting no leaf node further will improve the accuracy on the training data) and then remove subtrees that are not contributing significantly towards generalization accuracy. It is argued that this method is better than stop-splitting rules, because it can compensate, to some extent, for the suboptimality of greedy tree induction. For instance, if there is very good node T_2 a few levels below a not-so-good node T_1, a stop-splitting rule will stop tree growth at T_1, whereas pruning may give a high rating for, and retain, the whole subtree at T_1. Kim and Koehler [249] analytically investigate the conditions under which pruning is beneficial for accuracy. Their main result states pruning is more beneficial with increasing skewness in class distribution and/or increasing sample size.

Breiman *et al.*'s pruning method [44], *cost complexity* pruning (a.k.a. weakest link pruning or error complexity pruning) proceeds in two stages. In the first stage, a sequence of increasingly smaller trees are built on the training data. In the second stage, one of these trees is chosen as the pruned tree, based on its classification accuracy on a *pruning set*. Pruning set is a portion of the training data that is set aside exclusively for pruning alone. Use of a separate pruning set is a fairly common practice. A method other than cost complexity pruning that needs a separate pruning set is Quinlan's reduced error pruning [393]. This method, unlike cost complexity pruning, does not build a sequence of trees and hence is claimed to be faster. Chou *et al.*[85] extended Breiman *et al.*'s pruning method to tree structured vector quantizers.

The requirement for an independent pruning set might be problematic especially when small training samples are involved. Several solutions have been suggested to get around this problem. Breiman *et al.* [44] describe a cross validation procedure that avoids reserving part of training data for pruning, but has a large computational complexity. Quinlan's pessimistic pruning [393, 398] does away with the need for a separate pruning set by using a statistical correlation test.

Crawford [99] analyzed Breiman *et al.*'s cross validation procedure, and pointed out that it has a large variance, especially for small training samples. He suggested a .632 bootstrap method [13] as an effective alternative. Gelfand *et al.* [162] claimed that the cross validation method is both inefficient and possibly ineffective in finding the optimally pruned tree. They suggested an efficient iterative tree growing and pruning algorithm that is guaranteed to converge. This algorithm divides the training sample into two halves and iteratively grows the tree using one half and prunes using the other half, exchanging the roles of the halves in each iteration.

Several other pruning methods exist. Quinlan and Rivest [402] used minimum description length [408] for tree construction as well as for pruning. An error in their coding method (which did not have an effect on their main conclusions) was pointed out in [491]. Forsyth *et al.* [149] recently suggested a pruning method that is based on viewing the decision tree as an encoding for the training data. Use of dynamic programming to prune trees optimally and efficiently has been explored recently in [33].

A few studies have been done to study the relative effectiveness of pruning methods [324, 91, 125]. Just as in the case of splitting criteria, no single pruning method has been adjudged to be superior to the others. The choice of a pruning method depends on the size of the training set, availability of extra data for pruning etc.

2.5 Other issues

Tree construction involves many issues other than finding good splits and knowing when to stop recursive splitting. In this section, we tie together several issues. The issues discussed in this section include feature subset selection, feature construction, chosing good subsamples, improving on greedy induction, use of fuzziness to remove data fragmentation and class overlap, incorporating attribute measurement costs and misclassification costs into tree construction, estimating class probabilities from trees, use of multiple trees to reduce variance and incremental induction of trees.

2.5.1 Sample size vs. dimensionality

The relationship between the size of the training set and the dimensionality of the problem is studied extensively in the pattern recognition literature. (For some pointers, see [212, 238, 145, 77, 236, 268, 227, 156].) Researchers considered the problem of how sample size should vary according to dimensionality and *vice versa.* Intuitively, an imbalance between the number of samples and the number of features (i.e., too many samples with too few

[13] In bootstrapping, B independent learning samples, each of size N are created by random sampling with replacement from the original learning sample L. In cross validation, L is divided randomly into B mutually exclusive, equal sized partitions. Efron [120] showed that, although cross validation closely approximates the true result, bootstrap has much less variance, especially for small samples. However, there exist arguments that cross validation is clearly preferable to bootstrap in practice [256].

attributes, or too few samples with too many attributes) can make induction more difficult. Some conclusions from the above papers can be summarized, informally, as follows:

- For a finite sized data with little or no *a priori* information, the ratio of the sample size to dimensionality must be as large as possible to suppress optimistically biased evaluations of the performance of the classifier.
- For a given sample size used in training a classifier, there exists an optimum feature size and quantization complexity (the latter refers to the number of ranges a dimension is split into). This result is true for both two-class problems and multi-class problems. [14]
- The ratio of the sample size to dimensionality should vary inversely proportional to the amount of available knowledge about the class conditional densities.

In tasks where more features than the "optimal" are available, decision tree quality is known to be affected by the redundant and irrelevant attributes [6, 424]. To avoid this problem, either a feature subset selection method (Section 2.5.1) or a method to form a small set of composite features (Section 2.5.1) can be used as a preprocessing step to tree induction. On the other hand, if the training sample has too many objects, a subsample selection method (Section 2.5.1) can be employed to filter out the unnecessary observations.

Feature subset selection

There is a large body of work on choosing relevant subsets of features (for example, see the texts [116, 35, 322]). Most of this work was not developed in the context of tree induction, but a lot of it has direct applicability. There are two components to any method that attempts to choose the best subset of features. The first is a metric using which two feature subsets can be compared to determine which is better. Feature subsets have been compared in the literature using either a feature evaluation criterion discussed in Section 2.3.1 (e.g. Bhattacharya distance was used for comparing subsets of features in [358]), or using direct error estimation [148, 230].

The second component of feature subset selection methods is a search algorithm through the space of possible feature subsets. Most existing search procedures are heuristic in nature, [15] as exhaustive search for the best feature subset is typically prohibitively expensive. A heuristic commonly used is the greedy heuristic. In *stepwise forward selection*, we start with an empty feature set, and add, at each stage, the best feature according to some criterion. In *stepwise backward elimination*, we start with the full feature set and

[14] Van Campenhout [67] argues that increasing the amount of information in a measurement subset through enlarging its size or complexity never worsens the error probability of a truly Bayesian classifier. Even after this guarantee, the cost and complexity due to additional measurements may not be worth the slight (if any) improvement in accuracy. Moreover, most real world classifiers are not truly Bayesian.

[15] An exception is the optimal feature subset selection method using zero-one integer programming, suggested by Ichino and Sklansky [217].

remove, at each step, the worst feature. When more than one feature is greedily added or removed, *beam search* is said to have been performed [445, 69]. A combination of forward selection and backward elimination, a bidirectional search, was attempted in [445].

Comparisons of heuristic feature subset selection methods resound the conclusions of studies comparing feature evaluation criteria and studies comparing pruning methods — no feature subset selection heuristic is far superior to the others. Cover *et al.* [94, 484] showed that heuristic sequential feature selection methods can do arbitrarily worse than the optimal strategy. Mucciardi and Gose [342] compared seven feature subset selection techniques empirically and concluded that no technique was uniformly superior to the others. There has been a recent surge of interest in feature subset selection methods in the machine learning community, resulting in several empirical evaluations. Some of these studies produced interesting insights on how to increase the efficiency and effectiveness of the heuristic search for good feature subsets. For examples of this work, see [251, 276, 69, 113, 336, 3].

Composite features

Sometimes the aim is not to choose a good subset of features, but instead to find a few good "composite" features, which are arithmetic or logical combinations of the atomic features. In the decision tree literature, Henrichon and Fu [206] were probably the first to discuss "transgenerated" features, features generated from the original attributes. Friedman's tree induction method [152] could consider with equal ease atomic and composite features. Techniques to search for multivariate splits (Section 2.3.2) can be seen as ways for constructing composite features. Use of linear regression to find good feature combinations is explored recently in [36].

Discovery of good combinations of Boolean features to be used as tests at tree nodes is explored in the machine learning literature in [372] as well as in signal processing [17]. Ragavan and Rendell [403] describe a method that constructs Boolean features using lookahead, and uses the constructed feature combinations as tests at tree nodes. Lookahead for construction of Boolean feature combinations is also considered in [515]. Linear threshold unit trees for Boolean functions are described in [418]. Decision trees having first order predicate calculus representations, with Horn clauses as tests at internal nodes, are considered in [497].

Subsample selection

Feature subset selection attempts to choose useful features. Similarly, subsample selection attempts to choose appropriate training samples for induction. Quinlan suggested "windowing", a random training set sampling method, for his programs ID3 and C4.5 [398, 506]. A initially randomly chosen window can be iteratively expanded to include only the "important" training samples. Several ways of choosing representative samples for Nearest Neighbor learning methods exist (see [104, 105], for examples). Some of these may be helpful for inducing trees efficiently on large samples, *if* it is possible to choose good subsamples efficiently.

2.5.2 Incorporating costs

In most real-world domains, attributes can have costs of measurement, and objects can have misclassification costs. If the measurement (misclassification) costs are not identical between different attributes (classes), decision tree algorithms need to be designed explicitly to prefer cheaper trees. Several attempts have been made to make tree construction cost-sensitive. These involve incorporating attribute measurement costs (see [365, 366, 469, 478] in machine learning literature, [107, 340] in pattern recognition and [250] in statistics) and incorporating misclassification costs [44, 96, 115, 72, 478]. Methods to incorporate attribute measurement costs typically include a cost term into the feature evaluation criterion, whereas variable misclassification costs are accounted for by using prior probabilities or cost matrices.

2.5.3 Missing attribute values

In real world data sets, it is often the case that some attribute values are missing from the data. Several researchers have addressed the problem of dealing with missing attribute values in the training as well as testing sets. For training data, Friedman [152] suggested that all objects with missing attribute values can be ignored while forming the split at each node. If it is feared that too much discrimination information will be lost due to ignoring, missing values may be substituted by the mean value of the particular feature in the training *subsample* in question. Once a split is formed, all objects with missing values can be passed down to all child nodes, both in the training and testing stages. The classification of an object with missing attribute values will be the largest represented class in the union of all the leaf nodes at which the object ends up. Breiman *et al.*'s CART system [44] more or less implemented Friedman's suggestions. Quinlan also considered the problem of missing attribute values [395].

2.5.4 Improving on greedy induction

Most tree induction systems use a greedy approach — trees are induced top-down, a node at a time. Several authors (e.g., [159, 405]) pointed out the inadequacy of greedy induction for difficult concepts. The problem of inducing globally optimal decision trees has been addressed time and again. For early work using dynamic programming and branch-and-bound techniques to convert decision tables to optimal trees, see [338].

Tree construction using partial or exhaustive lookahead has been considered in statistics [139, 122], in pattern recognition [197], for tree structured vector quantizers [410], for Bayesian class probability trees [62], for neural trees [102] and in machine learning [365, 403, 354]. Most of these studies indicate that lookahead does not cause considerable improvements over greedy induction. Murthy and Salzberg [354] argued that one-level lookahead does not help build significantly better trees, and that lookahead may actually *worsen* the quality of trees, causing *pathology* [360].

Constructing optimal or near-optimal decision trees using a two-stage approach has been attempted by many authors. In the first stage, a sufficient partitioning is induced using any reasonably good (greedy) method. In the second stage, the tree is *refined* to be as close to optimal as possible. Refinement techniques attempted include dynamic programming [318], fuzzy logic search [494] and multi-linear programming [30].

The build-and-refine strategy can be seen as a search through the space of all possible decision trees, starting at the greedily built suboptimal tree. In order to escape local minima in the search space, randomized search techniques such as genetic programming [264] and simulated annealing [55, 303] have been attempted. These methods search the space of all decision trees using random perturbations, additions and deletions of the splits. A deterministic hill-climbing search procedure has also been suggested for searching for optimal trees, in the context of sequential fault diagnosis [463].

Inducing topologically minimal trees, trees in which the number of occurrences of each attribute along each path are minimized, is the topic of [489]. Suen and Wang [462] described an algorithm that attempted to minimize the entropy of the whole tree and the class overlap simultaneously. (Class overlap is measured by the number of terminal nodes that represent the same class.)

2.5.5 Use of fuzziness

Two common criticisms of decision trees are the following: (1) As decisions in the lower levels of a tree are based on increasingly smaller fragments of the data, some of them may not have much probabilistic significance (data fragmentation). (2) As several leaf nodes can represent the same class, unnecessarily large trees may result, especially when the number of classes is large (high class overlap). It has been shown that the use of fuzzy reasoning can help reduce both the above problems.

Several researchers have considered using *soft* splits of data for decision trees. A hard split divides the data into mutually exclusive partitions. A soft split, on the other hand, assigns a probability that each point belongs to a partition, thus allowing points to belong to multiple partitions. C4.5 [398] uses a simple form of soft splitting (chapter 8). Use of fuzzy splits in pattern recognition literature can be found in [432, 494]. Jordan and Jacobs [233] describe a parametric, hierarchical classifier with soft splits. Multivariate regression trees using fuzzy, soft splitting criteria, are considered [146]. Induction of fuzzy decision trees has also been considered in [281, 512].

2.5.6 Estimating probabilities

Decision trees have crisp decisions at leaf nodes. On the contrary, class probability trees assign a probability distribution for all classes at the terminal nodes. Breiman *et al.* ([44], Chapter 4) proposed a method for building class probability trees. Quinlan discussed methods of extracting probabilities from decision trees in [397]. Buntine [62] described Bayesian methods for building, smoothing and averaging class probability trees. [16] Smoothing in the context of tree structured vector quantizers is described in [17]. An approach, which refines the class probability estimates in a greedily induced decision tree using local kernel density estimates has been suggested recently in [453].

Assignment of probabilistic goodness to splits in a decision tree is described in [187]. A unified methodology for combining uncertainties associated with attributes into that of a given test, which can then be systematically propagated down the decision tree, is given in [335].

2.5.7 Multiple trees

A known peril of decision tree construction is its variance, especially when the samples are small and the features are many [111]. Variance can be caused by random choice of training and pruning samples, by many equally good attributes only one of which can be chosen at a node, due to cross validation or because of other reasons. A few authors suggested using a collection of decision trees, instead of just one, to reduce the variance in classification performance [274, 443, 444, 62, 203]. The idea is to build a set of (correlated or uncorrelated) trees for the same training sample, and then combine their results. [17] Multiple trees have been built using randomness [203] or using different subsets of attributes for each tree [443, 444]. Classification results of the trees have been combined using either simplistic voting methods [203] or using statistical methods for combining evidence [443].

2.5.8 Incremental tree induction

Most tree induction algorithms use batch training — the entire tree needs to be recomputed to accommodate a new training example. A crucial property of neural network training methods is that they are incremental — network weights can be continually adjusted to accommodate training examples. Incremental induction of decision trees is considered by several authors. Friedman's binary tree induction method [152] could use "adaptive" features for some splits. An adaptive split depends on the training subsample it is splitting. (An overly simple example of an adaptive split is a test on the mean value of a feature.) Utgoff *et al.* proposed incremental tree induction methods in the context of univariate decision trees [479, 481] as well as multivariate trees [482]. Crawford [99] shows that approaches like Utgoff's, which attempt to update the tree so that the "best" split according to the

[16] Smoothing is the process of adjusting probabilities at a node in the tree based on the probabilities at other nodes on the same path. Averaging improves probability estimates by considering multiple trees.

[17] A lot of work exists in the neural networks literature on using committees or ensembles of networks to improve classification performance. See [193] for example. An alternative to multiple trees is a hybrid classifier that uses several small classifiers as parts of a larger classifier. Brodley [47] describes a system that automatically selects the most suitable among a univariate decision tree, a linear discriminant and an instance based classifier at each node of a hierarchical, recursive classifier.

updated sample is taken at each node, suffer from repeated restructuring. This occurs because the best split at a node vacillates widely while the sample at the node is still small. An incremental version of CART [44] that uses significance thresholds to avoid the above problem is described in [99].

2.5.9 Tree quality measures

The fact that several trees can correctly represent the same data raises the question of how to decide that one tree is better than another. Several measures have been suggested to quantify tree quality. Moret [338] summarizes work on measures such as tree size, expected testing cost and worst-case testing cost. He shows that these three measures are pairwise incompatible, which implies that an algorithm minimizing one measure is guaranteed *not* to minimize the others, for some tree. Fayyad and Irani [132] argue that, by concentrating on optimizing one measure, number of leaf nodes, one can achieve performance improvement along other measures.

Generalization accuracy is a popular measure for quantifying the goodness learning systems. The accuracy of the tree is computed using a testing set that is independent of the training set or using estimation techniques like cross-validation or bootstrap, and more accurate trees are preferred to the less accurate ones. Kononenko and Bratko [261] pointed out that comparisons on the basis of classification accuracy are unreliable, because different classifiers produce different types of estimates (e.g., some produce yes-or-no classifications, some output class probabilities) and accuracy values can vary with prior probabilities of the classes. They suggested an information based metric to evaluate a classifier, as a remedy to the above problems. Martin [311] argued that information theoretic measures of classifier complexity are not practically computable except within severely restricted families of classifiers, and suggested a generalized version of CART's [44] 1-standard error rule as a means of achieving a tradeoff between classifier complexity and accuracy.

Description length, the number of bits required to "code" the tree and the data using some compact encoding, has been suggested as a means to combine the accuracy and complexity of a classifier [402, 149] .

2.5.10 Miscellaneous

Most existing tree induction systems proceed in a greedy top-down fashion [464, 44, 391]. Bottom up induction of trees is considered in [275]. Bottom up tree induction is also common [378] in problems such as building identification keys and optimal test sequences. [18] A hybrid approach to tree construction, that combined top-down and bottom-up induction can be found in [247].

[18] Hierarchical unsupervised clustering can construct, using bottom-up or top-down methods, tree-structured classifiers. As mentioned in Section 2.1, these methods are beyond the scope of the current chapter.

We concentrate in this chapter on decision trees that are constructed from labelled examples. The problem of learning trees from decision rules instead of examples is addressed in [224]. The problem of learning trees solely from prior probability distributions is considered in [10]. Learning decision trees from qualitative causal models acquired from domain experts is the topic of [382].

Several attempts at generalizing the decision tree representation exist. Chou [82] considered decision *trellises*, where trellises are directed acyclic graphs with class probability vectors at the leaves and tests at internal nodes (i.e., trellises are trees in which internal nodes may have multiple parents). Option trees, in which every internal node holds several optional tests along with their respective subtrees, are discussed in [61, 62]. Oliver [368] suggested a method to build decision graphs, which are similar to Chou's decision trellises, using minimum length encoding principles [490]. Rymon [415] suggested SE-trees, set enumeration structures each of which can embed several decision trees.

All standard decision tree methods are applicable when rules are to be induced about one aspect, say, the presence or absence of a disease. However, it is sometimes necessary to infer rules on separate, but related aspects of a problem using identical or overlapping data sets. For example, as part of a scheduling process, decisions need to be made regarding the release of new orders into the system as well as the assignment of work pieces to available workstations. Chaturvedi and Nazareth [80] discuss possible solutions for this problem and provide algorithms for *conditional* classification.

Cox [95] argues that classification tree technology, as implemented in commercially available systems, is often more useful for pattern recognition than it is for decision support. He suggests several ways of modifying existing methods to be *prescriptive* rather than descriptive.

An interesting method for displaying decision trees on multidimensional data, using *block diagrams*, is proposed in [470]. Block diagrams can point out features of the data as well as the deficiencies in the classification method. Issues in preprocessing data to be in a form suitable to decision tree induction are discussed in some detail in [475]. Parallelization of tree induction algorithms is considered in [381]. Hardware architectures to implement decision trees are described in [226].

2.6 Analyses

Several researchers have tried to evaluate the tree induction method itself, to precisely answer questions such as is it possible to build optimal trees?, how good are particular heuristics (feature evaluation rules or pruning methods)? Most of these investigations are theoretical, though there have been a few recent empirical ones.

2.6.1 NP-completeness

Several aspects of optimal tree construction are known to be intractable [160]. Hyafil and Rivest [215] proved that the problem of building optimal decision trees from decision tables, optimal in the sense of minimizing the expected number of tests required to classify an unknown sample is NP-Complete. In the sequential fault diagnosis literature, Cox *et al.*[97] showed that, for an arbitrary distribution of attribute costs and for an arbitrary distribution of input vectors, the problem of constructing a minimum expected cost classification tree to represent a simple function, the linear threshold function, is NP-complete. They show that even the problem of identifying the root node in an optimal strategy is NP-hard. Building optimal trees from decision tables, in terms of the size of the tree (number of nodes), is considered by Murphy and McCraw in [344], who proved that for most cases, construction of storage optimal trees is NP-complete. Naumov [361] proved that optimal decision tree construction from decision tables is NP-complete under a variety of measures. The measures considered by the earlier papers on NP-completeness appear to be a subset of Naumov's measures, though he does not reference any of the existing work. The problem of constructing the smallest decision tree which best distinguishes characteristics of multiple distinct groups is shown to be NP-complete in [476].

Comer and Sethi [92] studied the asymptotic complexity of trie index construction in the document retrieval literature. Megiddo [317] investigated the problem of polyhedral separability (separating two sets of points using k hyperplanes), and proved that several variants of this problem are NP-complete. Results in the above three papers throw light on the complexity of decision tree induction. Lin *et al.*[288, 287] disussed NP-hardness of the problem of designing optimal pruned tree structured vector quantizers (TSVQ).

Most of the above results consider only univariate decision tree construction. Intuitively, linear or multivariate tree construction should be more difficult than univariate tree construction, as there is a much larger space of splits to be searched. More precisely, hyperplanes can dichotomize a set of n d-dimensional vectors in at most $2 * \sum_{k=0}^{d} \binom{(n-1)}{k}$ ways if $n > d+1$ and 2^n ways if $n \leq d+1$, and for any given n and d, one can find a set of vectors for which this bound is achieved ([473]). [19] Heath [202] proved that the problem of finding the split that minimizes the number of misclassified points, given two sets of mutually exclusive points, is NP-complete. Hoeffgen *et al.* [208] proved that a more general problem is NP-hard — they proved that, for any $C \geq 1$, the problem of finding a hyperplane that misclassifies no more than $C * opt$ examples, where opt is the minimum number of misclassifications possible using a hyperplane, is also NP-hard.

As the problem of finding a single linear split is NP-hard, it is no surprise that the problem of building the optimal linear decision trees is NP-hard. However, one might hope that, by reducing the size of the decision tree, or the dimensionality of the data, it might be possible to make the problem tractable. This does not seem to be the case either . Blum and Rivest [32] showed that the problem of constructing an optimal 3-node neural network is NP-complete. Goodrich [176] proved that optimal (smallest) linear decision tree construction is NP-complete even in three dimensions.

[19] Thanks to Kevin Van Horn for pointing this out.

2.6.2 Other analytical results

Goodman and Smyth [174] showed that greedy top-down induction of decision trees is directly equivalent to a form of Shannon-Fano prefix coding [131]. A consequence of this result is that top-down tree induction (using mutual information) is necessarily suboptimal in terms of average tree depth. Trees of maximal size generated by the CART algorithm [44] have been shown to have an error rate bounded by twice the Bayes error rate, and to be asymptotically Bayes optimal [177]. Miyakawa [328] considered the problem of coverting decision tables to optimal trees, and studied the properties of *optimal* variables, the class of attributes only members of which can be used at the root of an optimal tree.

Eades and Staples [119] showed that the optimality in search trees, in terms of worst-case depth, is very closely related to *regularity.* A c-regular tree is a tree in which all nodes have c children, and if one child of an internal node is a leaf, then so are all other children. A tree is regular is it is c-regular for any c. As irregular trees are not likely to be optimal, splitting rules (Section 2.3.1) that tend to slice off small corners of the attribute space building highly unbalanced trees are less likely to find optimal trees.

Computational Learning Theory is a young discipline that studies the "learnability" of specific concepts or concept classes. For a good introduction to the theory of learnability, see [242]. We summarize below significant learnability results for decision trees. Ehrenfeucht and Haussler [121] gave an algorithm for PAC-learning (without membership queries) decision trees of constant rank in polynomial time. They also gave a PAC-learning algorithm for general polynomial size decision trees in time $O(n^{O(\log n)})$. Kushilevitz and Mansour [273] gave a polynomial time PAC-learning algorithm with membership queries for decision trees under the uniform distribution. Hancock [189] gave a polynomial time algorithm for PAC-learning read-k decision trees. Bshouty [54] showed that decision trees are learnable under the model of exact learning with membership queries and unrestricted equivalence queries. Recently, agnostic PAC-learning [13] and pruning [205] have been studied by the learnability theory community.

In the context of ordered binary decision diagrams (OBDD), the bounds on the tree size have been investigated, as a function of the tree compaction operators and the specific Boolean functions being represented (eg., [315, 457, 201]).

2.6.3 Tools

Some authors pointed out the similarity or equivalence between the problem of constructing decision trees and existing, seemingly unrelated, problems. Such view points provide valuable tools for analyzing decision trees. Wang and Suen [493] show that entropy-reduction point of view is powerful in theoretically bounding search depth and classification error. Chou and Gray [84] view decision trees as variable-length encoder-decoder pairs, and show that rate is equivalent to tree depth while distortion is the probability of misclassification.

Goodman and Smyth [174] establish the equivalence between decision tree induction and a form of Shannon-Fano prefix coding, and show that this comparison leads to several interesting insights. Brandman *et al.* [37] suggested a universal technique to lower bound the size and other characteristics of decision trees for arbitrary Boolean functions. This technique is based on the power spectrum coefficients of the n-dimensional Fourier transform of the function. Turksen and Zhao [477] proved the equivalence between a pseudo-Boolean analysis and the ID3 algorithm [391].

2.6.4 Assumptions and biases

Most tree induction methods are heuristic in nature. They use several assumptions and biases, hoping that together the heuristics produce good trees. Some authors have attempted to evaluate the validity and relevance of the assumptions and biases in tree induction. [20]

- Assumption: *Multi-stage classifiers may be more accurate than single stage classifiers.* Analysis: However, the data fragmentation caused by multi-stage hierarchical classifiers may compensate for the gain in accuracy. Michie [320] argues that top-down induction algorithms may provide overly complex classifiers that have no real conceptual structure in encoding relevant knowledge. As a solution to this problem, Gray [179] suggested an induction method that generates a single disjuncts of conjuncts rule, using the same time complexity as tree induction. The efficacy of multi-level decision trees is compared by Holte [209] to simple, one-level classification rules. He concluded that, on most real world data sets commonly used by the machine learning community [346], decision trees do not perform significantly better than one level rules. These conclusions, however, were refuted by Elomaa [123] on several grounds. Elomaa argued that Holte's observations may have been the peculiarities of the data he used, and that the slight differences in accuracy that Holte observed were still significant.
- Bias: *Smaller consistent decision trees have higher generalization accuracy than larger consistent trees* (Occam's Razor). Analysis: Murphy and Pazzani [347] empirically investigated the truth of this bias. Their experiments indicate that this conjecture seems to be true. However, their experiments indicate that the smallest decision trees typically have lesser generalization accuracy than trees that are slightly larger. In an extension of this study, Murphy [345] evaluated the size bias as a function of concept size. He concluded that (1) bias for smaller trees is generally beneficial in terms of accuracy and that (2) though larger trees perform better than smaller ones for high-complexity concepts, it is better to guess the correct size randomly than to have a prespecified size bias.
- Assumption: *Locally optimizing information or distance based splitting criteria,* (Section 2.3.1) *tends to produce small, shallow, accurate trees.* Analysis: Aclass of binary splits $\mathcal{S}$ for a data set is said to be complete if, informally, for every partition of

[20] It is argued empirically [111] that the variance in decision tree methods is more a reason than bias for their poor performance on some domains.

the data, there exists a member of $\mathcal{S}$ that induces the partition. Zimmerman [516] considered the problem of building identification keys for complete classes of splits, given arbitrary class distributions. Garey and Graham [159] analyze the properties of recursive greedy splitting on the quality of trees induced from decision tables, and showed that greedy algorithms using information theoretic splitting criteria can be made to perform arbitrarily worse than the optimal. Kurzynski [270] showed that, for globally optimum performance, decisions made at each node should "emphasize the decision that leads to a greater joint probability of correct classification at the next level", i.e., decisions made at different nodes in the tree should *not* be independent. Loveland [296] analyzed the performance of variants of Gini index in the context of sequential fault diagnosis.

Goodman and Smyth [174, 175] analyzed mutual information based greedy tree induction from an information theoretic view point. They proved that mutual information-based induction is equivalent to a form of Shannon-Fano prefix coding, and through this insight argued that greedily induced trees are nearly optimal in terms of depth. This conjecture is substantiated empirically in [353], where it is shown that the expected depth of trees greedily induced using information gain [391] and Gini index [44] is very close to that of the optimal, under a variety of experimental conditions. Relationship between feature evaluation by Shannon's entropy and the probability of error is investigated in [263, 406].

2.7 Comparisons with other exploration methods

There exist several alternatives to decision trees for data exploration, such as neural networks, nearest neighbor methods and regression analysis. Several researchers have compared trees to these other methods on specific problems.

An early study comparing machine learning methods for learning from examples can be found in [112]. Comparisons of symbolic and connectionist methods can be found in [501, 440]. Quinlan empirically compared decision trees to genetic classifiers [394] and to neural networks [400]. Thrun *et al.* [471] compared several learning algorithms on simulated Monk's problems. Palvia and Gordon [373] compared decision tables, decision trees and decision rules, to determine which formalism is best for decision analysis.

Multilayer perceptrons and CART (with and without linear combinations) [44] are compared in [12] to find that there is not much difference in accuracy. Similar conclusions were reached in [142] when ID3 [391] and backpropagation were compared. Talmon *et al.* [467] compared classification trees and neural networks for analyzing electrocardiograms (ECG) and concluded that no technique is superior to the other. In contrast, ID3 is adjudged to be slightly better than connectionist and Bayesian methods in [458]. Brown *et al.* [50] compared backpropagation neural networks with decision trees on three problems that are known to be multimodal. Their analysis indicated that there was not much difference between both methods, and that neither method performed very well in its "vanilla" state.

The performance of decision trees improved in [50] when multivariate splits were used, and backpropagation networks did better with feature selection.

Giplin *et al.* [171] compared stepwise linear discriminant analysis, stepwise logistic regression and CART [44] to three senior cardiologists, for predicting the problem of predicting whether a patient would die within a year of being discharged after an acute myocardial infarction. Their results showed that there was no difference between the physicians and the computers, in terms of the prediction accuracy. Kors and Van Bemmel [262] compared statistical multivariate methods with heuristic decision tree methods, in the domain of electrocardiogram (ECG) analysis. Their comparisons show that decision tree classifiers are more comprehensible and flexible to incorporate or change existing categories. Pizzi and Jackson [384] compare an expert systems developed using traditional knowledge engineering methods to Quinlan's ID3 [391] in the domain of tonsillectormy. Comparisons of CART to multiple linear regression and discriminant analysis can be found in [66] where it is argued that CART is more suitable than the other methods for very noisy domains with lots of missing values.

Comparisons between decision trees and statistical methods like linear discriminant function analysis and automatic interaction detection (AID) are given in [313], where it is argued that machine learning methods sometimes outperform the statistical methods and so should not be ignored. Feng *et al.* [138] present a comparison of several machine learning methods (including decision trees, neural networks and statistical classifiers) as a part of the European Statlog [21] project. Their main conclusions are that (1) no method seems uniformly superior to others, (2) machine learning methods seem to be superior for multimodal distributions, and (3) statistical methods are computationally the most efficient.

Long *et al.* [295] compared Quinlan's C4 [398] to logistic regression on the problem of diagnosing acute cardiac ischemia, and concluded that both methods came fairly close to the expertise of the physicians. In their experiments, logistic regression outperformed C4. Curram and Mingers [100] compare decision trees, neural networks and discriminant analysis on several real world data sets. Their comparisons reveal that linear discriminant analysis is the fastest of the methods, when the underlying assumptions are met, and that decision trees methods overfit in the presence of noise. Dietterich *et al.* [110] argue that the inadequacy of trees for certain domains may be due to the fact that trees are unable to take into account some statistical information that is available to other methods like neural networks. They show that decision trees perform significantly better on the text-to-speech conversion problem when extra statistical knowledge is provided.

[21] The Statlog project is initiated by the European Commission, and its full title is “The Comparative Testing of Statistical and Logical Learning Algorithms on Large-Scale Applications to Classification, Prediction and Control”.

2.8 Selected real-world applications

This section lists a few recent real-world applications of decision trees. The aim is to give the reader a "feel" for the versatility and usefulness of decision tree methods for data exploration, and *not* to be useful for readers interested in finding the potential of tree classifiers in specific domains. Our coverage of applications is, by necessity, very limited. **All the application papers cited below were published between 1993 and 1995 in refereed journals**. We also restrict to application domains where the domain scientists tried to use decision trees, rather than where decision tree researchers tested their algorithm(s) on several application domains. The application areas are listed below in alphabetical order.

- **Agriculture:** Application of a range of machine learning methods to problems in agriculture and horticulture is described in [316].
- **Astronomy:** Astronomy has been an active domain for using automated classification techniques. [22] Use of decision trees for filtering noise from Hubble Space Telescope images was reported recently in [424]. Decision trees have helped in star-galaxy classification [500], determining galaxy counts [499] and discovering quasars [244] in the Second Palomar Sky Survey. Use of neural trees for ultraviolet stellar spectral classification is described in [183].
- **Biomedical Engineering:** Use of decision trees for identifying features to be used in implantable devices can be found in [169].
- **Control Systems:** Automatic induction of decision trees was recently used for control of nonlinear dynamical systems [213].
- **Financial analysis:** Use of CART [44] for asserting the attractiveness of buy-writes is reported in [319].
- **Manufacturing and Production:** Decision trees have been recently used to nondestructively test welding quality [124], for semiconductor manufacturing [225], for increasing productivity [243], for material procurement method selection [103], to accelerate rotogravure printing [126], for process optimization in electrochemical machining [130], to schedule printed circuit board assembly lines [383], to uncover flaws in a Boeing manufacturing process [407] and for quality control [185]. For a recent review of the use of machine learning (decision trees and other techniques) in scheduling, see [14].
- **Medicine:** Medical research and practice have long been important areas of application for decision tree techniques. Recent uses of automatic induction of decision trees can be found in diagnosis [259], cardiology [295, 129, 258], psychiatry [314], gastroenterology [234], for detecting microcalcifications in mammography [508], to

[22] For a general description of modern classification problems in astronomy, which prompt the use of pattern recognition and machine learning techniques, see [269].

analyze Sudden Infant Death (SID) syndrome [504] and for diagnosing thyroid disorders [140].

- **Molecular biology:** Initiatives such as the Human Genome Project and the GenBank database offer fascinating opportunities for machine learning and other data exploration methods in molecular biology. Recent use of decision trees for analyzing amino acid sequences can be found in [442] and [423].
- **Object recognition:** Tree based classification has been used recently for recognizing three dimensional objects [456, 57] and for high level vision [255].
- **Pharmacology:** Use of tree based classification for drug analysis can be found in [101].
- **Physics:** Decision trees have been used for the detection of physical particles [34].
- **Plant diseases:** CART [44] was recently used to assess the hazard of mortality to pine trees [19].
- **Power systems:** Power system security assessment [199] and power stability prediction [414] are two areas in power systems maintenance for which decision trees were used.
- **Remote Sensing:** Remote sensing has been a strong application area for pattern recognition work on decision trees (see [464, 247]). A recent use of tree-based classification in remote sensing can be found in [416].
- **Software development:** Regression trees (and backpropagation networks) were recently used to estimate the development effort of a given software module in [266], where it is argued that machine learning methods compare favorably with traditional methods.
- **Text processing:** A recent use of ID3 [391] for medical text classification can be found in [282].
- **Miscellaneous:** Decision trees have also been used recently for building personal learning assistants [327] and for classifying sleep signals [267].

2.9 A word of caution

The hierarchical, recursive tree construction methodology is simple and intuitively appealing. However, the simplicity of the methodology should not lead a practitioner to take a slack attitude towards using decision trees. Just as in the case of statistical methods or neural networks, building a successful tree classifier for an application requires a thorough understanding of the problem itself, and a deep knowledge of tree methodology.

This chapter attempted a multi-disciplinary survey of work in automatically constructing decision trees from data. We gave pointers to work in fields such as pattern recognition, statistics, machine learning, mathematical programming, neural networks etc.

We attempted to provide a self-contained, concise description of the directions which decision tree work has taken over the years. Our larger goal is to help avoid some redundant, *ad hoc* effort, both from researchers and from system developers.

Chapter 3

Oblique decision trees

3.1 Introduction

Many variants of decision tree (DT) algorithms have concentrated on decision trees in which each node checks the value of a single attribute. In numeric attribute spaces, the tests have the form $x_i > k$, where x_i is one of the attributes of an example and k is a constant. This class of decision trees may be called *axis-parallel*, because the tests at each node are equivalent to axis-parallel hyperplanes in the attribute space. An example of such a decision tree is given in Figure 3.1, which shows both a tree and the partitioning it creates in a 2-D attribute space.

In this chapter, we examine decision trees that test a linear combination of the attributes at each internal node. More precisely, let an example take the form $X = x_1, x_2, \ldots, x_d, C_j$ where C_j is a class label and the x_i's are real-valued attributes.[23] The test at each node will then have the form:

$$\sum_{i=1}^{d} a_i x_i + a_{d+1} > 0 \tag{3.1}$$

where $a_1, \ldots, a_{d+1}$ are real-valued coefficients. Because these tests are equivalent to hyperplanes at an oblique orientation to the axes, we call this class of decision trees *oblique* decision trees. (Trees of this form have also been called "linear" (Section 2.3.2) and "multivariate" [49] . We prefer the term "oblique" to aid geometric intuition and because "multivariate" includes non-linear combinations of the variables, i.e., curved surfaces.) It is clear that these are simply a more general form of axis-parallel trees, since by setting $a_i = 0$ for all coefficients but one, the test in Eq. 3.1 becomes the familiar univariate test. Note that oblique decision trees produce polygonal (polyhedral) partitionings of the attribute

[23] The constraint that $x_1, \ldots, x_d$ are real-valued does not necessarily restrict oblique decision trees to numeric domains. Several researchers have studied the problem of converting symbolic (unordered) domains to numeric (ordered) domains and vice versa (Section 2.3.3). To keep the discussion simple, however, we will assume that all attributes have numeric values.

Figure 3.1: The left side of the figure shows a simple axis-parallel tree that uses two attributes. The right side shows the partitioning that this tree creates in the attribute space.

space, while axis-parallel trees produce partitionings in the form of hyper-rectangles that are parallel to the feature axes.

It should be intuitively clear that when the underlying concept is defined by a polygonal space partitioning, it is preferable to use oblique decision trees for classification. For example, there exist many domains in which one or two oblique hyperplanes will be the best model to use for classification. In such domains, axis-parallel methods will have to approximate the correct model with a staircase-like structure, while an oblique tree-building method could capture it with a tree that was both smaller and more accurate.[24] Figure 3.2 gives an illustration.

For a review of work on oblique (linear) decision trees, see Section 2.3.2. The purpose of the current chapter is to review the strengths and weaknesses of some of the existing methods, to design a system that combines some of the strengths and overcomes the weaknesses, and to evaluate that system empirically and analytically. The main contributions and conclusions of the current chapter are as follows:

- We have developed a new, randomized algorithm for inducing oblique decision trees from examples. This algorithm extends the work of Breiman *et al.*[44] (Chapter 5). Randomization helps significantly in learning many concepts.
- Our algorithm is fully implemented as an oblique decision tree induction system and is available over the Internet. The code can be retrieved by anonymous ftp from ftp://ftp.cs.jhu.edu/pub/oc1/oc1.tar.Z.
- The randomized hill-climbing algorithm used in OC1 is more efficient than other existing randomized oblique decision tree methods (described below). In fact, the current implementation of OC1 guarantees a worst-case running time that is only

[24] Note that though a given oblique tree may have fewer leaf nodes than an axis-parallel tree—which is what we mean by "smaller"—the oblique tree may in some cases be larger in terms of information content, because of the increased complexity of the tests at each node.

Figure 3.2: The left side shows a simple 2-D domain in which two oblique hyperplanes define the classes. The right side shows an approximation of the sort that an axis-parallel decision tree would have to create to model this domain.

$O(\log n)$ times greater than the worst-case time for inducing axis-parallel trees (i.e., $O(dn^2 \log n)$ vs. $O(dn^2)$).

- The ability to generate oblique trees often produces very small trees compared to axis-parallel methods. When the underlying problem requires an oblique split, oblique trees are also more accurate than axis-parallel trees. Allowing a tree-building system to use both oblique and axis-parallel splits broadens the range of domains for which the system should be useful.

The remaining sections of this chapter follow this outline: the remainder of this section discusses the complexity issues involved in inducing oblique decision trees. Section 3.2 briefly reviews some existing techniques for oblique DT induction, outlines some limitations of each approach, and introduces the OC1 system. Section 3.3 describes the OC1 system in detail. Section 3.4 describes experiments that (1) compare the performance of OC1 to that of several other axis-parallel and oblique decision tree induction methods on a range of real-world datasets, (2) demonstrate empirically that OC1 significantly benefits from its randomization steps and (3) demonstrate that randomization is sparingly used in OC1, ensuring efficiency.

3.1.1 Complexity of inducing oblique decision trees

One reason for the relatively few papers on the problem of inducing oblique decision trees is the increased computational complexity of the problem when compared to the axis-parallel case. There are two important issues that must be addressed. In the context of top-down decision tree algorithms, we must address the complexity of finding optimal separating hyperplanes (decision surfaces) for a given node of a decision tree. An optimal hyperplane will minimize the impurity measure used; e.g., impurity might be measured by the total

Figure 3.3: For n points in d dimensions ($n \geq d$), there are at most $n \cdot d$ distinct axis-parallel splits. However, there can be $2 * \sum_{k=0}^{d} \binom{(n-1)}{k}$ oblique splits if $n > d + 1$ and 2^n oblique splits if $n \leq d + 1$. This figure shows all distinct oblique and axis-parallel splits for two specific points in 2-D.

number of examples mis-classified. The second issue is the lower bound on the complexity of finding optimal (e.g., smallest size) trees.

Let us first consider the issue of the complexity of selecting an optimal oblique hyperplane for a single node of a tree. In a domain with n training instances, each described using d real-valued attributes, there are at most $2^d \cdot \binom{n}{d}$ distinct d-dimensional oblique splits; i.e., hyperplanes that divide the training instances uniquely into two nonoverlapping subsets. This upper bound derives from the observation that every subset of size d from the n points can define a d-dimensional hyperplane, and each such hyperplane can be rotated slightly in 2^d directions to divide the set of d points in all possible ways. Figure 3.3 illustrates these upper limits for two points in two dimensions.

More precisely, hyperplanes can dichotomize a set of n d-dimensional vectors in at most $2 * \sum_{k=0}^{d} \left(\begin{array}{c} n-1 \\ k \end{array} \right)$ ways if $n > d + 1$ and 2^n ways if $n \leq d + 1$, and for any given n and d, one can find a set of vectors for which this bound is achieved [473]. For axis-parallel splits, on the other hand, there are only $n \cdot d$ distinct possibilities, and axis-parallel methods such as C4.5 [398] and CART [44] can exhaustively search for the best split at each node. The problem of searching for the best oblique split is therefore much more difficult than that of searching for the best axis-parallel split. In fact, the problem is NP-hard (see Section 2.3.2) and any method for finding the optimal oblique split is likely to have exponential cost (assuming $P \neq NP$). Intuitively, the problem is that it is impractical to enumerate all distinct hyperplanes and choose the best, as is done in axis-parallel decision trees. However, any non-exhaustive deterministic algorithm for searching through all these hyperplanes is prone to getting stuck in local minima.

On the other hand, it is possible to define impurity measures for which the problem of finding optimal hyperplanes can be solved in polynomial time. For example, if one minimizes the sum of distances of mis-classified examples, then the optimal solution can be found using linear programming methods. However, classifiers are usually judged by how many points they classify correctly, regardless of how close to the decision boundary a point may lie. Thus most of the standard measures for computing impurity (Section 2.3.1) base their calculation on the discrete number of examples of each category on either side of the hyperplane. Section 3.2 discusses the linear programming approach further, and Section 3.4.2 presents comparisons of our approach with a method that uses linear programming to find splits.

Now let us address the second issue, that of the complexity of building a small tree. The problem of inducing the smallest axis-parallel decision tree is known to be NP-hard (Section 2.6.1). It is also easy to see that the problem of constructing an optimal (e.g., smallest) oblique decision tree is NP-hard. This conclusion follows from the work of Blum and Rivest [32]. Their result implies that in d dimensions (i.e., with d attributes) the problem of producing a 3-node oblique decision tree that is consistent with the training set is NP-complete. More specifically, they show that the following decision problem is NP-complete: given a training set T with n examples and d Boolean attributes, does there exist a 3-node neural network consistent with T? From this it is easy to show that the following question is NP-complete: given a training set T, does there exist a 3-leaf-node oblique decision tree consistent with T?

Note that one can generate the smallest axis-parallel tree that is consistent with the training set in polynomial time *if* the number of attributes is a constant. This can be done by using dynamic programming (e.g.: [318]) or branch and bound techniques (e.g.: [39]). But when the tree uses oblique splits, it is not clear, even for a fixed number of attributes, how to generate an optimal (e.g., smallest) decision tree in polynomial time. Goodrich [176] showed that the problem of inducing the smallest oblique decision tree is NP-hard even in three dimensions. This suggests that the complexity of constructing good oblique trees is greater than that for axis-parallel trees.

As a result of these complexity considerations, we took the pragmatic approach of trying to generate small trees, but not looking for the smallest tree. The greedy approach used by OC1 and virtually all other decision tree algorithms implicitly tries to generate small trees. In addition, it is easy to construct example problems for which the optimal split at a node will not lead to the best tree; thus our philosophy as embodied in OC1 is to find locally "good" splits, but not to spend excessive computational effort on improving the quality of these splits.

```
To induce a split at node $T$ of the decision tree:
  Normalize values for all $d$ attributes.
  $L = 0$
  While (TRUE)
      $L = L + 1$
      Let the current split $s_L$ be $v \leq c$, where $v = \sum_{i=1}^{d} a_i x_i$.
      For $i = 1, \ldots, d$
          For $\gamma$ = -0.25,0,0.25
              Search for the $\delta$ that maximizes the goodness of the split $v - \delta(a_i + \gamma) \leq c$.
          Let $\delta^*$,$\gamma^*$ be the settings that result in highest goodness in these 3 searches.
          $a_i = a_i - \delta^*$, $c = c - \delta^* \gamma^*$.
      Perturb $c$ to maximize the goodness of $s_L$, keeping $a_1, \ldots, a_d$ constant.
      If |goodness($s_L$) - goodness($s_{L-1}$)| $\leq \epsilon$ exit while loop.
  Eliminate irrelevant attributes in $\{a_1, \ldots, a_d\}$ using backward elimination.
  Convert $s_L$ to a split on the un-normalized attributes.
  Return the better of $s_L$ and the best axis-parallel split as the split for $T$.
```

Figure 3.4: The procedure used by CART with linear combinations (CART-LC) at each node of a decision tree.

3.2 Details of some existing methods

Before describing the OC1 algorithm, we will briefly discuss some existing oblique DT induction methods. The methods discussed are CART with linear combinations, Linear Machine Decision Trees, Simulated Annealing of Decision Trees and Linear Programming based tree building methods. For a more comprehensive list of pointers to existing work on oblique trees, see Section 2.3.2.

CART-LC:

The first oblique decision tree algorithm to be proposed was CART with linear combinations [44, chapter 5]. This algorithm, referred to henceforth as CART-LC, is an important basis for OC1. Figure 3.4 summarizes (using Breiman *et al.*'s notation) what the CART-LC algorithm does at each node in the decision tree. The core idea of the CART-LC algorithm is how it finds the value of δ that maximizes the goodness of a split. This idea is also used in OC1, and is explained in detail in Section 3.3.1.

After describing CART-LC, Breiman *et al.*point out that there is still much room for further development of the algorithm. OC1 is an extension of CART-LC that includes some significant additions. It addresses the following limitations of CART-LC:

- CART-LC is fully deterministic. There is no built-in mechanism for escaping local minima, although such minima may be very common for some domains. Figure 3.5

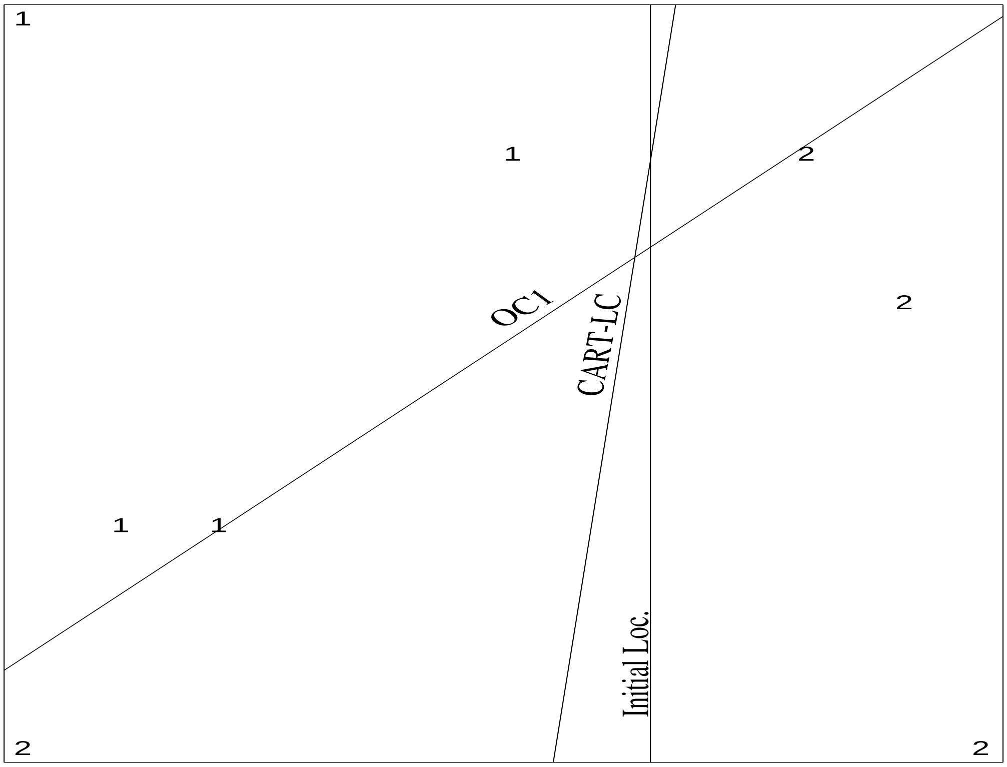

Figure 3.5: The deterministic perturbation algorithm of CART-LC fails to find the correct split for this data, even when it starts from the location of the best axis-parallel split. OC1 finds the correct split using one random jump.

shows a simple data set, containing just 8 points in 2 dimensions, for which CART-LC gets stuck in a local minimum.

- CART-LC sometimes makes adjustments that increase the impurity of a split. This feature was probably included to allow it to escape some local minima. Because of this feature, there is no upper bound on the time spent at any node in the decision tree. CART-LC halts when no perturbation changes the impurity more than ϵ, but because impurity may increase and decrease, the algorithm can spend arbitrarily long time at a node. We found several simple data sets for which CART-LC does not halt, unless an arbitrary stopping condition is imposed.

To emphasize that the above two characteristics of CART-LC are indeed limitations in real-world data, we trace in Figure 3.6 typical runs of CART-LC and OC1 on the *Dim Star/Galaxy* data set described in Section 3.4.2. This figure plots the value of the impurity at every perturbation. It can be seen from the figure that (1) CART-LC does not find as good a split as OC1 because of local minima, and (2) the impurity of the split found by CART-LC's does not monotonically decrease with time.

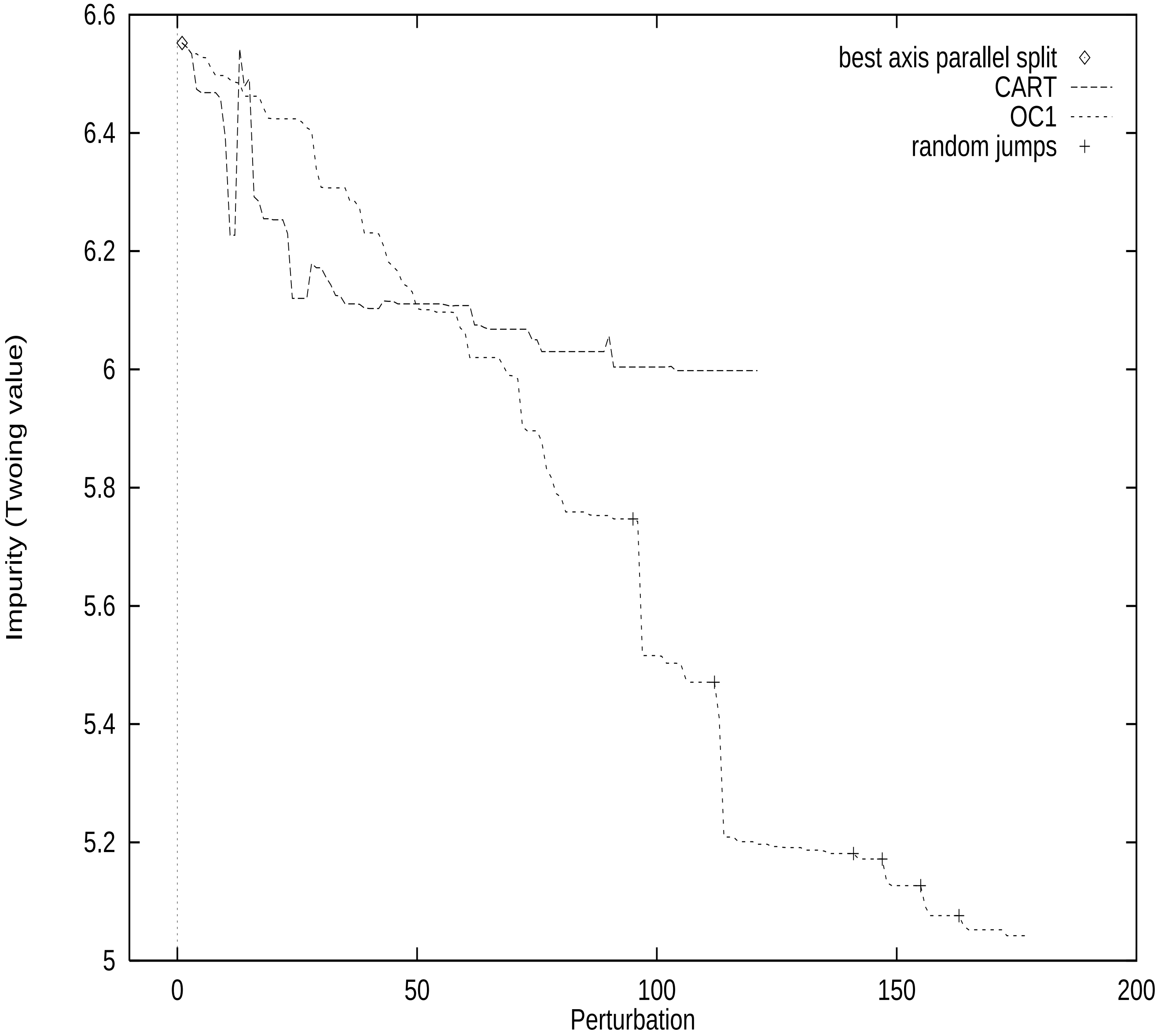

Figure 3.6: The local minima in the Dim Star-Galaxy data. The locations where OC1 uses random jumps to escape local minima are marked. Note that the impurity of OC1's perturbations is monotonically decreasing unlike that of CART-LC.

LMDT:

Another oblique decision tree algorithm, one that uses a very different approach from CART-LC, is the Linear Machine Decision Trees (LMDT) system [483, 48], which is a successor to the Perceptron Tree method [480, 482]. Each internal node in an LMDT tree is a Linear Machine [364]. The training algorithm presents examples repeatedly at each node until the linear machine converges. Because convergence cannot be guaranteed, LMDT uses heuristics to determine when the node has stabilized. To make the training stable even when the set of training instances is not linearly separable, a "thermal training" method [150] is used, similar to simulated annealing.

SADT:

A third system that creates oblique trees is Simulated Annealing of Decision Trees (SADT) [204] which, like OC1, uses randomization. SADT uses simulated annealing [252] to find good values for the coefficients of the hyperplane at each node of a tree. SADT first places a hyperplane in a canonical location, and then iteratively perturbs all the coefficients by small random amounts. Initially, when the temperature parameter is high, SADT accepts almost any perturbation of the hyperplane, regardless of how it changes the goodness score. However, as the system "cools down," only changes that improve the goodness of the split are likely to be accepted. Though SADT's use of randomization allows it to effectively avoid some local minima, it compromises on efficiency. It runs much slower than either CART-LC, LMDT or OC1, sometimes considering tens of thousands of hyperplanes at a single node before it finishes annealing.

Linear Programming:

An alternative way of finding splits is through the use of linear programming (LP). (See Section 2.3.2 for pointers to work using LP for tree construction.) Typically, LP methods would find a split by minimizing the distance of misclassified points to the decision boundary. In our experience, we found this approach to be very competitive, in terms of efficiency and effectiveness, with methods that optimize a discrete count-based goodness measure. However, there seem to be three significant problems with LP-based tree building methods.

- LP methods are unlikely to be robust to non-uniformly distributed noise in the data. We are currently experimenting with the LP formulations of [117, 25] to verify this hypothesis, and our preliminary results support the hypothesis.
- For some multimodal class distributions, impurity-based methods can "shave-off" homogeneous corners of the attribute space, successively reducing the problem size and complexity. LP-based methods instead attempt to find a split that is good for the whole data set, which may not exist. As a result, LP can produce overly large trees. This is illustrated in Figure 3.7, which shows the partitionings generated by OC1 and LP for the POL data set described in Section 3.4.3.

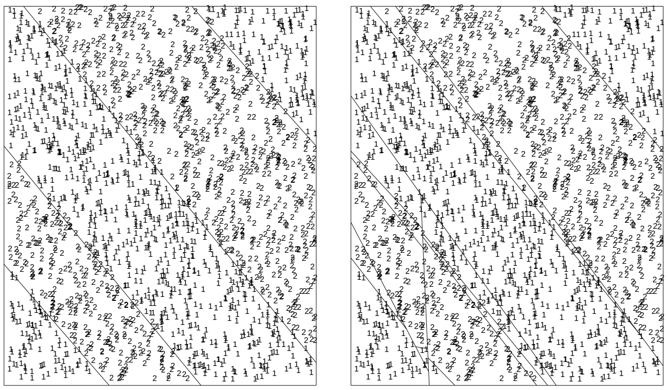

Figure 3.7: Trees induced on the POL data by OC1 and LP.

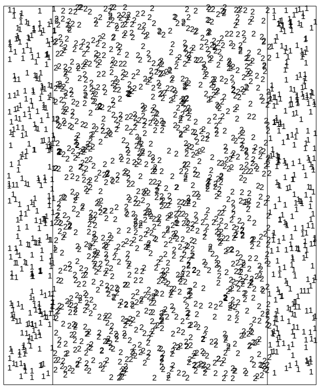

Figure 3.8: Linear programming formulations have trouble finding a split when the centres of the data sets to be separated coincide. The two LP formulations we experimented with did not find any split for this data set.

- Most LP-based methods produce null/useless solutions when the two sets to be separated have the same centre. Figure 3.8 shows a simple data set for which the LP formulations in [117, 25] fail to produce *any* split.

Our experimental section includes results showing how each of these methods compares to OC1. Our algorithm, OC1, uses deterministic hill climbing most of the time, ensuring computational efficiency. In addition, it uses two kinds of randomization to avoid local minima. By limiting the number of random choices, the algorithm is guaranteed to spend only polynomial time at each node in the tree. In addition, randomization itself has produced several benefits: for example, it means that the algorithm can produce many different trees for the same data set. This offers the possibility of a new family of classifiers: k-decision-tree algorithms, in which an example is classified by the majority vote of k trees (see Section 2.5.7 for pointers to work on using multiple trees). Finally, our experiments indicate that OC1 efficiently finds small, accurate decision trees for many different types of

```
To find a split of a set of examples $T$:
    Find the best axis-parallel split of $T$. Let $I$ be the impurity of this split.
    Repeat $R$ times:
        Choose a random hyperplane $H$.
        (For the first iteration, initialize $H$ to be the best axis-parallel split.)
        Step 1: Until the impurity measure does not improve, do:
            Perturb each of the coefficients of $H$ in sequence.
        Step 2: Repeat at most $J$ times:
            Choose a random direction and attempt to perturb $H$ in that direction.
            If this reduces the impurity of $H$, go to Step 1.
        Let $I_1$ = the impurity of $H$. If $I_1 < I$, then set $I = I_1$.
    Output the split corresponding to $I$.
```

Figure 3.9: Overview of the OC1 algorithm for a single node of a decision tree.

classification problems.

3.3 Oblique Classifier 1 (OC1)

In this section we discuss details of the oblique decision tree induction system OC1. As part of this description, we include:

- the method for finding coefficients of a hyperplane at each tree node,
- methods for computing the impurity or goodness of a hyperplane,
- a tree pruning strategy, and
- methods for coping with missing and irrelevant attributes.

Section 3.3.1 focuses on the most complicated of these algorithmic details; i.e. the question of how to find a hyperplane that splits a given set of instances into two reasonably "pure" non-overlapping subsets. This randomized perturbation algorithm is the main novel contribution of OC1. Figure 3.9 summarizes the basic OC1 algorithm, used at each node of a decision tree. This figure will be explained further in the following sections.

3.3.1 Perturbation algorithm

OC1 imposes no restrictions on the orientation of the hyperplanes. However, in order to be at least as powerful as standard DT methods, it first finds the best axis-parallel (univariate) split at a node before looking for an oblique split. OC1 uses an oblique split only when it improves over the best axis-parallel split.[25]

[25] As pointed out in [44, Chapter 5], it does not make sense to use an oblique split when the number of examples at a node n is less than or almost equal to the number of features d, because

The search strategy for the space of possible hyperplanes is defined by the procedure that perturbs the current hyperplane H to a new location. Because there are an exponential number of distinct ways to partition the examples with a hyperplane, any procedure that simply enumerates all of them will be unreasonably costly. The two main alternatives considered in the past have been simulated annealing, used in the SADT system [204], and deterministic heuristic search, as in CART-LC [44]. OC1 combines these two ideas, using heuristic search until it finds a local minimum, and then using a non-deterministic search step to get out of the local minimum. (The non-deterministic step in OC1 is *not* simulated annealing, however.)

We will start by explaining how we perturb a hyperplane to split the training set T at a node of the decision tree. Let n be the number of examples in T, d be the number of attributes (or dimensions) for each example, and k be the number of categories. Then we can write $T_j = (x_{j1}, x_{j2}, \ldots, x_{jd}, C_j)$ for the jth example from the training set T, where x_{ji} is the value of attribute i and C_j is the category label. As defined in Eq. 3.1, the equation of the current hyperplane H at a node of the decision tree is written as $\sum_{i=1}^{d}(a_i x_i) + a_{d+1} = 0$. If we substitute a point (an example) T_j into the equation for H, we get $\sum_{i=1}^{d}(a_i x_{ji}) + a_{d+1} = V_j$, where the sign of V_j tells us whether the point T_j is above or below the hyperplane H; i.e., if $V_j > 0$, then T_j is above H. If H splits the training set T perfectly, then all points belonging to the same category will have the same sign for V_j. i.e., sign(V_i) = sign(V_j) iff category(T_i) = category(T_j).

OC1 adjusts the coefficients of H individually, finding a locally optimal value for one coefficient at a time. This key idea was introduced by Breiman *et al.*. It works as follows. Treat the coefficient a_m as a variable, and treat all other coefficients as constants. Then V_j can be viewed as a function of a_m. In particular, the condition that T_j is above H is equivalent to

$$V_j > 0$$

$$a_m > \frac{a_m x_{jm} - V_j}{x_{jm}} \overset{\text{def}}{=} U_j \qquad (3.2)$$

assuming that $x_{jm} > 0$, which we ensure by normalization. Using this definition of U_j, the point T_j is above H if $a_m > U_j$, and below otherwise. By plugging all the points from T into this equation, we will obtain n constraints on the value of a_m.

The problem then is to find a value for a_m that satisfies as many of these constraints as possible. (If all the constraints are satisfied, then we have a perfect split.) This problem is easy to solve optimally: simply sort all the values U_j, and consider setting a_m to the midpoint between each pair of different values. This is illustrated in Figure 3.10. In the figure, the categories are indicated by font size; the larger U_i's belong to one category, and the smaller to another. For each distinct placement of the coefficient a_m, OC1 computes the impurity of the resulting split; e.g., for the location between U_6 and U_7 illustrated here, two examples on the left and one example on the right would be misclassified (see Section 3.3.4

the data *underfits* the concept. By default, OC1 uses only axis-parallel splits at tree nodes at which $n < 2d$. The user can vary this threshold.

Figure 3.10: Finding the optimal value for a single coefficient a_m. Large U's correspond to examples in one category and small u's to another.

```
Perturb(H,m)
  For $j = 1, \ldots, n$
      Compute $U_j$ (Eq. 3.2)
  Sort $U_1, \ldots, U_n$ in non-decreasing order.
  $a'_m$ = best univariate split of the sorted $U_j$s.
  $H_1$ = result of substituting $a'_m$ for $a_m$ in $H$.
  If $(impurity(H_1) < impurity(H))$
      { $a_m = a'_m$ ; $P_{move} = P_{stag}$ }
  Else if $(impurity(H) = impurity(H_1))$
      { $a_m = a'_m$ with probability $P_{move}$
        $P_{move} = P_{move} - 0.1 * P_{stag}$ }
```

Figure 3.11: Perturbation algorithm for a single coefficient a_m.

for different ways of computing impurity). As the figure illustrates, the problem is simply to find the best one-dimensional split of the Us, which requires considering just $n-1$ values for a_m. The value a'_m obtained by solving this one-dimensional problem is then considered as a replacement for a_m. Let H_1 be the hyperplane obtained by "perturbing" a_m to a'_m. If H has better (lower) impurity than H_1, then H_1 is discarded. If H_1 has lower impurity, H_1 becomes the new location of the hyperplane. If H and H_1 have identical impurities, then H_1 replaces H with probability P_{stag}.[26] Figure 3.11 contains pseudocode for our perturbation procedure.

Now that we have a method for locally improving a coefficient of a hyperplane, we need to decide which of the $d+1$ coefficients to pick for perturbation. We experimented with three different methods for choosing which coefficient to adjust, namely, sequential,

[26] The parameter P_{stag}, denoting "stagnation probability", is the probability that a hyperplane is perturbed to a location that does not change the impurity measure. To prevent the impurity from remaining stagnant for a long time, P_{stag} decreases by a constant amount each time OC1 makes a "stagnant" perturbation; thus only a constant number of such perturbations will occur at each node. This constant can be set by the user. P_{stag} is reset to 1 every time the global impurity measure is improved.

best first and random.

```
Seq:   Repeat until none of the coefficient values is modified in the For loop:
           For $i = 1$ to $d$, Perturb($H, i$)
Best:  Repeat until coefficient $m$ remains unmodified:
           $m$ = coefficient which when perturbed, results in the
                 maximum improvement of the impurity measure.
           Perturb($H, m$)
R-50:  Repeat a fixed number of times (50 in our experiments):
           $m$ = random integer between 1 and $d + 1$
           Perturb($H, m$)
```

Our previous experiments [350] indicated that the order of perturbation of the coefficients does not affect the classification accuracy as much as other parameters, especially the randomization parameters (see below). Since none of these orders was uniformly better than any other, we used sequential (Seq) perturbation for all the experiments reported in Section 3.4.

3.3.2 Randomization

The perturbation algorithm halts when the split reaches a local minimum of the impurity measure. For OC1's search space, a local minimum occurs when no perturbation of any single coefficient of the current hyperplane will decrease the impurity measure. (Of course, a local minimum may also be a global minimum.) We have implemented two ways of attempting to escape local minima: perturbing the hyperplane with a random vector, and re-starting the perturbation algorithm with a different random initial hyperplane.

The technique of perturbing the hyperplane with a random vector works as follows. When the system reaches a local minimum, it chooses a random vector to add to the coefficients of the current hyperplane. It then computes the optimal amount by which the hyperplane should be perturbed along this random direction. To be more precise, when a hyperplane $H = \sum_{i=1}^{d} a_i x_i + a_{d+1}$ cannot be improved by deterministic perturbation, OC1 repeats the following loop J times (where J is a user-specified parameter, set to 5 by default).

- Choose a random vector $R = (r_1, r_2, \ldots, r_{d+1})$. [27]

[27] One needs to be careful in choosing a random vector. If the components of the vector are randomly chosen independently of each other, the resulting random vectors will not be uniformly distributed. Their distribution will be biased towards the corners of a hyperrectangle, and the bias increases with increasing dimensionality. This implies that such a strategy to choose random vectors may be ineffective in escaping local minima for higher dimensional data. A better strategy is to choose the polar coordinates of the point at random, independently of each other, and then convert them to the rectilinear coordinates.

- Let α be the amount by which we want to perturb H in the direction R. In other words, let $H_1 = \sum_{i=1}^{d} (a_i + \alpha r_i)x_i + (a_{d+1} + \alpha r_{d+1})$.
- Find the optimal value for α.
- If the hyperplane H_1 thus obtained decreases the overall impurity, replace H with H_1, exit this loop and begin the deterministic perturbation algorithm for the individual coefficients.

Note that we can treat α as the only variable in the equation for H_1. Therefore each of the n examples in T, if plugged into the equation for H_1, imposes a constraint on the value of α. OC1 therefore can use its coefficient perturbation method (see Section 3.3.1) to compute the best value of α. If J random jumps fail to improve the impurity, OC1 halts and uses H as the split for the current tree node.

An intuitive way of understanding this random jump is to look at the dual space in which the algorithm is actually searching. Note that the equation $H = \sum_{i=1}^{d} a_i x_i + a_{d+1}$ defines a space in which the axes are the coefficients a_i rather than the attributes x_i. Every point in this space defines a distinct hyperplane in the original formulation. The deterministic algorithm used in OC1 picks a hyperplane and then adjusts coefficients one at a time. Thus in the dual space, OC1 chooses a point and perturbs it by moving it parallel to the axes. The random vector R represents a random *direction* in this space. By finding the best value for α, OC1 finds the best distance to adjust the hyperplane in the direction of R.

Note that this additional perturbation in a random direction does not significantly increase the time complexity of the algorithm (see Section 3.3.3). We found in our experiments that even a single random jump, when used at a local minimum, proves to be very helpful. Classification accuracy improved for every one of our data sets when such perturbations were made. See Section 3.4.3 for some examples.

The second technique for avoiding local minima is a variation on the idea of performing multiple local searches. The technique of multiple local searches is a natural extension to local search, and has been widely mentioned in the optimization literature (see Roth [412] for an early example). Because most of the steps of our perturbation algorithm are deterministic, the initial hyperplane largely determines which local minimum will be encountered first. Perturbing a single initial hyperplane is thus unlikely to lead to the best split of a given data set. In cases where the random perturbation method fails to escape from local minima, it may be helpful to simply start afresh with a new initial hyperplane. We use the word *restart* to denote one run of the perturbation algorithms, at one node of the decision tree, using one random initial hyperplane.[28] That is, a restart cycles through and perturbs the coefficients one at a time and then tries to perturb the hyperplane in a random direction when the algorithm reaches a local minimum. If this last perturbation reduces the impurity, the algorithm goes back to perturbing the coefficients one at a time.

[28] The first run through the algorithm at each node always begins at the location of the best axis-parallel hyperplane; all subsequent restarts begin at random locations.

The restart ends when neither the deterministic local search nor the random jump can find a better split. One of the optional parameters to OC1 specifies how many restarts to use. If more than one restart is used, then the best hyperplane found thus far is always saved. In all our experiments, the classification accuracies increased with more than one restart. Accuracy tended to increase up to a point and then level off (after about 20–50 restarts, depending on the domain). Overall, the use of multiple initial hyperplanes substantially improved the quality of the decision trees found (see Section 3.4.3 for some examples).

Our initial hyperplanes are chosen uniformly randomly. Goodrich *et al.*[176] evaluated the benefits of choosing the initial hyperplanes in a *data sensitive* manner. Their method chooses d (= dimensionality) points randomly from the input, fits a hyperplane to them, and uses it as an initial hyperplane for a restart of OC1. In addition to making it possible to analytically quantify how close the perturbation algorithm comes to the optimal split, data sensitive restarts produce trees of almost identical quality to OC1's default restarts.

Best Axis-Parallel Split. It is clear that axis-parallel splits are more suitable for some data distributions than oblique splits. To take into account such distributions, OC1 computes the best axis-parallel split *and* an oblique split at each node, and then picks the better of the two.[29] Calculating the best axis-parallel split takes an additional $O(dn \log n)$ time, and so does not increase the asymptotic time complexity of OC1. As a simple variant of the OC1 system, the user can opt to "switch off" the oblique perturbations, thus building an axis-parallel tree on the training data. Section 3.4.2 empirically demonstrates that this axis-parallel variant of OC1 compares favorably with existing axis-parallel algorithms.

3.3.3 Computational complexity

By carefully combining hill-climbing and randomization, OC1 ensures a an efficient worst case time for inducing a decision tree. In the following, we show that for a data set with n examples (points) and d attributes per example, OC1 uses at most $O(dn^2 \log n)$ time for inducing the tree. We assume $n > d$ for our analysis.

For the analysis here, we assume the coefficients of a hyperplane are adjusted in sequential order (the Seq method described in Section 3.3.1). The number of restarts at a node will be r, and the number of random jumps tried will be j. Both r and j are constants, fixed in advance of running the algorithm.

Initializing the hyperplane to a random position takes just $O(d)$ time. We need to consider first the maximum amount of work OC1 can do before it finds a new location for the hyperplane. Then we need to consider how many times it can move the hyperplane.

1. Attempting to perturb the first coefficient (a_1) takes $O(dn + n \log n)$ time. Computing U_i's for all the points (equation 3.2) requires $O(dn)$ time, and sorting the U_i's takes

[29] Sometimes a simple axis-parallel split is preferable to an oblique split, even if the oblique split has slightly lower impurity. The user can specify such a bias as an input parameter to OC1.

$O(n \log n)$. This gives us $O(dn + n \log n)$ work.

2. If perturbing a_1 does not improve things, we try to perturb a_2. Computing all the new U_i's will take just $O(n)$ time because only one term is different for each U_i. Re-sorting will take $O(n \log n)$, so this step takes $O(n) + O(n \log n) = O(n \log n)$ time.

3. Likewise $a_3, \ldots, a_d$ will each take $O(n \log n)$ additional time, assuming we still have not found a better hyperplane after checking each coefficient. Thus the total time to cycle through and attempt to perturb all these additional coefficients is $(d-1) * O(n \log n) = O(dn \log n)$.

4. Summing up, the time to cycle through all coefficients is $O(dn \log n) + O(dn + n \log n) = O(dn \log n)$.

5. If none of the coefficients improved the split, then we attempt to make up to j random jumps. Since j is a constant, we will just consider $j = 1$ for our analysis. This step involves choosing a random vector and running the perturbation algorithm to solve for α, as explained in Section 3.3.2. As before, we need to compute a set of U_i's and sort them, which takes $O(dn + n \log n)$ time. Because this amount of time is dominated by the time to adjust all the coefficients, the total time so far is still $O(dn \log n)$. This is the most time OC1 can spend at a node before either halting or finding an improved hyperplane.

6. Assuming OC1 is using the Sum Minority or Max Minority error measure (Section 3.3.4), it can only reduce the impurity of the hyperplane n times. This is clear because each improvement means one more example will be correctly classified by the new hyperplane. Thus the total amount of work at a node is limited to $n * O(dn \log n) = O(dn^2 \log n)$. In practice, we have found that the number of improvements per node is much smaller than n.

7. The measures Information Gain, Gini Index and Twoing Rule (Section 3.3.4) can take $\theta(n^2)$ values at a node with n points belonging to one of two classes. (More precisely, information gain can take on $(n+2)^2/8$ distinct values. Gini index and Twoing rule can take on fewer values, as they have more inherent symmetry than information gain, but the upper bound is $\theta(n^2)$ for all three measures.) So, the time bounds derived for max minority and sum minority increase by a linear factor for these measures. The bounds derived are not applicable to a measure that, for example, uses the distances of mis-classified objects to the hyperplane.

Assuming that OC1 only adjusts a hyperplane when it improves the impurity measure, it will do $O(dn^2 \log n)$ work in the worst case. However, OC1 allows a certain number of adjustments to the hyperplane that do not improve the impurity, although it will never accept a change that worsens the impurity. The number allowed is determined by a constant known as "stagnant-perturbations". Let this value be s. This works as follows.

Each time OC1 finds a new hyperplane that improves on the old one, it resets a counter to zero. It will move the new hyperplane to a different location that has *equal* impurity at most s times. After each of these moves it repeats the perturbation algorithm. Whenever impurity is reduced, it re-starts the counter and again allows s moves to equally good locations. Thus it is clear that this feature just increases the worst-case complexity of OC1 by a constant factor, s.

Finally, note that the overall cost of OC1 is also $O(dn^2 \log n)$, i.e., this is an upper bound on the total running time of OC1 independent of the size of the tree it ends up creating. (This upper bound applies to Sum Minority and Max Minority; an open question is whether a similar upper bound can be proven for Information Gain or the Gini Index.) Thus the worst-case asymptotic complexity of our system is comparable to that of systems that construct axis-parallel decision trees, which have $O(dn^2)$ worst-case complexity. To sketch the intuition that leads to this bound, let G be the total impurity summed over all leaves in a partially constructed tree (i.e., the sum of currently misclassified points in the tree). Now observe that each time we run the perturbation algorithm on any node in the tree, we either halt or improve G by at least one unit. The worst-case analysis for one node is realized when the perturbation algorithm is run once for every one of the n examples, but when this happens, there would no longer be any mis-classified examples and the tree would be complete.

3.3.4 Other details

Impurity measures

OC1 attempts to divide the d-dimensional attribute space into homogeneous regions; i.e., regions that contain examples from just one category. The goal of adding new nodes to a tree is to split up the sample space so as to minimize the "impurity" of the training set. Some algorithms measure "goodness" instead of impurity, the difference being that goodness values should be maximized while impurity should be minimized. Many different measures of impurity have been studied (Section 2.3.1).

The OC1 system is designed to work with a large class of impurity measures. Stated simply, if the impurity measure uses only the counts of examples belonging to every category on both sides of a split, then OC1 can use it. (See Chapter 7 for ways of mapping other kinds of impurity measures to this class of impurity measures.) The user can plug in any impurity measure that fits this description. The OC1 implementation includes six impurity measures. Though all six of the measures have been defined elsewhere in the literature, in some cases we have made slight modifications that are defined precisely below.

In each of the following definitions, the set of examples T at the node about to be split contains n (> 0) instances that belong to one of k categories. (Initially this set is the entire training set.) A hyperplane H divides T into two non-overlapping subsets T_L and T_R (i.e., left and right). L_j and R_j are the number of instances of category j in T_L and T_R respectively. All the impurity measures initially check to see if T_L and T_R are homogeneous (i.e., all examples belong to the same category), and if so return minimum (zero) impurity.

1. **Information Gain:** This measure of information gained from a particular split was

popularized in machine learning by Quinlan (1986). Quinlan's definition makes information gain a goodness measure; i.e., something to maximize. Because OC1 attempts to minimize whatever impurity measure it uses, we use the reciprocal of the standard value of information gain in the OC1 implementation.

2. **Gini Index:** The Gini criterion (or index) was proposed for decision trees by Breiman *et al.*(1984). The Gini Index as originally defined measures the probability of misclassification of a set of instances, rather than the impurity of a split. We implement the following variation:

$$\text{GiniL} = 1.0 - \sum_{i=1}^{k} (L_i/|T_L|)^2$$

$$\text{GiniR} = 1.0 - \sum_{i=1}^{k} (R_i/|T_R|)^2$$

$$\text{Impurity} = (|T_L| * \text{GiniL} + |T_R| * \text{GiniR})/n$$

where GiniL is the Gini Index on the "left" side of the hyperplane and GiniR is that on the right.

3. **Twoing Rule:** The Twoing Rule was first proposed by Breiman et al. (1984). The value to be computed is defined as:

$$\text{TwoingValue} = (|T_L|/n) * (|T_R|/n) * (\sum_{i=1}^{k} |L_i/|T_L| - R_i/|T_R||)^2$$

where $|T_L|$ ($|T_R|$) is the number of examples on the left (right) of a split at node T, n is the number of examples at node T, and L_i (R_i) is the number of examples in category i on the left (right) of the split. The TwoingValue is actually a goodness measure rather than an impurity measure. Therefore OC1 attempts to minimize the reciprocal of this value.

4. **Max Minority:** The measures Max Minority, Sum Minority and Sum Of Variances were defined in the context of decision trees by Heath, Kasif, and Salzberg [204]. Max Minority has the theoretical advantage that a tree built minimizing this measure will have depth at most $\log n$. Our experiments indicated that this is not a great advantage in practice: seldom do other impurity measures produce trees substantially deeper than those produced with Max Minority. The definition is:

$$\text{MinorityL} = \sum_{i=1, i \neq \max L_i}^{k} L_i$$

$$\text{MinorityR} = \sum_{i=1, i \neq \max R_i}^{k} R_i$$

$$\text{Max Minority} = \max(\text{MinorityL}, \text{MinorityR})$$

5. **Sum Minority:** If MinorityL and MinorityR are defined as for the Max Minority measure, then Sum Minority is just the sum of these two values. This measure is the simplest way of quantifying impurity, as it simply counts the number of misclassified instances. Though Sum Minority performs well on some domains, it has some obvious flaws. As one example, consider a domain in which $n = 100, d = 1$, and $k = 2$ (i.e., 100 examples, 1 numeric attribute, 2 classes). Suppose that when the examples are sorted according to the single attribute, the first 50 instances belong to category 1, followed by 24 instances of category 2, followed by 26 instances of category 1. Then *all* possible splits for this distribution have a sum minority of 24. Therefore it is impossible when using Sum Minority to distinguish which split is preferable, although splitting at the alternations between categories is clearly better. [30]

6. **Sum Of Variances:** [31] The definition of this measure is:

$$\text{VarianceL} = \sum_{i=1}^{|T_L|} (Cat(T_{L_i}) - \sum_{j=1}^{|T_L|} Cat(T_{L_j})/|T_L|)^2$$

$$\text{VarianceR} = \sum_{i=1}^{|T_R|} (Cat(T_{R_i}) - \sum_{j=1}^{|T_R|} Cat(T_{R_j})/|T_R|)^2$$

$$\text{Sum of Variances} = \text{VarianceL} + \text{VarianceR}$$

where Cat(T_i) is the category of instance T_i. As this measure is computed using the actual class labels, it is easy to see that the impurity computed varies depending on how numbers are assigned to the classes. For instance, if T_1 consists of 10 points of category 1 and 3 points of category 2, and if T_2 consists of 10 points of category 1 and 3 points of category 5, then the Sum Of Variances values are different for T_1 and T_2. To avoid this problem, OC1 uniformly reassigns category numbers according to the frequency of occurrence of each category at a node before computing the Sum Of Variances.

Our experiments indicated that, on average, Information Gain, Gini Index and the Twoing Rule perform better than the other three measures for both axis-parallel and oblique trees. The Twoing Rule is the current default impurity measure for OC1, and it was used in all of the experiments reported in Section 3.4. There are, however, artificial data sets for which Sum Minority and/or Max Minority perform much better than the rest of the measures. For instance, Sum Minority easily induces the exact tree for the POL data set described in Section 3.4.3, while all other methods have difficulty finding the best tree.

[30] Lubinsky [298] also used this measure for tree construction, referring to it as *inaccuracy*. He suggested an improvement to inaccuracy called *weighted inaccuracy*.

[31] Sum Of Variances was called Sum of Impurities by Heath *et al.*. The earliest use of this measure we found was in the Automatic Interaction Detection (AID) program [139].

Pruning

Virtually all decision tree induction systems prune the trees they create in order to avoid overfitting the data. Many studies have found that judicious pruning results in both smaller and more accurate classifiers, for decision trees as well as other types of machine learning systems (see Section 2.4.1). For the OC1 system we implemented an existing pruning method, but note that any tree pruning method will work fine within OC1. Based on the experimental evaluations of Mingers [324] and other work, we chose Breiman *et al.*'s Cost Complexity (CC) pruning [44] as the default pruning method for OC1. This method, which is also called Error Complexity or Weakest Link pruning, requires a separate pruning set. The pruning set can be a randomly chosen subset of the training set, or it can be approximated using cross validation. OC1 randomly chooses 10% (the default value) of the training data to use for pruning. In the experiments reported below, we only used this default value.

Briefly, the idea behind CC pruning is to create a set of trees of decreasing size from the original, complete tree. All these trees are used to classify the pruning set, and accuracy is estimated from that. CC pruning then chooses the smallest tree whose accuracy is within k standard errors squared of the best accuracy obtained. When the 0-SE rule ($k = 0$) is used, the tree with highest accuracy on the pruning set is selected. When $k > 0$, smaller tree size is preferred over higher accuracy. For details of Cost Complexity pruning, see Breiman *et al.*[44] or Mingers [324].

Irrelevant attributes

Irrelevant attributes pose a significant problem for most machine learning methods. Decision tree algorithms, even axis-parallel ones, can be confused by too many irrelevant attributes (see Section 2.5.1 for pointers to existing work). Because oblique decision trees learn the coefficients of each attribute at a DT node, one might hope that the values chosen for each coefficient would reflect the relative importance of the corresponding attributes. Clearly, though, the process of searching for good coefficient values will be much more efficient when there are fewer attributes; the search space is much smaller. For this reason, oblique DT induction methods can benefit substantially by using a feature selection method (an algorithm that selects a subset of the original attribute set) in conjunction with the coefficient learning algorithm [44, 49].

Currently, OC1 does not have a built-in mechanism to select relevant attributes. However, it is easy to include any of several standard methods (e.g., stepwise forward selection or stepwise backward selection) or even an ad hoc method to select features before running the tree-building process. For example, in separate experiments on data from the Hubble Space Telescope (Section 6.1), we used feature selection methods as a preprocessing step to OC1, and reduced the number of attributes from 20 to 2. The resulting decision trees were both simpler and more accurate. Work is currently underway to incorporate an efficient feature selection technique into the OC1 system.

Regarding missing values, if an example is missing a value for any attribute, OC1 uses the mean value for that attribute. One can of course use other techniques for handling missing values, but those were not considered in this study.

3.4 Experiments

In this section, we present three experiments to support the following three claims, respectively.

1. OC1 compares favorably over a variety of real-world domains with several existing axis-parallel and oblique decision tree induction methods.
2. Randomization, both in the form of multiple local searches and random jumps, improves the quality of decision trees produced by OC1.
3. OC1 uses randomization sparingly, ensuring efficient search.

The experimental method used for all the experiments is described in Section 3.4.1. Sections 3.4.2, 3.4.3 and 3.4.4 describe experiments corresponding to the above three claims. Each experimental section begins with a description of the data sets, and then presents the experimental results and discussion.

3.4.1 The setup

We used five-fold cross validation (CV) in all our experiments to estimate classification accuracy. A k-fold CV experiment consists of the following steps.

1. Randomly divide the data into k equal-sized disjoint partitions.
2. For each partition, build a decision tree using all data outside the partition, and test the tree on the data in the partition.
3. Sum the number of correct classifications of the k trees and divide by the total number of instances to compute the classification accuracy. Report this accuracy and the average size of the k trees.

Each entry in Tables 3.1 and 3.2 is a result of ten 5-fold CV experiments; i.e., the result of tests that used 50 decision trees. Each of the ten 5-fold cross validations used a different random partitioning of the data. Each entry in the tables reports the mean and standard deviation of the classification accuracy, followed by the mean and standard deviation of the decision tree size (measured as the number of leaf nodes). Good results should have high values for accuracy, low values for tree size, and small standard deviations.

In addition to OC1, we also included in the experiments an axis-parallel version of OC1, which only considers axis-parallel hyperplanes. We call this version, described in Section 3.3.2, OC1-AP. In all our experiments, both OC1 and OC1-AP used the Twoing Rule (Section 3.3.4) to measure impurity. Other parameters to OC1 took their default values unless stated otherwise. (Defaults include the following: number of restarts at each node: 20. Number of random jumps attempted at each local minimum: 5. Order of coefficient perturbation: Sequential. Pruning method: Cost Complexity with the 0-SE rule, using 10% of the training set exclusively for pruning.)

In our comparison, we used the oblique version of the CART algorithm, CART-LC. We implemented our own version of CART-LC, following the description in Breiman *et al.*[44, Chapter 5]; however, there may be differences between our version and other versions of this system (note that CART-LC is not freely available). Our implementation of CART-LC measured impurity with the Twoing Rule and used 0-SE Cost Complexity pruning with a separate test set, just as OC1 does. We did not include any feature selection methods in CART-LC or in OC1, and we did not implement normalization. Because the CART coefficient perturbation algorithm may alternate indefinitely between two locations of a hyperplane (see Section 3.2), we imposed an arbitrary limit of 100 such perturbations before forcing the perturbation algorithm to halt.

We also included axis-parallel CART and C4.5 in our comparisons. We used the implementations of these algorithms from the IND 2.1 package [63]. The default cart0 and c4.5 "styles" defined in the package were used, without altering any parameter settings. The cart0 style uses the Twoing Rule and 0-SE cost complexity pruning with 10-fold cross validation. The pruning method, impurity measure and other defaults of the c4.5 style are the same as those described in Quinlan [398].

The last method we included in our comparisons is a modification of OC1 that uses linear programming (LP) to find the split at each node. We call this variation OC1-LP. To implement this method, we replaced OC1's hyperplane-finding routine with a routine that formulates and solves a LP problem. The LP formulation we used is the one suggested in [25]. [32] We used LOQO to solve the linear programs. LOQO is a linear and quadratic programming problem solver written by Robert J. Vanderbi, Program in Statistics and Operations Research, Princeton University. [33] Note that we have implemented the LP-formulation only for 2-class problems.

3.4.2 OC1 vs. existing tree methods

Table 3.1 compares the performance of OC1 to three well-known decision tree induction methods, OC1-AP and OC1-LP on six different real-world data sets. In the next section we will consider artificial data, for which the concept definition can be precisely characterized.

Description of data sets

Star/Galaxy Discrimination. Two of our data sets came from a large set of astronomical images collected by Odewahn *et al.*[367]. In their study, they used these images to

[32] Thanks to Kristin Bennett for providing the code, and for helpful discussions.

[33] Though LOQO is a commercial product, academic institutions can obtain a free copy for research purposes only. Contact Robert Vanderbi at *rvdb@jazz.princeton.edu*.

train artificial neural networks running the perceptron and back propagation algorithms. The goal was to classify each example as either "star" or "galaxy." Each image is characterized by 14 real-valued attributes, where the attributes were measurements defined by astronomers as likely to be relevant for this task. The objects in the image were divided by Odewahn *et al.*into "bright" and "dim" data sets based on the image intensity values, where the dim images are inherently more difficult to classify. (Note that the "bright" objects are only bright in relation to others in this data set. In actuality they are extremely faint, visible only to the most powerful telescopes.) The bright set contains 2462 objects and the dim set contains 4192 objects.

In addition to the results reported in Table 3.1, the following results have appeared on the Star/Galaxy data. Odewahn et al. (1992) reported accuracy of 99.8% accuracy on the bright objects, and 92.0% on the dim ones, although it should be noted that this study used a single training and test set partition. Heath [202] reported 99.0% accuracy on the bright objects using SADT, with an average tree size of 7.03 leaves. This study also used a single training and test set. Salzberg [422] reported accuracies of 98.8% on the bright objects, and 95.1% on the dim objects, using 1-Nearest Neighbor (1-NN) coupled with a feature selection method that reduces the number of features.

Breast Cancer Diagnosis. Mangasarian and Bennett have compiled data on the problem of diagnosing breast cancer to test several new classification methods [309, 25, 26]. This data represents a set of patients with breast cancer, where each patient was characterized by nine numeric attributes plus the diagnosis of the tumor as benign or malignant. The data set currently has 683 entries and is available from the UC Irvine machine learning repository [346]. Heath *et al.*[204] reported 94.9% accuracy on a subset of this data set (it then had only 470 instances), with an average decision tree size of 4.6 nodes, using SADT. Salzberg [421] reported 96.0% accuracy using 1-NN on the same (smaller) data set. Herman and Yeung [207] reported 99.0% accuracy using piece-wise linear classification, again using a somewhat smaller data set. Bennett and Mangasarian [25] reported 97.4% accuracy using their MSM1 algorithm, using a different experimentation method from the one we employ.

Classifying Irises. This is Fisher's famous iris data, which has been extensively studied in the statistics and machine learning literature. The data consists of 150 examples, where each example is described by four numeric attributes. There are 50 examples of each of three different types of iris flower. Weiss and Kapouleas [501] obtained accuracies of 96.7% and 96.0% on this data with back propagation and 1-NN, respectively. Note that Table 3.1 does not report results of OC1-LP on the Iris data. This is because we have implemented the LP-formulation for only two-class problems.

Housing Costs in Boston. This data set, also available as a part of the UCI ML repository, describes housing values in the suburbs of Boston as a function of 12 continuous attributes and 1 binary attribute [194]. The category variable (median value of owner-

Algorithm	Bright S/G	Dim S/G	Cancer	Iris	Housing	Diabetes
OC1	98.9±0.2	95.0±0.3	96.2±0.3	94.7±3.1	82.4±0.8	74.4±1.0
	4.3±1.0	13.0±8.7	2.8±0.9	3.1±0.2	6.1±3.0	5.4±3.8
CART-LC	98.8±0.2	92.8±0.5	95.3±0.6	93.5±2.9	81.4±1.2	73.7±1.2
	3.9±1.3	24.2±8.7	3.5±0.9	3.2±0.3	5.8±3.2	8.0±5.2
OC1-LP	99.2±0.1	95.5±0.1	96.7±0.3		85.8±0.9	75.5±0.9
	2.5±0.5	6.4±3.1	3.0±1.2		4.2±1.9	6.6±4.3
OC1-AP	98.1±0.2	94.0±0.2	94.5±0.5	92.7±2.4	81.8±1.0	73.8±1.0
	6.9±2.4	29.3±8.8	6.4±1.7	3.2±0.3	8.6±4.5	11.4±7.5
CART-AP	98.5±0.5	94.2±0.7	95.0±1.6	93.8±3.7	82.1±3.5	73.9±3.4
	13.9±5.7	30.4±10	11.5±7.2	4.3±1.6	15.1±10	11.5±9.1
C4.5	98.5±0.5	93.3±0.8	95.3±2.0	95.1±3.2	83.2±3.1	71.4±3.3
	14.3±2.2	77.9±7.4	9.8±2.2	4.6±0.8	28.2±3.3	56.3±7.9

Table 3.1: Comparison of OC1 and other decision tree induction methods on six different data sets. The first line for each method gives accuracies, and the second line gives average tree sizes.

occupied homes) is actually continuous, but we discretized it so that category = 1 if value < $21000, and 2 otherwise. For other uses of this data, see [20, 399].

Diabetes diagnosis. This data catalogs the presence or absence of diabetes among Pima Indian females, 21 years or older, as a function of eight numeric-valued attributes. The original source of the data is the National Institute of Diabetes and Digestive and Kidney Diseases, and it is now available in the UCI repository. Smith et al. [452] reported 76% accuracy on this data using their ADAP learning algorithm, using a different experimental method from that used here.

Discussion

The table shows that, for the six data sets considered here, OC1 consistently finds better trees than the original oblique CART method. Its accuracy was greater in all six domains, although the difference was significant (more than 2 standard deviations) only for the dim star/galaxy problem. The average tree sizes were roughly equal for five of the six domains, and for the dim stars and galaxies, OC1 found considerably smaller trees. These differences will be analyzed and quantified further by using artificial data, in the following section.

The oblique methods (OC1, OC1-LP and CART-LC) generally find much smaller trees than the axis-parallel methods. This difference can be quite striking for some domains—note, for example, that OC1 produced a tree with 13 nodes on average for the dim star/galaxy problem, while C4.5 produced a tree with 78 nodes, 6 times larger. Of course, in domains for which an axis-parallel tree is the appropriate representation, axis-parallel methods should compare well with oblique methods in terms of tree size. In fact, for the Iris data, all the methods found similar-sized trees.

OC1-LP has the highest accuracy among all the methods, for *all* the domains it was applied to. This result substantiates our claim that oblique decision trees are more accurate and concise in some domains. However, it is surprising that linear programming (LP) beats our randomized search method, considering that LP methods may be oversensitive to outliers (Section 3.2). One possible reason for this is that the underlying trees are small for all the data sets we used. The bright star/galaxy data is almost linearly separable and the cancer and iris data sets have very accurate trees that have just more than 3 leaf nodes. We are currently investigating the properties of OC1-LP in more detail.

3.4.3 Randomization helps OC1

In our second set of experiments, we examine more closely the effect of introducing randomized steps into the algorithm for finding oblique splits. Our experiments demonstrate that OC1's ability to produce an accurate tree from a set of training data is clearly enhanced by the two kinds of randomization it uses. More precisely, we use three artificial data sets (for which the underlying concept is known to the experimenters) to show that OC1's performance improves substantially when the deterministic hill climbing is augmented in any of three ways:

- with multiple restarts from random initial locations,
- with perturbations in random directions at local minima, or
- with both of the above randomization steps.

In order to find clear differences between algorithms, one needs to know that the concept underlying the data is indeed difficult to learn. For simple concepts (say, two linearly separable classes in 2-D), many different learning algorithms will produce very accurate classifiers, and therefore the advantages of randomization may not be detectable. It is known that many of the commonly-used data sets from the UCI repository are easy to learn with very simple representations [209]; therefore those data sets may not be ideal for our purposes. Thus we created a number of artificial data sets that present different problems for learning, and for which we know the "correct" concept definition. This allows us to quantify more precisely how the parameters of our algorithm affect its performance.

A second purpose of this experiment is to compare OC1's search strategy with that of two existing oblique decision tree induction systems – LMDT [48] and SADT [204]. We show that the quality of trees induced by OC1 is as good as, if not better than, that of the trees induced by these existing systems on three artificial domains. We also show that OC1 achieves a good balance between amount of effort expended in search and the quality of the tree induced.

Both LMDT and SADT used information gain for this experiment. However, we did not change OC1's default measure (the Twoing Rule) because we observed, in experiments not reported here, that OC1 with information gain does not produce significantly different results. The maximum number of successive, unproductive perturbations allowed at any node was set at 10000 for SADT. For all other parameters, we used default settings provided with the systems.

Description of artificial data

LS10 The LS10 data set has 2000 instances divided into two categories. Each instance is described by ten attributes x_1,...,x_{10}, whose values are uniformly distributed in the range [0,1]. The data is linearly separable with a 10-D hyperplane (thus the name LS10) defined by the equation $x_1 + x_2 + x_3 + x_4 + x_5 < x_6 + x_7 + x_8 + x_9 + x_{10}$. The instances were all generated randomly and labelled according to which side of this hyperplane they fell on. Because oblique DT induction methods intuitively should prefer a linear separator if one exists, it is interesting to compare the various search techniques on this data set where we know a separator exists. The task is relatively simple for lower dimensions, so we chose 10-dimensional data to make it more difficult.

POL This data set is shown in Figure 3.12. It has 2000 instances in two dimensions, again divided into two categories. The underlying concept is a set of four parallel oblique lines (thus the name POL), dividing the instances into five homogeneous regions. This concept is more difficult to learn than a single linear separator, but the minimal-size tree is still quite small.

RCB RCB stands for "rotated checker board"; this data set is also used in Chapter 7 for a different set of experiments. The data set, shown in Figure 3.12, has 2000 instances in 2-D, each belonging to one of eight categories. This concept is difficult to learn for any axis-parallel method, for obvious reasons. It is also quite difficult for oblique methods, for several reasons. The biggest problem is that the "correct" root node, as shown in the figure, does not separate out any class by itself. Some impurity measures (such as Sum Minority) will fail miserably on this problem, although others (e.g., the Twoing Rule) work much better. Another problem is that a deterministic coefficient perturbation algorithm can get stuck in local minima in many places on this data set.

Table 3.2 summarizes the results of this experiment in three smaller tables, one for each data set. In each smaller table, we compare four variants of OC1 with LMDT and SADT. The different results for OC1 were obtained by varying both the number of restarts and the number of random jumps. When random jumps were used, up to twenty random jumps were tried at each local minimum. As soon as one was found that improved the impurity of the current hyperplane, the algorithm moved the hyperplane and started running the deterministic perturbation procedure again. If none of the 20 random jumps improved the impurity, the search halted and further restarts (if any) were tried. The same

Linearly Separable 10-D (LS10) data

R:J	Accuracy	Size	Hyperplanes
0:0	89.8±1.2	67.0±5.8	2756
0:20	91.5±1.5	55.2±7.0	3824
20:0	95.0±0.6	25.6±2.4	24913
20:20	97.2±0.7	13.9±3.2	30366
LMDT	99.7±0.2	2.2±0.5	9089
SADT	95.2±1.8	15.5±5.7	349067

Parallel Oblique Lines (POL) data

R:J	Accuracy	Size	Hyperplanes
0:0	98.3±0.3	21.6±1.9	164
0:20	99.3±0.2	9.0±1.0	360
20:0	99.1±0.2	14.2±1.1	3230
20:20	99.6±0.1	5.5±0.3	4852
LMDT	89.6±10.2	41.9±19.2	1732
SADT	99.3±0.4	8.4±2.1	85594

Rotated Checker Board (RCB) data

R:J	Accuracy	Size	Hyperplanes
0:0	98.4±0.2	35.5±1.4	573
0:20	99.3±0.3	19.7±0.8	1778
20:0	99.6±0.2	12.0±1.4	6436
20:20	99.8±0.1	8.7±0.4	11634
LMDT	95.7±2.3	70.1±9.6	2451
SADT	97.9±1.1	32.5±4.9	359112

Table 3.2: The effect of randomization in OC1. The first column, labelled R:J, shows the number of restarts (R) followed by the maximum number of random jumps (J) attempted by OC1 at each local minimum. Results with LMDT and SADT are included for comparison after the four variants of OC1. Size is average tree size measured by the number of leaf nodes. The third column shows the average number of hyperplanes each algorithm considered while building one tree.

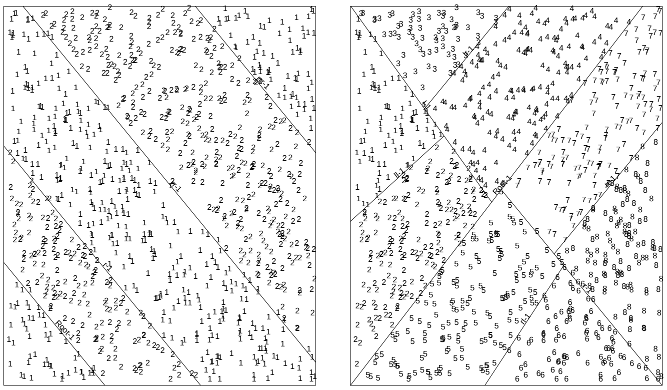

Figure 3.12: The POL and RCB data sets

training and test partitions were used for all methods for each cross-validation run (recall that the results are an average of ten 5-fold CVs). The trees were not pruned for any of the algorithms, because the data were noise-free and furthermore the emphasis was on search.

Table 3.2 also includes the number of hyperplanes considered by each algorithm while building a complete tree. Note that for OC1 and SADT, the number of hyperplanes considered is generally much larger than the number of perturbations actually made, because both these algorithms compare newly generated hyperplanes to existing hyperplanes before adjusting an existing one. Nevertheless, this number is a good estimate of much effort each algorithm expends, because every new hyperplane must be evaluated according to the impurity measure. For LMDT, the number of hyperplanes considered is identical to the actual number of perturbations.

Discussion

The OC1 results here are quite clear. The first line of each table, labelled 0:0, gives the accuracies and tree sizes when no randomization is used — this variant is very similar to the CART-LC algorithm. As we increase the use of randomization, accuracy increases while tree size decreases, which is exactly the result we had hoped for when we decided to introduce randomization into the method.

Looking more closely at the tables, we can ask about the effect of random jumps alone. This is illustrated in the second line (0:20) of each table, which attempted up to 20 random jumps at each local minimum and no restarts. Accuracy increased by 1-2% on each domain, and tree size decreased dramatically, roughly by a factor of two, in the POL and RCB domains. Note that because there is no noise in these domains, very high accuracies should be expected. Thus increases of more than a few percent in accuracy are not possible.

Looking at the third line of each sub-table in Table 3.2, we see the effect of multiple restarts on OC1. With 20 restarts but no random jumps to escape local minima, the improvement is even more noticeable for the LS10 data than when random jumps alone were used. For this data set, accuracy jumped significantly, from 89.8 to 95.0%, while tree size dropped from 67 to 26 nodes. For the POL and RCB data, the improvements were comparable to those obtained with random jumps. For the RCB data, tree size dropped by a factor of 3 (from 36 leaf nodes to 12 leaf nodes) while accuracy increased from 98.4 to 99.6%.

The fourth line of each table shows the effect of both the randomized steps. Among the OC1 entries, this line has both the highest accuracies and the smallest trees for all three data sets, so it is clear that randomization is a big win for these kinds of problems. In addition, note that the smallest tree for the RCB data should have eight leaf nodes, and OC1's average trees, without pruning, had just 8.7 leaf nodes. It is clear that for this data set, which we thought was the most difficult one, OC1 came very close to finding the optimal tree on nearly every run. (Recall that numbers in the table are the average of 10 5-fold CV experiments; i.e., an average of 50 decision trees.) The LS10 data show how difficult it can be to find a very simple concept in higher dimensions—the optimal tree there is just a single hyperplane (two nodes), but OC1 was unable to find it with the current parameter settings.[34] The POL data required a minimum of 5 leaf nodes, and OC1 found this minimal-size tree most of the time, as can be seen from the table. Although not shown in the Table, OC1 using Sum Minority performed better for the POL data than the Twoing Rule or any other impurity measure; i.e., it found the correct tree using less time.

The results of LMDT and SADT on this data lead to some interesting insights. Not surprisingly, LMDT does very well on the linearly separable (LS10) data, and does not require an inordinate amount of search. Clearly, if the data is linearly separable, one should use a method such as LMDT or linear programming. OC1 and SADT have difficulty finding the linear separator, although in our experiments OC1 did eventually find it, given sufficient time.

On the other hand, for both of the non-linearly separable data sets, LMDT produces much larger trees that are significantly less accurate than those produced by OC1 and SADT. Even the deterministic variant of OC1 (using zero restarts and zero random jumps) outperforms LMDT on these problems, with much less search.

Although SADT sometimes produces very accurate trees, its main weakness was the enormous amount of search time it required, roughly 10-20 times greater than OC1 even using the 20:20 setting. One explanation of OC1's advantage is its use of directed search, as opposed to the strictly random search used by simulated annealing. Overall, Table 3.2 shows that OC1's use of randomization was quite effective for the non-linearly separable data.

It is natural to ask *why* randomization helps OC1 in the task of inducing decision trees. Researchers in combinatorial optimization have observed that randomized search usually succeeds when the search space holds an abundance of good solutions [186]. Furthermore, randomization can improve upon deterministic search when many of the local

[34] In a separate experiment, we found that OC1 consistently finds the linear separator for the LS10 data when 10 restarts and 200 random jumps are used.

maxima in a search space lead to poor solutions. In OC1's search space, a local maximum is a hyperplane that cannot be improved by the deterministic search procedure, and a "solution" is a complete decision tree. If a significant fraction of local maxima lead to bad trees, then algorithms that stop at the first local maximum they encounter will perform poorly. Because randomization allows OC1 to consider many different local maxima, if a modest percentage of these maxima lead to good trees, then it has a good chance of finding one of those trees. Our experiments with OC1 thus far indicate that the space of oblique hyperplanes usually contains numerous local maxima, and that a substantial percentage of these locally good hyperplanes lead to good decision trees.

3.4.4 Different kinds of perturbations

In this experiment, we examine the relative effectiveness of OC1's three types of perturbations: hill climbing, jumps in random directions and stagnant perturbations. Our aim is to see how often each kind of perturbation is used. If either random jumps or stagnant perturbations are being used excessively, that will indicate that OC1 might be wasting search. On the other hand, if most of the time is being spent in hill climbing perturbations, with random jumps and stagnant perturbations being used judiciously, that will indicate that OC1 combines the strengths of methods like CART-LC and SADT.

We use the LS10 data set (Section 3.4.3) for this experiment. As the aim is to evaluate the search and not tree quality, we build only one internal node trees — we start with a random hyperplane and apply OC1's perturbation algorithm until no more perturbations can be made. We don't use any restarts. We use the sequential method of perturbation, trying a maximum of 100 random jumps at every local maximum. If none of the 100 random jumps are effective in finding a better position for the hyperplane, we stop the search. Else, we go back to the hill climbing procedure. After every perturbation, we record the resulting impurity (information gain) value and the type of the perturbation.

Figure 3.13 shows how the impurity value changes in a typical application of OC1's perturbation procedure. The X-axis shows the serial number of the perturbation, and the Y-axis shows the value of the impurity after the perturbation. So, a point $(54, 6.74)$ on the graph means that the impurity was 6.74 after the 54th perturbation. The perturbations are counted consecutively in spite of their type. So, the 54th perturbation can be a random jump, a stagnant perturbation or a hill climbing move. There are three curves in Fig. 3.13, corresponding to three different thresholds for it stagnant perturbations. When $stag = k$, the hyperplane is allowed at most k consecutive stagnant perturbations before trying a random jump. The curves are identical for the first 30 or so perturbations, after which they diverge. The first perturbation, for all three values of $stag$, was a hill climbing perturbation which brought the impurity down from 427.64 to 38.11. As showing such a large drop will reduce the clarity of the rest of the figure, the first perturbation is omitted for all curves in Fig. 3.13. On each curve, the locations of random jumps are marked.

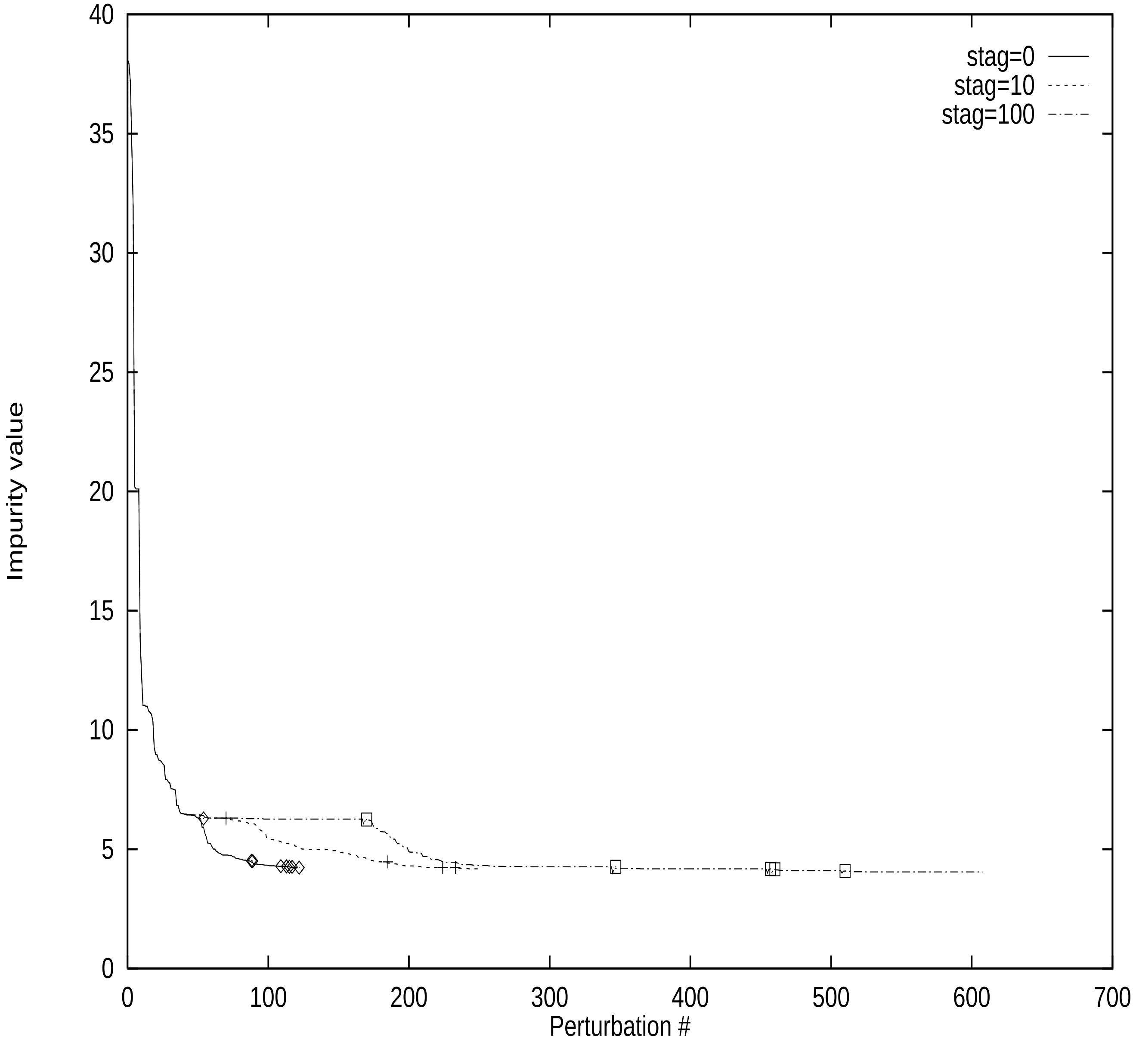

Figure 3.13: A typical search space of OC1 for the LS10 data. The first hill climbing perturbation for all the curves shown reduces the impurity from 427.64 to 38.11, and is not shown to improve clarity. The locations where random perturbations are made are marked on all curves.

Discussion

Several interesting observations can be made from Figure 3.13.

- The first few (30 to 40) hill climbing perturbations bring the impurity down from 427 to about 7. Even if we had aborted the search after these few perturbations, we would have obtained a good solution, spending much less time. We observed that the first few perturbations are the most effective for several domains. We are currently investigating ways to take advantage of this property. One possible application is to detect bad restarts early-on. For example, we can have a "checkpoint" after a few initial perturbations for each restart. From the second restart onwards, if the impurity at the checkpoint is much worse than that in the previous unaborted restart, we may be able to avoid the rest of the search in the current restart, without compromising on the solution.
- Random jumps are much rarer compared to hill climbing moves, in all the curves. For instance, when $stag = 100$, only 5 out of 600 perturbations were random jumps. This is in accordance with our earlier claim that randomization is used judiciously in OC1.
- As the number of stagnant perturbations allowed is increased, the search increases and increasingly large plateaus appear in the impurity curve. This is to be expected, as all the stagnant perturbations are tried before any random jump. However, this situation is not easy to rectify. If we swap the order of doing random and stagnant perturbations (i.e., try random jumps whenever the hill climbing is ineffective, before trying any stagnant perturbations) this can slow down the algorithm significantly, as random jumps are more computationally expensive than stagnant perturbations. Judicious choice of the number of stagnant perturbations is needed to ensure that search is not wasted. OC1's current default is to try ten stagnant perturbations before attempting a random jump.

3.5 Conclusions

This chapter described OC1, a new system for constructing oblique decision trees. We have shown experimentally that OC1 can produce good classifiers for a range of real-world and artificial domains. We have also shown how the use of randomization improves upon the original algorithm proposed by Breiman et al. (1984), without significantly increasing the computational cost of the algorithm. [35] We argued that the ability to produce oblique splits at a node broadens the capabilities of decision tree algorithms, especially as regards domains with numeric attributes.

Finding the best split at a node and finding the best decision tree are basically search problems. Axis-parallel tree methods use exhaustive search to find the best split

[35] OC1's deterministic coefficient perturbation algorithm was constructed independently of CART, and this is the reason for the slight differences in the algorithm. We realized the similarity with CART subsequently.

at a node, and greedy search to choose the best tree. For oblique trees, we retained the latter of these heuristics, and showed that the former (exhaustively searching for the best oblique split at a node) is impractical. We demonstrated that one particular local search algorithm, augmented with two specific types of randomization, is effective for finding a good split at a node. There are obviously other local search strategies and randomization steps one could use. For instance, one may use linear programming in place of our deterministic perturbation algorithm, and perhaps impose limits on the amount of deterministic search to be performed before the random perturbations are attempted. Random search methods like simulated annealing or threshold accepting [118] may be used in place of our randomization steps. However, we think that, with any search heuristics, an analysis of the search space should be done to identify the heuristics that are likely to be most useful. The experiment in Section 3.4.4 was intended to serve this purpose, but several alternate experiments can be designed. Interesting analyses of search methods, though not in the context of decision tree induction, can be found in [239, 166, 434].

Chapter 4

Limited lookahead search

The standard algorithm for constructing decision trees from a set of examples is greedy induction — a tree is induced top-down with locally optimal choices made at each node, without lookahead or backup. As the greedy approach can produce suboptimal trees [174], it is naturally of interest to explore ways to improve the greedy strategy. Fixed-depth lookahead search is a standard technique for improving greedy algorithms [426]. Though scattered uses of lookahead exist in the literature (Section 2.5.4), there have not been any systematic evaluations (analytical or empirical). The advantages, or lack thereof, of lookahead search have not been systematically quantified in the context of decision tree or rule induction. [36]

With the rapid increases in computing power in recent years, limited lookahead is now feasible for moderately large data sets. The question that therefore arises is, *what are the benefits (if any) that we might gain from employing this more costly approach?* In the current chapter, we attempt to answer this question empirically. We systematically compare, using a large number of real and artificial data sets, the quality of *axis-parallel* decision trees induced by the greedy approach to that of axis-parallel trees induced using one-level lookahead. The main observations from our experiments are:

- Limited lookahead search does *not* produce significantly better decision trees. On average, it produces trees with approximately the same classification accuracy and size as greedy induction, with slightly shorter longest paths.
- Limited lookahead search produces inferior (less accurate, larger and/or deeper) decision trees in a significant number of cases; i.e., decision tree induction exhibits the same *pathology* that has been observed in game trees [360].
- Tree post-processing techniques such as pruning are at least as beneficial as limited lookahead for a variety of real-world data sets.

[36] Quinlan and Jones recently experimentally analyzed "oversearching" in the context of rule induction [401]. Their conclusions are similar in spirit to ours, namely that oversearching does not help, and can hurt.

Our empirical evaluations are based on both synthetic and real-world data sets. To create the synthetic data, we defined two concept classes $\mathcal{C}$ and $\mathcal{C_S}$, and built *all possible* decision trees in these classes, many thousands of trees. We then created training and tests sets for all of these concepts. We also experimented with seven real-world domains from the UCI machine learning repository [346]. Our experiments only consider one level lookahead in this chapter. Although deeper lookahead might also be interesting, it is prohibitively expensive for the experimental design we employed.

Section 4.1 describes related work on using lookahead to improve greedy search. Section 4.2 describes our experimental method. Sections 4.3 and 4.4 present the results with synthetic and real world data respectively. Section 4.5 provides general conclusions.

4.1 Related work

Decision tree induction is fundamentally an optimization problem. Optimization algorithms typically consider a set of choices at each step towards a solution, improving things along the way. A greedy algorithm makes the choice that looks best at the moment, and does not return later to reconsider that choice. That is, it makes locally optimal choices in the hope that they will lead to a globally optimal solution. Greedy algorithms do not always yield optimal solutions, but for some problems they do. Examples of the latter variety include methods for constructing minimum spanning trees [178] and methods for producing optimal Huffman codes for data compression [283].

Greedy search is used as a heuristic for a number of well-known NP-hard problems. Examples are the 0/1 knapsack problem [419], multiprocessor scheduling [210] and the problem under consideration here, decision tree induction. For these problems, greedy polynomial algorithms obviously can not guarantee optimal solutions, assuming $P \neq NP$. It is natural, therefore, to look for techniques that systematically bridge the gap between the approximate solutions provided by greedy search and the optimal solutions. Fixed-depth lookahead search is one such technique. The idea is to repeatedly apply a greedy algorithm to selected subproblems in order to make less greedy (and hopefully more prudent) decisions. Sarkar *et al.*[426] show theoretically that lookahead search can guarantee ϵ-bounded solutions for the 0/1-knapsack problem and for a scheduling problem, where ϵ depends on the depth of lookahead.

Several variations of optimal decision tree induction are known to be NP-Complete (Section 2.6.1). Virtually all implemented decision tree systems use a greedy, top-down approach. There have been, however, some attempts to augment the simple greedy algorithm. (See Section 2.5.4 for a more comprehensive list of pointers.) Hartmann *et al.*[197] describe a tree induction algorithm based on an information theoretic criterion between branching levels in a tree. With the appropriate parameter settings, their Generalized Optimum Testing Algorithm (GOTA) can do fixed depth lookahead, varying depths of lookahead at different levels of the tree or even exhaustive search. However, Hartmann et al. do not demonstrate that lookahead yields any improvements over greedy search. The ideas in GOTA motivated

Norton's IDX system [365], which is a variant of ID3 [391] that performs lookahead. Norton conducted experiments on a voting records database (see Section 4.4) using ID3, IDX and GOTA, and found that lookahead reduced the average decision tree depth. With a few exceptions, though, the advantages of lookahead were very small in Norton's experiments. Moreover, since this study only considered a single data set, it is not clear how well these results generalize to other domains.

Interesting approaches to slightly different problems include Ragavan and Rendell's Lookahead Feature Construction (LFC) algorithm [403]. This method uses lookahead to construct composite Boolean features, and uses the constructed features to induce concise decision trees. This method is more efficient than methods like IDX because it caches the features found while looking ahead. Ragavan and Rendell describe experiments that show that LFC outperforms more straightforward approaches to feature construction and lookahead, on symbolic domains.

4.2 Experimental method

Our goal in this chapter is to evaluate systematically the gains (or losses) of limited lookahead search for decision tree induction. The greedy and one level lookahead algorithms we used in all our experiments are described below. S is a set of training examples whose attributes are assumed to be all numeric.

```
Algorithm Greedy(S)
1. If S contains examples from only one class, halt.
2. Consider all distinct tests T of the form $x < k$ on the features of S.
   The $k$'s are chosen to be the midpoints between adjacent feature values.
   Choose the best test $T^*$ according to a pre-defined goodness measure.
3. Split S into two subsets S1 and S2 using $T^*$.
4. Recursively run this procedure on S1 and S2.
```

The algorithm for one level lookahead, *Look*, uses the same set of candidate splits as *Greedy*. However, the goodness of a candidate split T is computed by examining all splits one level down from T.

```
Algorithm Look(S)
1. Execute step 1 of algorithm Greedy.
2. For each test T of the form $x < k$, do:
      (a) Split S into sets S1 and S2 using T.
      (b) Find the best split of S1 into sets S11 and S12, using
          steps 1-3 of algorithm Greedy.
      (c) Repeat (b) on S2, forming sets S21 and S22.
      (d) Compute the goodness of splitting S into S11, S12, S21, and S22,
          using the same goodness measure as in Greedy. This is T's goodness.
3. Execute steps 3,4 of algorithm Greedy.
```

We experimented with two pre-defined goodness measures, namely, the Gini index of diversity [44] and information gain [391]. [37] This gave us four algorithms, which we named *Greedy-Gini*, *Greedy-Info*, *Look-Gini*, and *Look-Info*. Note that *Greedy-Gini* is essentially identical to the CART algorithm [44] and *Greedy-Info* to the ID3 algorithm [391].

All our experiments measured tree quality in terms of four measures. *Accuracy* is the classification accuracy on either an independently generated test set (for the synthetic domains) or obtained by cross validation (for the real world domains). *Tree Size* is the number of leaf nodes in a tree. *Maximum Depth* is the length of the longest path, from the root to a leaf node. *Number of candidate splits* is the total number of splits evaluated by the tree induction algorithm while building a tree. The final measure quantifies the amount of search performed during tree induction.

4.2.1 Synthetic data

For a systematic evaluation of the benefits of lookahead, we compared the trees induced with limited lookahead to those induced with greedy search *over entire classes of decision trees.* (This style of empirical investigation has been made possible by the existence of extremely fast, inexpensive workstations. Murphy and Pazzani [347] used this style of experimentation to evaluate the Occam's Razor principle by constructing all decision trees consistent with a fixed concept.) In the current study, we have defined two classes of decision trees $\mathcal{C}$ and $\mathcal{C}_\mathcal{S}$, which are small enough to be amenable to systematic experimentation on the entire class, and general enough to be interesting. We built decision trees for all possible concepts in $\mathcal{C}$ and $\mathcal{C}_\mathcal{S}$ with and without lookahead, and compared the results. The precise definitions of $\mathcal{C}$ and $\mathcal{C}_\mathcal{S}$ and the experimental results are given in Section 4.3.

For our experiments with synthetic data, we first generated an unlabeled training set *TRAIN* and an unlabeled test set *TEST*. *TRAIN* has 500 examples and *TEST* has 5000 examples, with two real-valued attributes for each example, and all attribute values generated uniformly at random in the interval $(0, 10)$. The same unlabeled training and test sets are used in all the experiments. As *TRAIN* is noise-free, no pruning was used. We performed the following experiment on classes $\mathcal{C}$ and $\mathcal{C}_\mathcal{S}$, using both Gini index and information gain.

```
For each of the decision trees D in the class:
    – label TRAIN and TEST according to D;
    – greedily induce a decision tree $D_1$ on TRAIN;
    – induce a decision tree $D_2$ on TRAIN using lookahead;
    – and record the accuracy on TEST, tree size, max. depth and
      the number of splits considered, for $D_1$ and $D_2$.
```

[37] We chose Gini index and information gain because they have been widely used for real world applications. Experiments with other goodness measures may be interesting, but we suspect the results would be similar.

4.2.2 Real-world data

In addition to synthetic data, we have also experimented with noisy, real world data sets for which the underlying concepts are unknown. We evaluated the effects of one-level lookahead on seven data sets taken from the University of California at Irvine repository of machine learning databases [346].

If a greedy method can induce a highly accurate, concise classifier for a domain (the well-known Iris data is one such example), it is unlikely that we can observe significant benefits of lookahead on that domain. It has been observed that most of the data sets in the UCI repository can be described by very simple classification rules [209]. So, we needed to be careful in our choice of real world domains. We used a survey of results on several UCI data sets provided by Holte [209] to choose six "difficult" domains – domains for which the best known accuracy is at most 90%. Though the low accuracies may be due to factors other than the inadequacy of greedy induction, such as an overly small or noisy training set, there is no straightforward way of knowing this *a priori*. In addition to these six "difficult" domains, we also experimented with two variants of the congressional voting records data used by Norton [365] for his lookahead experiments. Brief descriptions of all our real world domains, and the results of the experiments are given in Section 4.4.

We augment all our algorithms (*Greedy-Gini, Look-Gini, Greedy-Info* and *Look-Info*) with pruning for the experiments with real world data – in effect, we experimented with eight different tree induction algorithms. We used Breiman *et al.*'s cost complexity pruning ([44], Chapter 3) with the one standard error rule, reserving 10% of the training data as the pruning set. We used 5-fold cross validation to estimate accuracy, as can be seen in the following summary of our experimental method for the real world domains. For each of our data sets DATA, we repeated this procedure with *Greedy-Gini*, *Greedy-Info*, *Look-Gini* and *Look-Info*, with and without pruning.

```
repeat ten times with different random seeds:
   – divide DATA randomly into 5 equal disjoint partitions;
   – foreach partition P
         – induce a decision tree on (DATA - P);
         – record tree size, max. depth and no. of misclassified examples in P;
   – report the accuracy on DATA and the average size and depth;
```

4.3 Experiments with synthetic data

4.3.1 $\mathcal{C}$: A class of simple data sets

Our first set of experiments are designed to measure how close to optimal are the trees produced by greedy induction on a fixed concept class. More precisely, we consider a class of concepts $\mathcal{C}$, all members of which are simple enough for one-level lookahead to be equivalent to exhaustive search, and systematically evaluate the effectiveness of greedy induction over this entire class. $\mathcal{C}$ is a class of binary decision trees defined as follows:

Figure 4.1: Class $\mathcal{C}$ consists of all balanced decision trees on a 10 X 10 grid such that each tree has three test (internal) nodes and no test node is trivial.

- Each member of $\mathcal{C}$ has 3 test nodes and 4 leaves, and is balanced. The tests at the three internal nodes (root, left, and right) are all non-trivial, in the sense that they split heterogeneous sets of examples.
- $\mathcal{C}$ contains only two attributes, x_1 and x_2 (to enable graphical display and comparison of trees).
- The tests are restricted to be of the form $x_i < k$ where k is an integer in $(0, 10)$.
- $\mathcal{C}$ contains exactly two classes, 1 and 2.

See Figure 4.1 for a graphical definition of the class $\mathcal{C}$, which has a total of 5844 distinct trees. (Trees that are equivalent except for having their class labels swapped are not considered distinct.) $\mathcal{C}$ is defined to make systematic exhaustive search on all possible concepts computationally feasible. One reason for defining class $\mathcal{C}$ in this form is that one level of lookahead from the root will always find the optimal decision tree in terms of the size and depth. Trees in this class may realistically occur in many situations as subtrees of a larger tree, and it is reasonable to ask if we should constantly check one level ahead to "finish off" a subtree. However, even one level of lookahead is very costly, so we wish to quantify its possible advantages. (The work done at a single node by the standard greedy algorithm is at most $O(dn \log n)$ for d attributes and n examples. One level of lookahead requires at most $O(d^2n^2)$ work.)

We used the experimental method defined in Section 4.2.1. Each decision tree in $\mathcal{C}$ was used to label *TRAIN* and *TEST*. Two sets of 5844 trees each were induced on *TRAIN* using *Greedy-Info* and *Greedy-Gini*. A third set of 5844 trees was induced on $\mathcal{C}$ using one level lookahead. Note that, as one level lookahead is the same as exhaustive search on $\mathcal{C}$, *Look-Info* and *Look-Gini* produce identical trees for this class. Figure 4.2 summarizes the

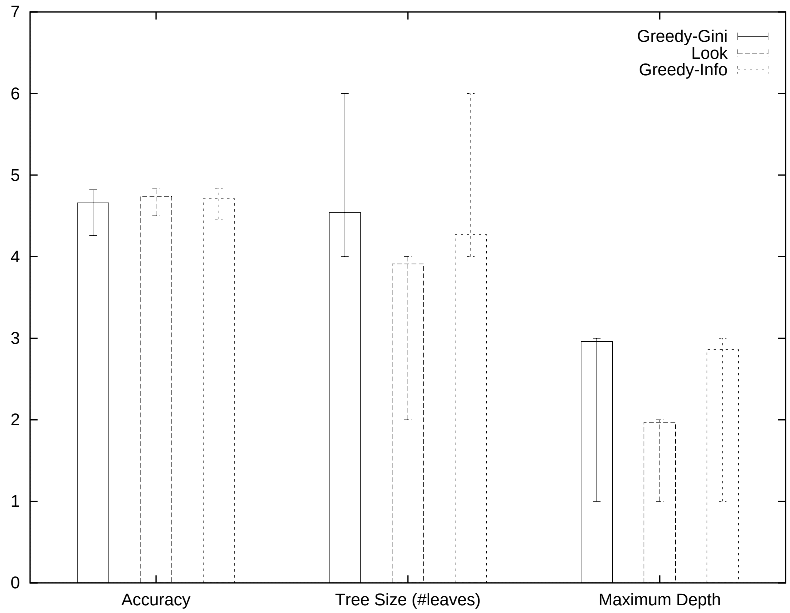

Figure 4.2: Summary of experiments with class $\mathcal{C}$. The mean and one quartile ranges for accuracy, tree size and maximum depth are shown for *Greedy-Gini*, one-level lookahead and *Greedy-Info*. The accuracies shown are the amounts above a baseline value of 95%; e.g., the accuracy of lookahead was 99.74%.

differences between the quality of decision trees induced by *Greedy-Gini*, *Greedy-Info*, and exhaustive search, over the entire class $\mathcal{C}$. The figure shows the mean and one quartile ranges of the accuracy, tree size and maximum depth. (One quartile range is the interval that includes 25% of the samples above and below the mean.)

As the figure shows, the differences between *Greedy-Gini*, *Greedy-Info*, and *Look* are quite small, in spite of the fact that greedy induction uses only about 0.004 times as much search as exhaustive search. *Greedy-Gini* evaluates 1798 candidate splits on average per tree, and *Greedy-Info* evaluates 1718 splits. In contrast, *Look* evaluates 419,301 splits for each tree, on average. The differences in average accuracy between the greedy algorithms and *Look* are negligible. The difference in average tree size between *Greedy-Info* and exhaustive search is 0.36 nodes, less than one standard deviation of 0.4 nodes . The difference of 0.63 between the average tree size of *Greedy-Gini* and *Look* is slightly more pronounced. The only aspect in which greedily induced trees are significantly worse than the optimal trees is maximum depth. Exhaustive search produces trees that are on average one level shallower than the greedy algorithms.

Figures 4.3 and 4.4 show the effects of one-level lookahead (equivalently, exhaustive search) for class $\mathcal{C}$ in more detail. The horizontal axis plots the improvement due to lookahead. Thus the line for accuracy shows the *increase* in accuracy due to lookahead, measured by percentage change. The lines for tree size and depth show the *decrease* in these measures when lookahead is used. The vertical axis plots the number of trees in which lookahead causes a particular improvement. For instance, a point (x, y) on the line

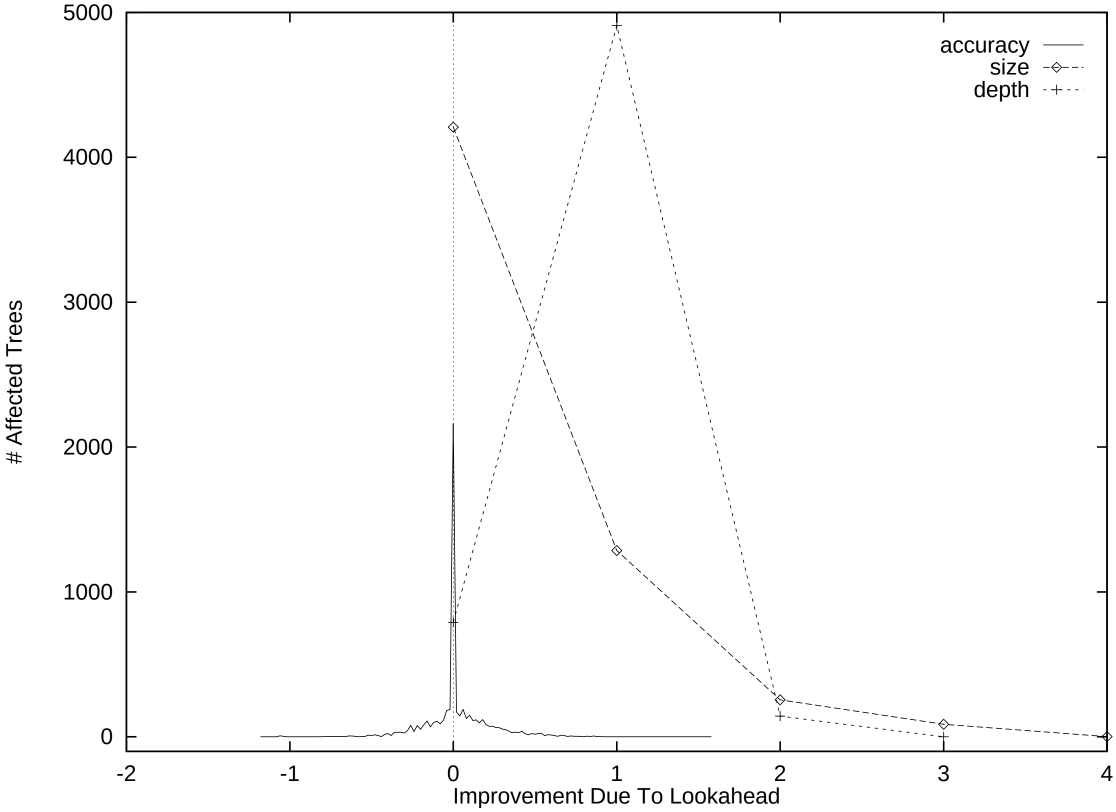

Figure 4.3: Effect of one level lookahead in trees produced with information gain, for class $\mathcal{C}$. Improvements in accuracy, size and maximum depth are shown, along with the number of trees in which these improvements occur. Negative values on the X-axis mean that lookahead produced inferior trees.

"accuracy" in the graph means that, for y trees out of the total of 5844, accuracy improved by x when lookahead was used. For points on $X = 0$, lookahead had no effect, and for points to the right of $X = 0$, lookahead was beneficial. For points to the left of the line $X = 0$, greedy induction was *better* than lookahead; i.e., the tree induction task exhibited pathology.

We discuss only the measurements for information gain (Fig. 4.3) as the graph for Gini index (Fig. 4.4) looks almost identical. Fig. 4.3 offers several interesting insights. First, each of the three lines has a single, prominent peak. The peaks for the accuracy and tree size lines are at $X = 0$, and the peak for depth is at $X = 1$. The peaks at $X = 0$ show that for a large number of trees, lookahead did not make *any* difference in terms of accuracy and number of leaves. The depth peak at $X = 1$ shows that the maximum depth of *most* of the greedily induced trees is exactly one more than optimal. To understand why the greedy approach builds deeper trees, we looked at several of these trees individually, and found that many had a structure similar to the one shown in Figure 4.5. Although

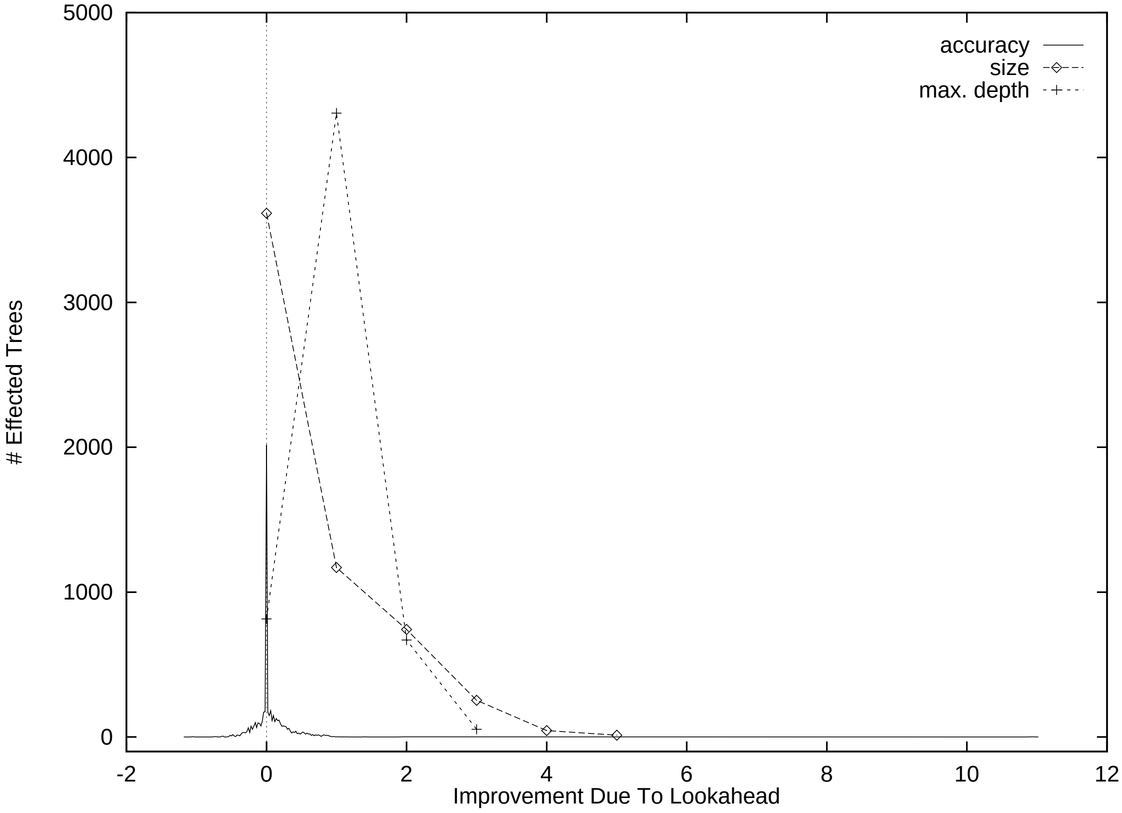

Figure 4.4: Effect of one level lookahead in trees produced with gini index, for class $\mathcal{C}$. Improvements in accuracy, size and maximum depth are shown, along with the number of trees in which these improvements occur. Negative values on the X-axis mean that lookahead produced inferior trees.

Figure 4.5: A tree with maximum depth of 3 produced by *Greedy-Info*. It is easy to see that this tree can be balanced. By making **Root** the left child of **R**, and **R** the root, the depth can be reduced to 2, without changing the partitioning induced (and the accuracy)

this tree partitions the data optimally, it is deeper than necessary. Section 4.3.4 describes a simple post-processing step to rebalance a greedily induced tree, in order to reduce its worst-case classification cost.

Second, it is interesting to note that lookahead actually *hurts* accuracy in almost as many trees as those in which it enhances accuracy. This property, where lookahead search finds inferior solutions, is known as *pathology* in the context of game trees [356, 360]. We discuss pathology for decision trees further in Section 4.3.2, where this trend is exhibited more prominently. Pathology cannot occur for tree size or depth for class $\mathcal{C}$, because one-level lookahead is equivalent to exhaustive search. However, our next class $\mathcal{C}_{\mathcal{S}}$ includes deeper trees, and limited lookahead can and does produce trees that are worse in terms of size and depth.

Third, we can see from Figure 4.3 that there are some greedily induced trees that have as many as 4 leaves more than those induced with lookahead. We looked at all such large trees, and found that they all had several "minimally useful" splits. For example, consider the tree shown in Figure 4.6. We can usually avoid such splits with a simple stop-splitting rule. Pruning will also tend to remove them, but pruning is not fully justified on our synthetic data sets as they are noise-free.

4.3.2 $\mathcal{C}_{\mathcal{S}}$: A more difficult class

In section 4.3.1, we experimented on a class $\mathcal{C}$ of concepts for which one-level lookahead is equivalent to exhaustive search, and showed that the benefits of lookahead are negligible. This section extends $\mathcal{C}$ to a class $\mathcal{C}_{\mathcal{S}}$, where one level of lookahead does not perform exhaustive search. $\mathcal{C}_{\mathcal{S}}$ is a more realistic concept class than $\mathcal{C}$, because in practice we can do only

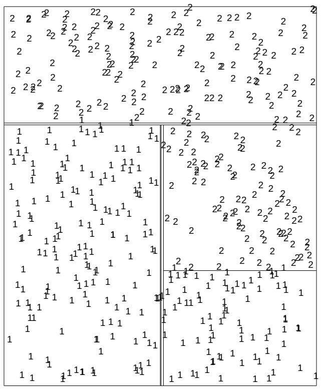

Figure 4.6: A tree with 8 nodes and maximum depth of 4 produced by *Greedy-Gini*. Each thick line consists of three or four splits, very close to each other. A simple stop-splitting rule can avoid most splits in this tree, without significantly changing the accuracy.

limited lookahead. $\mathcal{C}_{\mathcal{S}}$ is an extension of $\mathcal{C}$ obtained as follows:

```
$\mathcal{C}_{\mathcal{S}} = \emptyset$
For each tree $T$ in $\mathcal{C}$, do:
    1. Remove $T$ from $\mathcal{C}$.
    2. Randomly choose one of the four leaf nodes of $T$. Call this $L$.
    3. Split $L$ with a randomly chosen, non-trivial split $\mathcal{S}$ of the form
       $x_i < k$, where $k$ is an integer in the range $(0, 10)$. If there is no
       valid split, go to step 2 and choose a different leaf.
    4. Assign one side of $\mathcal{S}$ to class 1 and the other side to class 2.
    5. Add $T$ to $\mathcal{C}_{\mathcal{S}}$.
```

Each decision tree in $\mathcal{C}_{\mathcal{S}}$ is a binary tree with four decision nodes and has a maximum depth of 3. Note that while $\mathcal{C}_{\mathcal{S}}$ has 5844 trees, the same as $\mathcal{C}$, another run of the above procedure would create a different definition of $\mathcal{C}_{\mathcal{S}}$ because of the randomized steps. We considered using exhaustive enumeration in place of these random choices, but that would produce a class that is vastly larger, too large for systematic experimentation. The experimental method used for $\mathcal{C}_{\mathcal{S}}$ is identical to that used for $\mathcal{C}$, and is described in Section 4.2.1. One important difference is that, since one-level lookahead is not equivalent to exhaustive search on $\mathcal{C}_{\mathcal{S}}$, *Look-Gini* and *Look-Info* do not produce identical trees for this class.

The experimental results with class $\mathcal{C}_{\mathcal{S}}$ substantiate the conclusions drawn from experiments with class $\mathcal{C}$, in section 4.3.1. Figure 4.7 summarizes the differences in accuracy, tree size and maximum depth between *Greedy-Gini*, *Look-Gini*, *Greedy-Info*, and *Look-Info*

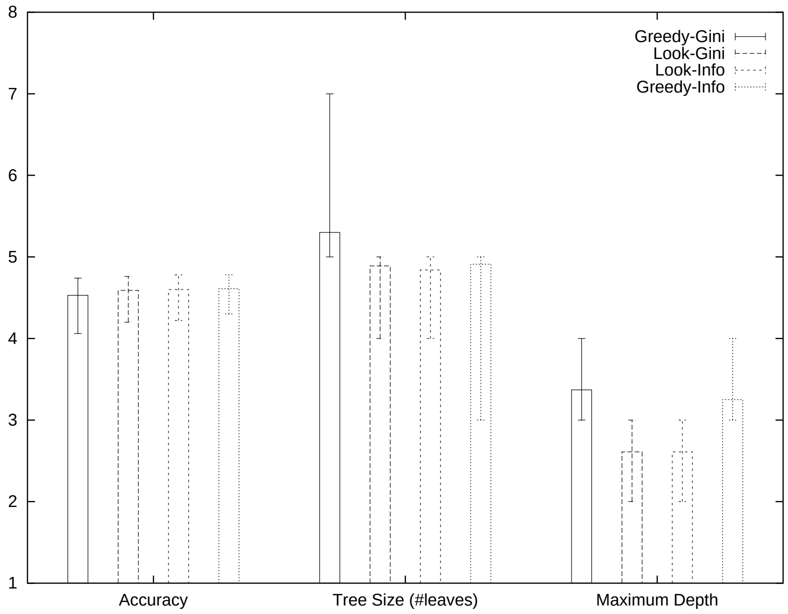

Figure 4.7: Summary of experiment with class $\mathcal{C}_{\mathcal{S}}$. The mean and one quartile ranges for accuracy, tree size and maximum depth are shown for *Greedy-Gini*, *Look-Gini*, *Greedy-Info*, and *Look-Info*. The accuracies shown are the amounts above a baseline value of 95%.

on class $\mathcal{C}_{\mathcal{S}}$. It can be seen from this figure that there is no significant improvement in accuracy due to one level lookahead. The differences between accuracy with and without lookahead are actually smaller here than they were for class $\mathcal{C}$, despite the fact that the relative cost of lookahead search was higher for this class. On average, *Greedy-Gini* considered 1952 candidate tests per tree and *Greedy-Info* considered 1847 splits. *Look-Gini* and *Look-Info*, in contrast, considered 745,689 and 747,037 tests respectively. The differences in tree size are insignificant for both goodness measures. The only quantity for which one-level lookahead caused noticeable improvement for class $\mathcal{C}_{\mathcal{S}}$ was maximum depth, where trees induced with lookahead were on average 0.6 levels shallower than greedily induced trees.

Figure 4.8 shows the effect of one-level lookahead for class $\mathcal{C}_{\mathcal{S}}$ in more detail for *Greedy-Info*. The corresponding picture for *Greedy-Gini* looks nearly identical and is shown in Fig. 4.9. As in Figure 4.3, points to the left of $X = 0$ are concepts in $\mathcal{C}_{\mathcal{S}}$ for which limited lookahead performs *worse* than greedy search; i.e., the problem exhibits pathology. Points on $X = 0$ are concepts for which lookahead makes no difference at all. Points to the right of $X = 0$ are concepts for which limited lookahead search produces better trees. The peaks for accuracy, tree size and maximum depth in Fig. 4.8 coincide with those in Figure 4.3. It is interesting to note that there were *more* trees for which lookahead hurt accuracy than there were those for which it benefited. lookahead produced worse trees in terms of tree size also. It produced trees that had as many as six more leaves (five more decision nodes) than greedy induction. Section 4.3.3 shows an example of pathology.

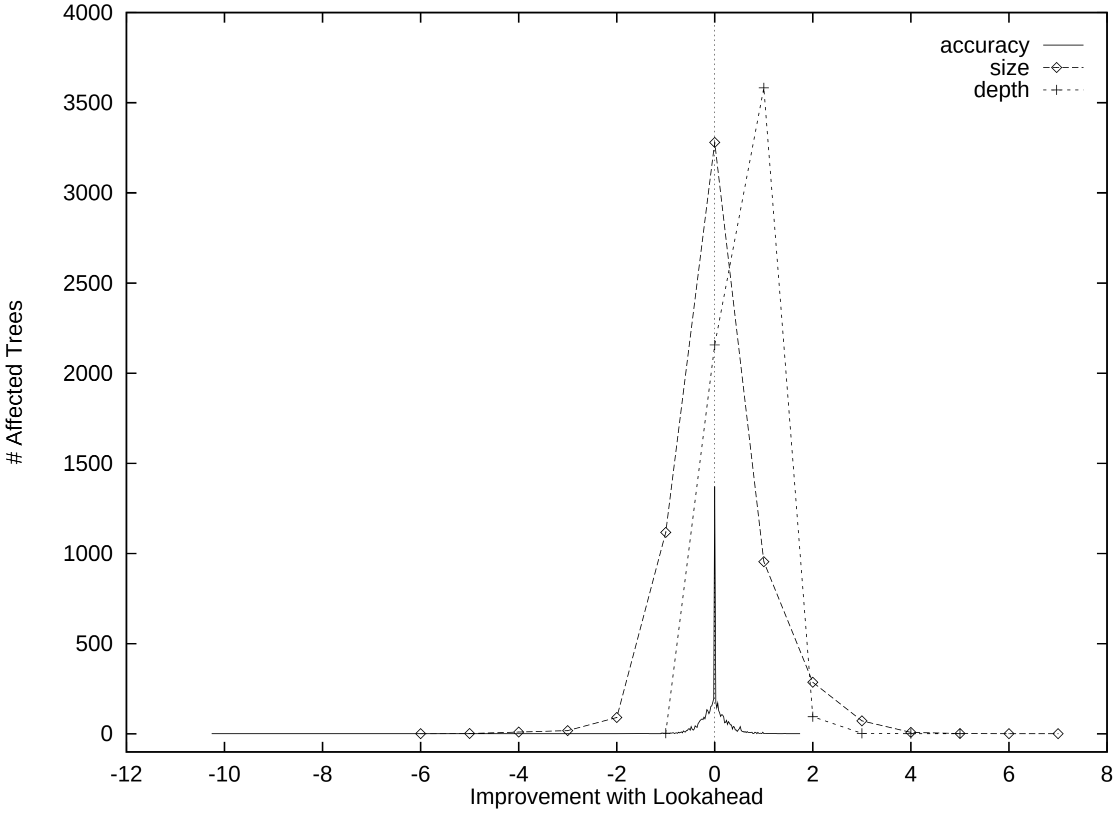

Figure 4.8: Effect of one level lookahead for trees in class $\mathcal{C}_{\mathcal{S}}$. Improvements in accuracy, size and maximum depth of trees built using *Look-Info* versus *Greedy-Info* are shown. Negative values on the x-axis show instances where lookahead produced inferior trees.

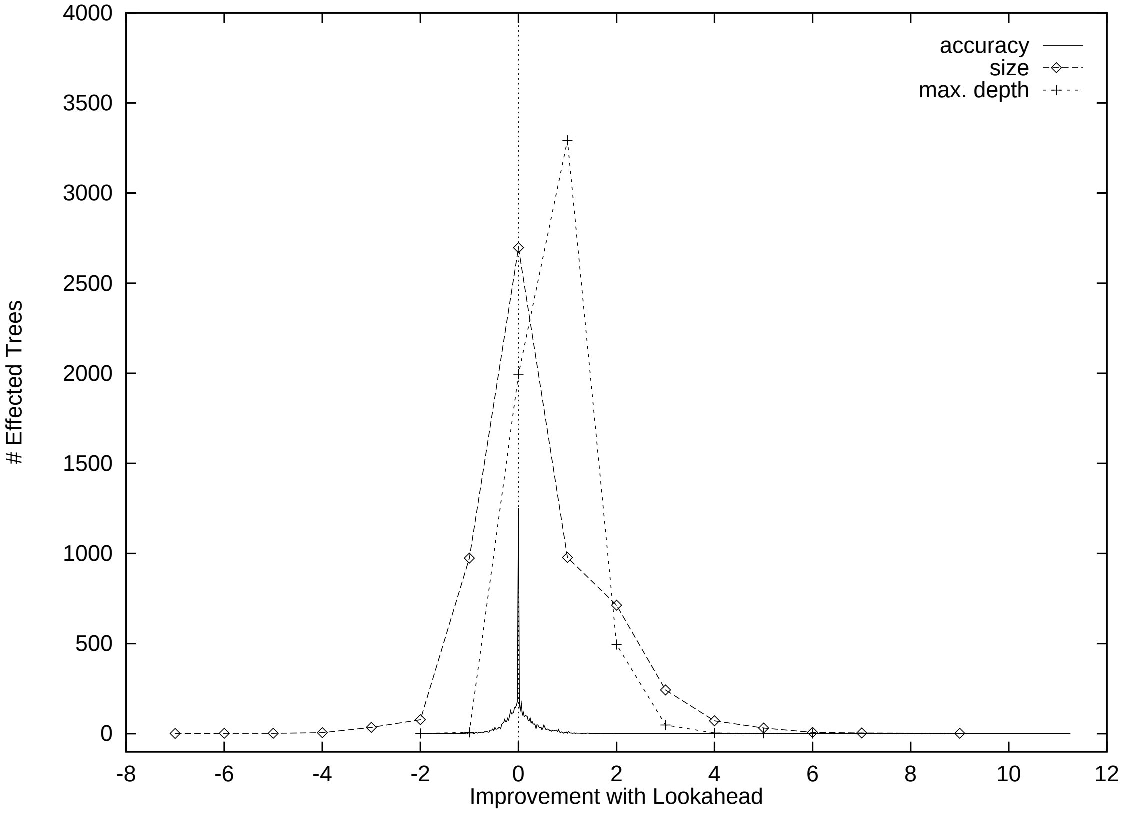

Figure 4.9: Effect of one level lookahead for trees in class $\mathcal{C}_{\mathcal{S}}$. Improvements in accuracy, size and maximum depth of trees built using *Look-Gini* versus *Greedy-Gini* are shown. Negative values on the x-axis show instances where lookahead produced inferior trees.

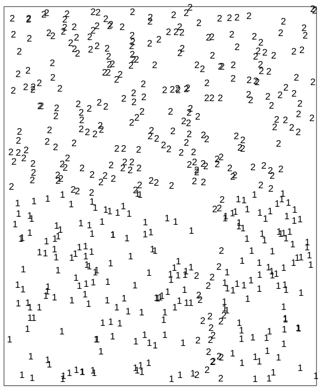

Figure 4.10: A data set in class $\mathcal{C}_{\mathcal{S}}$ on which information gain exhibits pathology.

4.3.3 An example of pathology

Fig. 4.10 shows a labeling of *TRAIN*, the training set we used for all our experiments with synthetic data. This labeling was obtained by a tree in class $\mathcal{C}_{\mathcal{S}}$. Information gain exhibits pathology on this data – *Greedy-Info* produces a better tree than that produced with *Look-Info*, in terms of *all* our three measures, accuracy, size and maximum depth.

We show below the tree induced by *Greedy-Info* and *Look-Info* on the data in Fig. 4.10, both in terms of the equations for the splits in the trees and the partitioning induced on the training data. Each equation of a split is of the form **label :equation**. **label** is a character string comprising of characters **l** and **r**, denoting left and right. All examples answering *yes* to the question at a decision node take the left branch, and all others take the right branch. The split at the root of the decision tree has an empty label. Every other split has a label specifying where in the tree the split occurs. For example, a split at the "**l**"eft child of the "**r**"ight child of the root has the label **rl**. Leaf nodes have class labels instead of equations. Figure 4.11 graphically shows the partitions induced by the *Greedy-Info* and *Look-Info* trees, making it obvious that greedy induction induces a much better tree for this domain.

Greedy-Info Tree

```
: $x_2 < 4.99$?
  l :$x_1 < 6.04$?
    ll :class 1
    lr :$x_1 < 7.01$?
      lrl :class 2
      lrr :class 1
  r :class 2
```

Look-Info Tree

```
: $x_2 < 3.05$?
  l :$x_1 < 6.04$?
    ll :class 1
    lr :$x_1 < 7.01$?
      lrl :class 2
      lrr :class 1
  r :$x_2 < 4.59$?
    rl :$x_1 < 6.32$?
      rll :class 1
      rlr :$x_1 < 6.98$?
        rlrl :class 2
        rlrr :class 1
    rr :$x_2 < 4.82$?
      rrl :$x_2 < 4.81$?
        rrll :class 1
        rrlr :class 2
      rrr :$x_2 < 4.99$?
        rrll :class 1
        rrlr :class 2
```

4.3.4 “Rebalancing” greedy trees

Some examples of methods to “refine” greedily induced trees are given in Section 2.5.4. We outline here a simple refining method, which we call tree rebalancing. Given a decision tree D for a training set *TRAIN*, we want to produce a tree D_B that induces the same partitioning as D on *TRAIN*, but has lower average depth than D (i.e., D_B is more “balanced” than D). The average depth of a tree is the path length between the root and a leaf node, averaged over all the leaf nodes. (This measure is also used in the experiments in Chapter 5.)

Although little if any work has been done on balancing decision trees, a great deal of research has considered balanced search trees (e.g., see [16, 79, 461, 93, 357]). Roughly speaking, this literature deals with techniques to restructure search trees when elements are inserted or deleted, in order to restrict the depth of these trees to a logarithmic function of the number of search keys. Examples of balanced search trees include AVL trees, B-trees, and red-black trees. An axis-parallel decision tree in a continuous space can be interpreted as a multi-dimensional binary search tree, where each internal node stores an axis number and a search key along that axis. Such an interpretation makes it possible to use search tree balancing techniques on decision trees.

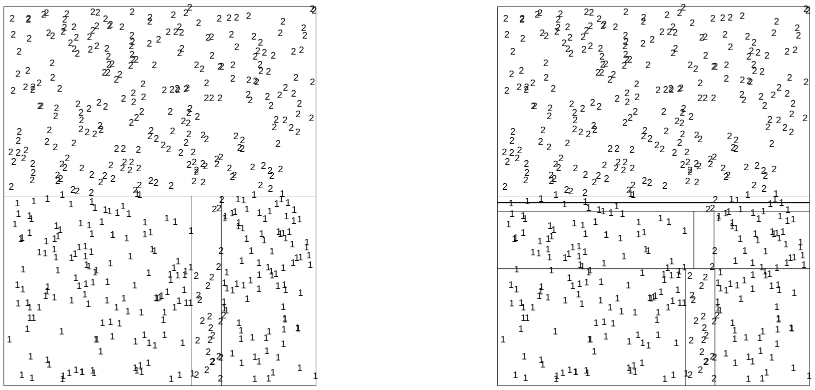

Figure 4.11: Trees induced by information gain without and with lookahead for the data in the previous figure. The tree induced by *Greedy-Info* has size 4, maximum depth 3 and accuracy 99.74% on an independent test set. On the other hand, *Look-Info* induces a tree with size 10, maximum depth 4 and accuracy of 99.10%. *Greedy-Info* considers a total of 1545 splits while inducing its tree, whereas *Look-Info* considers a total of 1,455,901 splits.

The main primitives used for rebalancing a tree in balanced search tree methods are *rotations*. Rotations are operations in which the parent-child links of some nodes in the tree are rearranged locally, while guaranteeing that the functionality of the whole tree remains invariant. We have adapted two simple tree rotation operators, left-rotate and right-rotate, to decision trees. These operators, illustrated in Figure 4.12, take constant time for each rotation. The following procedure can be used for balancing a decision tree T. Note that as this procedure itself is a greedy heuristic, it may not result in optimally balanced trees.

```
Algorithm Balance(T)
          if $T$ = then halt;
          if T and right-subtree(T) test the same attribute at the root
             $T_1$ = left-rotate(T);
             if avg. depth($T_1$) < avg. depth($T$) then T = $T_1$;
                skip the next If statement;
          if T and left-subtree(T) test the same attribute at the root
             $T_1$ = right-rotate(T);
             if avg. depth(T1) < avg. depth(T) then T = $T_1$;
          Balance (left-subtree(T));
          Balance (right-subtree(T));
```

Figures 4.13 and 4.14 show the effect of post-processing on the greedily induced trees in class $\mathcal{C}$ and class $\mathcal{C}_{\mathcal{S}}$ respectively. Results for *Greedy-Gini* are shown, and results for

Figure 4.12: Left and right rotations of a binary decision tree. Rotation operators can help reduce the average depth, and thus the expected cost of classification, of a decision tree without changing its accuracy. The leaf nodes L1, L2 etc. in this figure can be replaced with arbitrary subtrees.

Greedy-Info look very similar. Each figure consists of five stacked bars, the first three corresponding to maximum depth and the last two to tree size. Each bar shows the distribution of 5844 trees into bins of equal size or depth. The three maximum depth bars correspond to (i) no post-processing, (ii) tree balancing as a post-processing step and (iii) tree balancing and stop-splitting. The two tree size bars show the distribution of the trees with no postprocessing and with stop-splitting. (Note that tree size is not altered by balancing.)

Figures 4.13 and 4.14 show that tree balancing reduces the maximum depth of several trees in classes $\mathcal{C}$ and $\mathcal{C}_{\mathcal{S}}$. In addition, the stop-splitting rule reduces tree size and depth substantially. Note that stop-splitting, unlike balancing, decreases classification accuracy because it allows for heterogeneous partitions. However, this reduction was small for both classes. For class $\mathcal{C}$, the average accuracy with stop-splitting was 99.3, as opposed to 99.6 when no post-processing was used. For class $\mathcal{C}_{\mathcal{S}}$, average accuracy dropped from 99.5 to 99.0 when stop-splitting was used.

4.4 Experiments with real-world data

The experiments with synthetic data, presented in Section 4.3, do not suggest how lookahead search might perform on real world, noisy, higher dimensional domains. This section describes experiments we did with seven data sets available in the UCI Machine Learning repository [346]. Brief descriptions of each of the real data sets we used are given below. A brief rationale for choosing these particular data sets, and our experimental method for real world domains are given in Section 4.2.2.

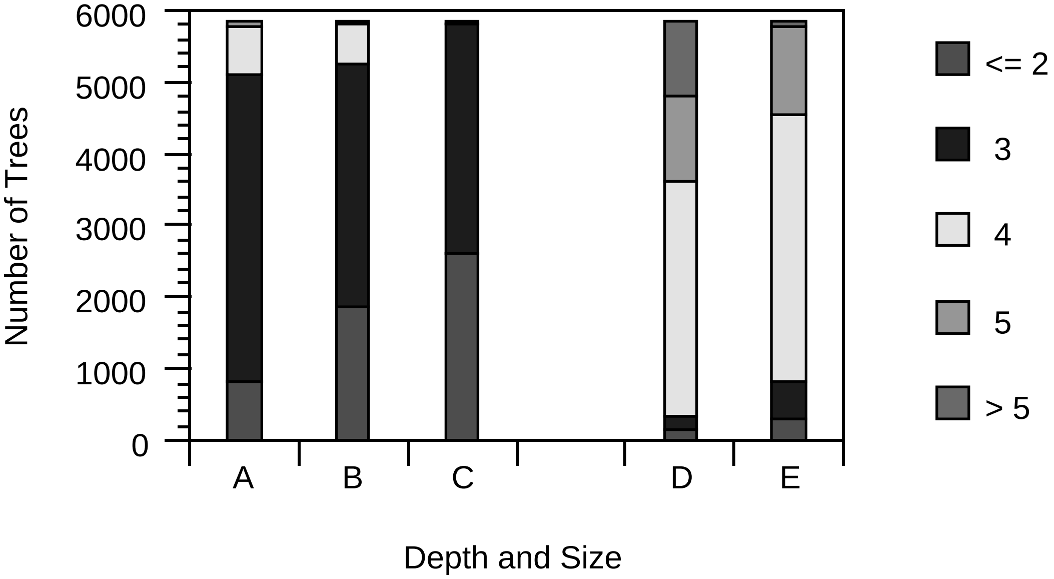

Figure 4.13: Improvement due to post-processing in greedily induced trees for class $\mathcal{C}$. First three bars show the distribution of class $\mathcal{C}$ trees according to maximum depth (i) with no postprocessing, (ii) with tree balancing and (iii) with stop-splitting and balancing. The last two bars show distribution of tree size (i) without any postprocessing and (ii) with stop-splitting. All the results are for *Greedy-Gini*.

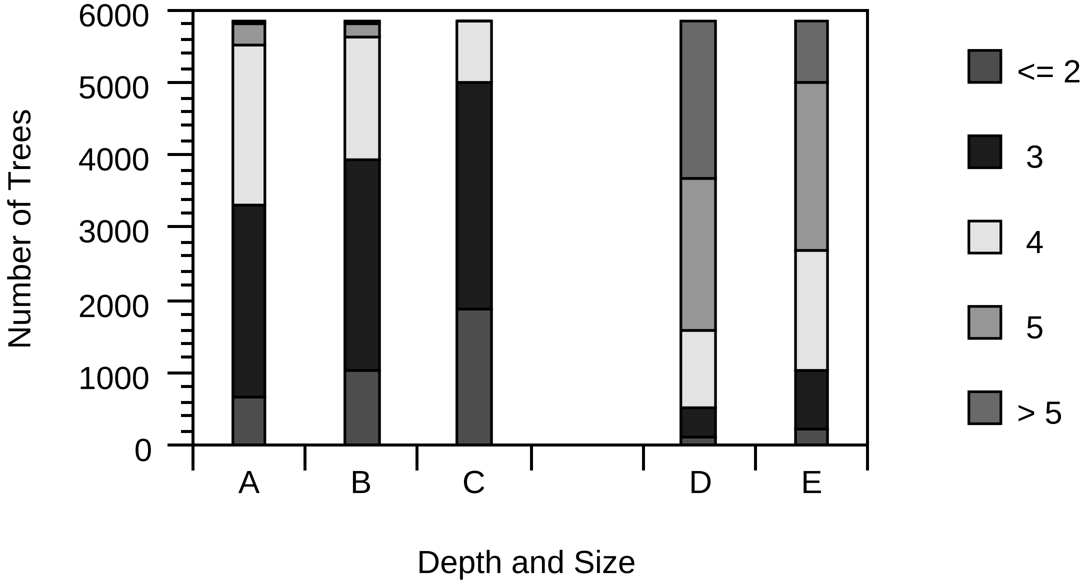

Figure 4.14: Improvement due to post-processing in greedily induced trees for class $\mathcal{C}_{\mathcal{S}}$. First three bars show the distribution of trees according to maximum depth (i) with no postprocessing, (ii) with tree balancing and (iii) with stop-splitting and balancing. The last two bars show distribution of trees according to size (i) without any postprocessing and (ii) with stop-splitting. Results for *Greedy-Gini* are shown.

BC Breast cancer recurrence data [38]. Contains 286 instances, each described by 9 attributes and one class label. The task is to predict if a breast cancer event is to recur.

CL Cleveland Clinic Foundation's heart disease diagnosis data [89]. Contains 303 instances, each described by 14 attributes including the class label. We use the "processed" data in the UCI repository, where there are only two classes, namely the presence and absence of heart disease.

GL Glass identification data. Contains 214 instances described using ten continuous valued attributes and a class label. The first attribute "Id number" is not used in our experiments.

HE Hepatitis domain. Contains 155 instances, described using 20 attributes including the class label. The task is to classify patients that die from hepatitis from those that do not.

LA Final settlements in labor negotiations in Canadian industry. 57 instances each described using 16 features.

LY Lymphography domain [304]. Contains 148 instances, each described using 19 attributes, including the class attribute.

VO 1984 United States congressional voting records database. This data is used by Norton [365] in his experiments. The data contains 435 instances, each described by 16 nominal attributes and one class label. The task is to classify democrats from republicans on the basis of their voting records.

V1 This data, used in [59, 209], is identical to the VO data, except that the "best" attribute *physician-fee freeze* is removed.

All the experimental results reported in this section are obtained with information gain. The experiments with Gini index did not offer any more insights, and are omitted for brevity. Figures 4.15, 4.16 and 4.17, one for each of the three measures accuracy, tree size and maximum depth, summarize the results of our experiments on the real-world data sets. In each figure we plot the values of a measure obtained using four induction methods: (i) *Greedy-Info*, (ii) *Look-Info*, (iii) *Greedy-Info* with pruning and (iv) *Look-Info* with pruning. Each value is the average of ten 5-fold cross validations, as explained in Section 4.2.2. There are eight lines in each figure, corresponding to eight data sets.

Consider the accuracy plot in Fig. 4.15. The first observation is that the accuracies do not vary much between various induction methods. On closer observation, accuracy *drops* for six out of the eight data sets (all except V1 and GL) when lookahead is used, though not significantly for all of them. In addition, *Greedy-Info* with pruning produces more accurate trees than *Look-Info* for the V1 data (and four others). Pruning, which is a much less expensive technique than lookahead, seems to be more beneficial in terms of accuracy. It

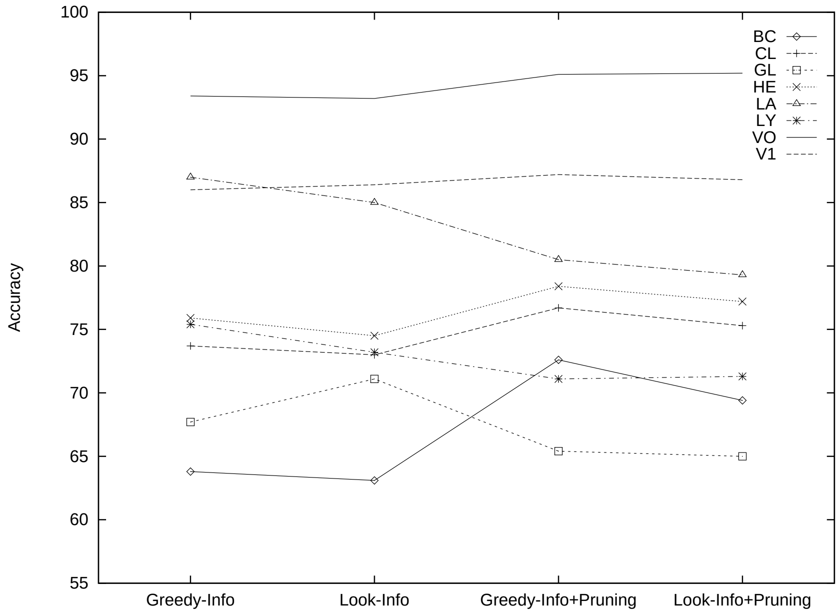

Figure 4.15: Effect of one level lookahead on accuracy for eight real-world data sets. The accuracies with and without lookahead, and with and without pruning are shown for information gain.

is of course unfair to expect lookahead to compensate for pruning on noisy domains. So, we evaluated lookahead plus pruning versus just pruning (columns four and three), to see if pruning complements lookahead induction better than it complements greedy induction. This doesn't seem to be the case — *Look-Info* doesn't do very well with pruning either. The differences in accuracy, between *Greedy-Info* with pruning and *Look-Info* with pruning are insignificant for all domains except one, the BC data. For this data, *Greedy-Info* with pruning is better than *Look-Info* with pruning. Our overall impression from the accuracy plot in Fig. 4.15 is that lookahead doesn't affect the accuracy significantly. Better benefits than lookahead can be obtained with pruning, which is a much less expensive alternative.

Now consider the tree size plot in Fig. 4.16. Lookahead does reduce the tree size, significantly for two domains (BC and GL) and slightly for five others. These benefits, however, are over-shadowed by the benefits of pruning. For all domains except the LA data, pruning helps produce much smaller trees than both the greedy and lookahead methods. As lookahead and pruning independently help reduce the tree size, it is natural to ask if using them together would accumulate the benefits. From the results in the third and fourth columns, it is clear that this is not the case. The differences between *Greedy-Info* with pruning and *Look-Info* with pruning are negligible for most domains. In the only case where this difference is significant (for the BC data), *Look-Info* with pruning produces worse trees than *Greedy-Info* with pruning.

The plot for maximum depth in Fig. 4.17 looks similar to the size plot in Fig: 4.16. *Look-Info* produces shallower trees for all eight domains than *Greedy-Info*, but the benefits due to lookahead are over-shadowed by the benefits due to pruning. Combining pruning

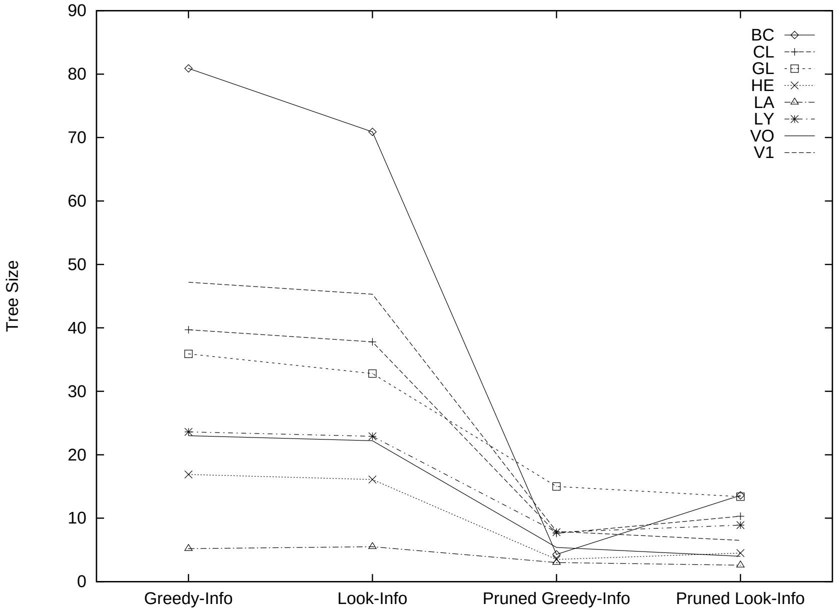

Figure 4.16: Effect of one level lookahead on tree size for eight real-world data sets. The tree sizes with and without lookahead, and with and without pruning are shown for information gain.

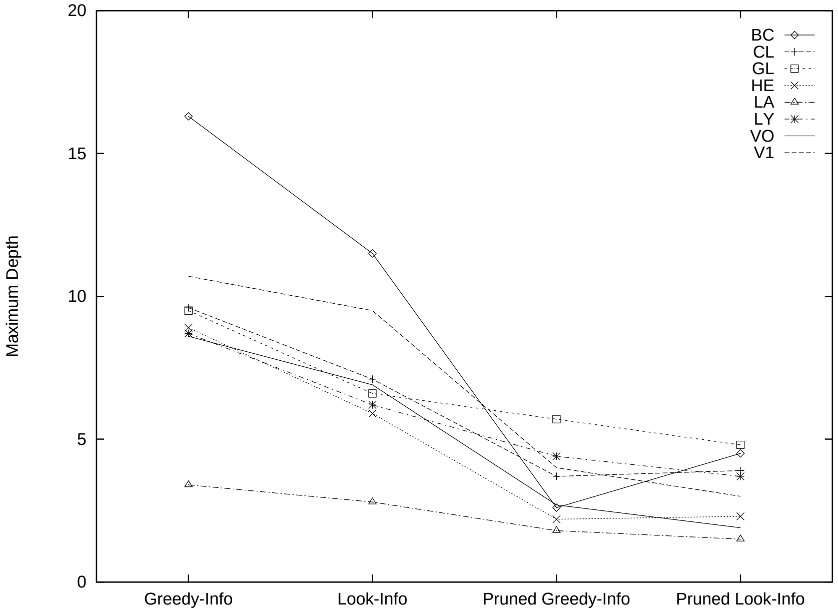

Figure 4.17: Effect of one level lookahead on maximum tree depth for eight real-world data sets. The depths with and without lookahead, and with and without pruning are shown for information gain (IG).

with lookahead doesn't produce any significant improvements over just pruning.

Overall, the results of our experiments with real data substantiate our results with the synthetic data. Limited lookahead did not benefit significantly in terms of classification accuracy or size, despite the fact that it is enormously more expensive. It helped produce shallower trees, but tree post-processing techniques (pruning in this case) which are much less expensive than lookahead were adequate to obtain comparable benefits. Finally, both goodness measures we used (Gini index and information gain) exhibited pathology on the real-world domains also.

4.5 Conclusions

Most existing decision tree induction methods use a greedy strategy – decisions in the tree are induced one node at a time. It is known that all these methods are necessarily suboptimal [174]. It is desirable, particularly in view of the huge increase in the available computing power, to have techniques that can systematically bridge the gap between greedily induced trees and the optimal trees. Limited lookahead search is commonly believed to be one such technique.

In this chapter, we described experiments that aimed to precisely quantify the benefits of limited lookahead search for tree induction. We restricted ourselves to one level lookahead. Our experiments used both synthetic and real world domains. For the experiments with synthetic data, we defined two moderately large classes of concepts and induced trees with greedy induction and limited lookahead on *all possible concepts* in these classes. For the experiments with real world data, we used seven domains from the UCI repository of machine learning databases. The tree induction methods we used are very similar to CART [44] and ID3 [391].

The benefits of limited lookahead search in all our experiments are marginal in spite of the enormous increase in the computational complexity. Greedy induction consistently produces trees that are as accurate and small as those produced with lookahead. The only significant benefit of lookahead is in producing slightly shallower trees than greedy induction. However, tree post-processing techniques, such as pruning and "balancing", narrow the difference between greedy and limited lookahead induction even further. Moreover, we measured maximum depth in all our experiments. The longest paths in a tree may be trying to split a small area of the attribute space, and thus may be used very rarely for actual classification. One can claim that lookahead produces shallow trees only on the basis of *expected* depth, the number of tests needed to classify an unseen example. We present experiments in the next chapter, which show that the expected depth of greedily induced trees is very close to the optimum, leaving little scope for improvement by lookahead or other techniques.

Not only does lookahead search not produce significant benefits, it can actually *hurt* tree quality. We discovered several synthetic and real world data sets for which limited lookahead search produces worse trees than greedy induction, in terms of accuracy, tree size and max. depth. Intuitively, doing more search (lookahead) should produce better decision trees, just as deeper search in game trees (e.g., for chess) produces better game-playing programs. However, it has been observed that for some games, deeper search can actually produce an inferior program, both with two players [360] and with multiple players [356]. Decision trees, one can argue, are analogous to a one-player game tree. Our discovery that deeper search can lead to inferior decision trees thus extends the earlier pathology results to a new domain.

It is intriguing that limited lookahead search can produce inferior decision trees as compared to greedy search. Pathology may be caused by the way heuristic goodness measures are defined. Greedy methods grow a decision tree by optimizing measures such as class entropy or diversity at each node of the tree (Section 2.3). However, as indicated by pathology, each such optimization is not necessarily improving the tree globally. It is commonly believed that information gain helps induce shallow trees [391, 403]. Our findings that *Look-Info* generally produces shallower trees than *Greedy-Info* are in accordance with this common belief. However, pathology in terms of maximum depth indicates that a split that optimizes information gain can in fact lead to a deeper tree.

Finally, a word of caution. Most of our experiments in this chapter are based on synthetic data. Experiments with synthetic data are necessarily limited in the generality and applicability of their conclusions. We used synthetic data in this chapter mainly to show the *existence* of pathology. To determine how often pathology occurs in real-world, more detailed analyses will be necessary.

Chapter 5

On the effectiveness of the greedy heuristic

Chapter 4 presented extensive experimentation which indicated that one-level lookahead does *not* produce better trees than the traditional greedy tree induction algorithm. This is surprising, as one would expect that more search should mean better results. One plausible explanation for this seemingly unintuitive result is that greedily induced trees are themselves so close to the optimal that improvement is hard to achieve. The current chapter investigates in this direction: it attempts to measure how close greedily induced trees are to the optimal.

We cannot, however, use the same synthetic data here as we had used in Chapter 4. Classes of small, shallow decision trees were chosen for the lookahead experiments so that one-level lookahead can have an effect. On these trees, there can obviously be not much difference between the greedily constructed and optimal trees. For the experiments in this chapter, we use much larger trees, varying dimensionality, varying levels of noise, etc.

What we mean by greedy tree induction is the standard top-down method: recursively do the following until no more nodes can be split: choose the best possible test at the current node according to some *goodness measure* and split the current node using that test; after a complete tree is grown, prune it back to avoid overfitting the training data. The choice of a "best" test is what makes this algorithm greedy. The best test at a given internal node of the tree is only a locally optimal choice; and a strategy choosing locally optimal splits necessarily produces suboptimal trees [174]. Optimality of a decision tree may be measured in terms of prediction accuracy, size or depth. It should be clear that it is desirable to build optimal trees in terms of one or more of these criteria. Maximizing classification accuracy on unseen data (within the constraints imposed by the training data) is obviously desirable. Smaller, shallower decision trees imply better comprehensibility and computational efficiency. Shallow trees are also more cost-effective, as the depth of a tree is a measure of its classification cost. However, because the problem of building optimal trees is known to be intractable (Section 2.6.1), a greedy heuristic might be wise given realistic computational constraints.

Although the greedy approach is suboptimal, it is commonly believed to produce reasonably good trees. In the current chapter, we attempt to verify this belief. We ask the question, if we had unlimited resources and could compute the optimal tree, how much better should we expect to perform? An alternative way of asking the same question is, what is the penalty that decision tree algorithms pay in return for the speed gained by the greedy heuristic? We quantify the goodness of greedy tree induction empirically in this chapter, using the popular decision tree algorithms, C4.5 [398] and CART [44]. We induce decision trees on thousands of synthetic data sets and compare them to the corresponding optimal trees, which in turn are found using a novel map coloring idea. We measure the effect on greedy induction of variables such as the underlying concept complexity, training set size, noise and dimensionality. The main observations from the experiments in this chapter are the following.

- The expected depth of greedily induced decision trees is consistently very close to the optimal.
- The prediction accuracy of a greedily induced tree is not dependent on concept complexity, provided there is adequate training data.
- Greedily induced trees are not much larger than the optimal, even for complex concepts. However, the variance in tree size increases with increase in concept complexity and/or dimensionality.
- There is almost no difference between the greedy goodness measures of CART and C4.5, in terms of the predictive accuracy, size or depth of trees they generate.

Section 5.1 describes our experimental method and Section 5.2 presents the results. Section 5.3 provides general conclusions.

5.1 Experimental setup

Our experimental framework is quite simple — we use C4.5 [398] and CART [44] to induce decision trees on a large number of random data sets, and in each case we compare the greedily induced tree to the optimal tree. The implementation of this framework raises some interesting issues.

Optimal Decision Tree for a Training Set. The problem of computing the shallowest or smallest decision tree for a given data set is NP-complete (Section 2.6.1), meaning that it is highly unlikely that a polynomial solution will be found. Previous studies that attempted comparisons to optimal trees (e.g., [97]) used approaches like dynamic programming to generate the optimal trees. Because it is slow, this option is impractical for our study, in which we use hundreds of thousands of artificial data sets. Our solution is to first generate a random decision tree D, and *then* generate data sets for which D is *guaranteed* to be the optimal tree. The main idea behind ensuring the optimality of a random decision tree is coloring its leaf nodes with appropriate class labels.

An *instance* is a real valued vector $X_i = (x_{i1}, x_{i2}, ..., x_{id})$ plus a class label c_i. x_is are the *attributes* of X_i, and d is its dimensionality. Consider a binary decision tree D in two attributes. (The ensuing argument applies to arbitrary dimensions.) D induces a hierarchical partitioning of the attribute space, which can be drawn as a map M. The boundaries of M are the splits (test nodes) in D, and the regions of M are the leaf nodes in D. Assuming that each leaf node of D contains instances of only one class, we can color M by assigning a distinct color to each class in D. Now consider a data set S consistent with D, which has the additional property that S requires every leaf node of D, i.e., every leaf node of D contains at least one instance of S.

It should be clear that D is the smallest binary decision tree consistent with S, *provided* no two neighboring regions of M have the same color. Informally, any decision tree that has fewer leaves than D needs to either ignore some decision regions of D, or merge (parts of) two or more regions into one. The former possibility is ruled out because S requires all decision regions in G. The latter is impossible because no decision regions of the same color are adjacent, so no two regions can be merged. Hence, *any* decision tree consistent with S has to have at least as many leaf nodes as D. Moreover, if D was a perfectly balanced tree to start with, then any decision tree consistent with S has to be at least as deep as D.

In our experiments, we start with perfectly balanced, empty trees. We then generate random tests at the decision nodes, ensuring that no leaf region is empty. Finally we color the leaves to ensure optimality with respect to size, using the following procedure. We first compute the adjacency information of the leaf nodes. After initializing the class labels at all leaf nodes to k ($\geq$ number of leaves), we go back and change the label of each leaf to be the smallest number in $[1, k]$ that is not yet assigned to any neighbor. This heuristic procedure worked quite well in all our experiments. (For instance, decision trees of 64 leaf nodes in the plane were colored with 5 classes on average.) Fig. 5.1 shows a sample random decision tree in 2-D, along with the class labels assigned by the above coloring procedure.

Tree Quality Measures. In all our experiments, we report tree quality using six measures:

- Classification accuracy: accuracy on the training data;
- Prediction accuracy: accuracy on an independent, noise-free testing set;
- Tree size: number of leaf nodes;
- Maximum depth: distance from the root to the farthest leaf node; (distance from A to B is the number of nodes between, and including, A and B)
- Average depth: mean distance from the root to a leaf node in the tree;
- Expected depth: number of tests needed to classify an unseen example. We compute expected depth by averaging, over all the examples in the testing set, the length of

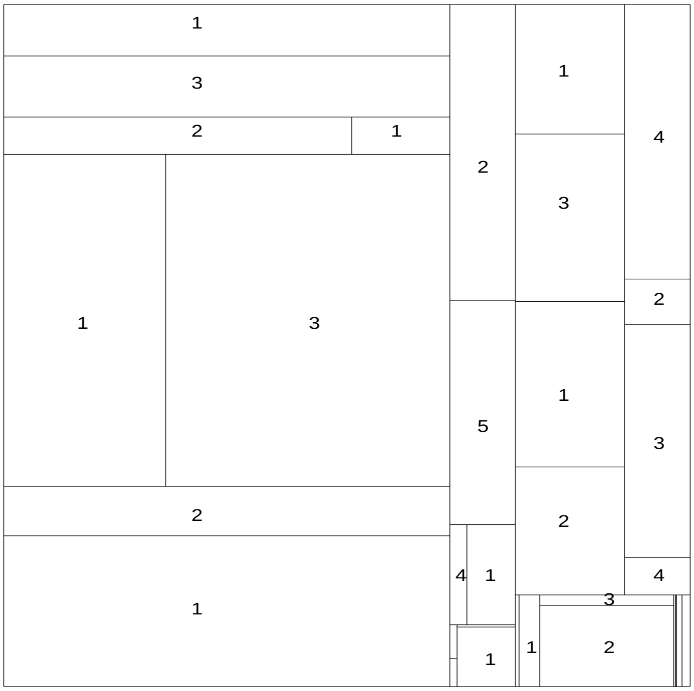

Figure 5.1: The partitioning induced by a random decision tree of 32 leaf nodes. Class labels assigned by our coloring procedure are shown (for most nodes).

the path that the example followed in the tree. If all attributes have equal costs of measurement, expected depth is equal to the expected cost of using the tree for classifying one example.

Control Variables. The effectiveness of greedy induction can not be measured independently of training data characteristics. For instance, if the training data is very noisy, it is likely that no induction method will be able to generate accurate trees. In this paper, we study the effectiveness of greedy induction in controlled settings with respect to the following parameters:

- concept complexity (measured as the size of the optimal decision tree),
- size of the training set,
- amount and nature of noise in the training data (noise in class labels versus noise in attributes), and
- dimensionality (number of attributes).

Tree Induction Methods Used. The tree induction methods we use are C4.5 [398] and CART [44]. One main difference between C4.5 and CART is the *goodness criterion*, the criterion used to choose the best split at each node. C4.5 uses the information gain [38] criterion, whereas CART uses either the Gini index of diversity or the twoing rule. All the experiments in this chapter were repeated using information gain, Gini index and twoing rule. *In no case* did the results show statistically significant differences between goodness measures—the differences in accuracies, sizes *and* measurements of depth were always much less than one standard deviation. For brevity, we report only the results with information gain (i.e., C4.5) in the rest of this chapter. We implemented all the goodness measures using the OC1 system (Chapter 3). Although C4.5 and CART differ in respects other than the goodness measures, we have not implemented these differences. In the experiments in which the training data is noise-free, no pruning was used with either method. In the experiments using noisy training sets, we augment both methods with cost complexity pruning [44], reserving 10% of the training data for pruning.

5.2 Experiments

This section describes six experiments, each of which is intended to measure the effectiveness of greedy induction as a function of one or more control variables described in Section 5.1. The procedure is more or less the same for all experiments.

– generate 100 random trees with no class labels;

[38] Quinlan suggested gain ratio as an improvement over information gain. However the two measures are equivalent in our experiments as all our decision trees are binary.

```
– for each tree $D_{opt}$ generated in the above step:
    – color $D_{opt}$ with class labels (Sec. 5.1);
    – generate a large, noise-free testing set for which $D_{opt}$ is optimal;
    – generate 50 training sets using $D_{opt}$;
    – for each training set $T$:
        – greedily induce a tree $D$ on $T$;
        – record the size, depth, accuracy etc. of $D$ and $D_{opt}$;
– report the mean and std. dev. of the quality measures for the 5000 trees.
```

The instances in the training and testing sets are always generated uniformly randomly, and are labeled using the optimal decision tree. The size of the testing set is linearly dependent on the concept complexity and the dimensionality of the data, whereas the size of the training set is a control variable. More precisely, $|T| = C * (D - 1) * 500$, where $|T|$ is the size of the testing set, C is the concept complexity and D is the number of attributes. For instance, for a size 16 concept in 4 dimensions, we use a testing set of size $16 * (4 - 1) * 500 = 24,000$. We ensure that no subtree of the optimal decision tree is consistent with the testing set.

In all the tables in this chapter, each entry comprises of the average value of a tree quality measure over 5000 trees and the standard deviation (one σ). Numbers in parenthesis correspond to the optimal trees. The σ's are omitted when they are zero. Optimal values are omitted when their values are obvious. The optimal trees always give 100% prediction accuracy in our experiments, because the testing set has no noise. In addition, they give 100% classification accuracy when the training data is noise-free.

Experiment 1: The purpose of this experiment is to evaluate the effectiveness of greedy induction as a function of concept complexity. All training sets comprise of 1000 random 2-D instances. There is no noise in the training data. Table 5.1 summarizes the results. Greedily induced trees give 100% classification accuracy in this experiment as there is no pruning.

Observations: The prediction accuracy of greedily induced trees decreases with increase in concept complexity. This can be either be due to the inadequacy of greedy search or due to inadequate training data. (The training set size remained at 1000 though the concept complexity increased from 8 to 64.) In Experiment 2, we increase the size of the training set in proportion with the concept complexity, in order to better isolate the effects due to greedy search.

The difference between the sizes of greedily induced and optimal trees increases with increasing concept complexity. However, it can be seen on closer observation that the variances, not just the differences in size, are increasing. Greedily induced tree sizes are just more than one σ away from the optimal in 3 out of 4 rows in Table 5.1, and less than one std. dev. away for concepts of size 64.

Optimal	Training	Prediction	Tree Size	Depth		
Tree Size	Set	Accuracy		Maximum	Average	Expected
8	1000	99.5±0.1	9.8±1.7	4.8±0.7 (3)	3.6±0.3 (3)	2.9±0.3 (3)
16	1000	98.7±0.3	20.7±3.3	7.2±1.0 (4)	5.0±0.4 (4)	3.9±0.4 (4)
32	1000	97.2±0.6	40.4±6.8	9.3±1.0 (5)	6.3±0.5 (5)	5.0±0.5 (5)
64	1000	94.3±0.9	71.7±10.3	11.5±1.2 (6)	7.4±0.5 (6)	5.8±0.5 (6)

Table 5.1: Effect of concept complexity. There is no noise in data, so all trees have 100.0% classification accuracy. Numbers in parenthesis are for optimal trees.

Optimal	Training	Prediction	Tree Size	Depth		
Tree Size	Set	Accuracy		Maximum	Average	Expected
8	200	97.5±0.7	8.5±1.5	4.4±0.6 (3)	3.4±0.3 (3)	2.8±0.3 (3)
16	400	97.1±0.7	17.5±3.3	6.6±0.9 (4)	4.7±0.5 (4)	3.8±0.5 (4)
32	800	96.6±0.7	38.0±7.1	9.1±1.0 (5)	6.2±0.5 (5)	4.8±0.5 (5)
64	1600	96.4±0.6	76.3±12.2	11.6±1.2 (6)	7.5±0.6 (6)	5.8±0.6 (6)

Table 5.2: Effects of concept complexity and training set size. There is no noise in the data, so all trees give 100.0% classification accuracy. Numbers in parentheses correspond to the optimal trees.

The maximum depth measurements in Table 5.1 show that greedily induced trees can have decision paths which are about twice as long as those in the optimal trees, even for moderately complex concepts. However, the average depth measurements show that the decision paths in greedily induced trees only have about one test more than those in the optimal trees. In terms of the third depth measurement, the expected depth, greedily induced trees are almost *identical* to the optimal ones, for all the concept sizes considered in this experiment. This is a very desirable, although somewhat counterintuitive, trend which is seen consistently throughout our experiments. (Note that no pruning was used in this experiment.)

Experiment 2: The purpose of this experiment is to isolate the effects of concept complexity on greedy induction, from those of the training set size. The size of the training sets now grows linearly with the concept complexity—25 training points on average are used per each leaf. There is no noise. Table 5.2 summarizes the results.

Observations: It is interesting to note that the prediction accuracy does *not* drop as much with increase in concept complexity as it does in Experiment 1. In fact, when the

Concept Size	Prediction Accuracy	Tree Size	Depth		
			Maximum	Average	Expected
8	98.3±0.5	8.9±1.6	4.4±0.8 (3)	3.4±0.4 (3)	3.0±0.4 (3)
16	96.7±0.9	17.2±3.2	6.3±1.0 (4)	4.6±0.4 (4)	3.9±0.4 (4)
32	93.9±1.4	31.1±6.2	8.2±0.9 (5)	5.6±0.4 (5)	4.8±0.5 (5)
64	89.4±2.0	50.2±9.0	10.1±1.1 (6)	6.6±0.5 (6)	5.7±0.5 (6)

Table 5.3: Effect of concept complexity. All classes are equally likely. As there is noise, all trees give 100.0% classification accuracy.

concept complexity increased by 8-fold (from 8 to 64), the accuracy went down by just more than one standard deviation. In addition, none of the differences in tree size between greedily induced and optimal trees in Table 5.2 are more than one standard deviation. This is surprising, considering no pruning was used in this experiment. In terms of the three depth measures, the observations made in Experiment 1 hold here also.

Comparing the entries of Tables 5.1 and 5.2, line by line, one can see the effect of the training set size on prediction accuracy. When the training set size increases, the prediction accuracy increases and its variance goes down. In other words, the more (noise-free) training data there is, the more accurately and reliably greedy induction can learn the underlying concept.

Experiment 3: This experiment is intended to evaluate the effects of training data distribution on the quality of greedily induced trees. Each training set comprises of 100 planar instances *per each class*. There is no noise. Data in each class is uniformly randomly distributed. Table 5.3 summarizes the results.

Consider a situation in which a particular class is much rarer than the others, but it is important to predict that class accurately. (e.g., Patients with cancer are a lot rarer than those without cancer.) The uniform distribution in Experiments 1 and 2 may generate too few training instances in the rare class, thus producing unbalanced training sets. The data distribution in this experiment ensures that all classes are well-represented in the training data. It is common practice in the real world to collect training instances in all (known) classes before using a learner on the data.

Observations: Prediction accuracy decreases in this experiment as the concept complexity increases, just as in Experiment 1. The reason for this decrease in accuracy may again be the inadequacy of training data or the inadequacy of the greedy heuristic for complex concepts. [39]

[39] Note that the training set sizes varied slightly across concept complexities in this experiment unlike in Experiment 1. The average number of classes assigned to the optimal trees by the coloring

Class	Classification	Prediction	Tree Size	Depth		
Noise	Accuracy	Accuracy		Maximum	Average	Expected
0	100.0 (100.0)	93.9±1.4	31.1±6.2	8.2±0.9	5.6±0.4	4.8±0.5
5	92.1±1.3 (95.1±0.01)	89.5±2.4	21.9±5.1	7.0±0.8	4.9±0.5	4.4±0.4
10	87.7±1.3 (90.5±0.02)	88.2±2.6	22.2±5.1	7.0±0.8	4.9±0.4	4.4±0.4
15	83.5±1.3 (86.1±0.05)	86.6±2.9	22.4±5.4	7.0±0.8	4.9±0.5	4.4±0.4
20	79.7±1.4 (81.9±0.05)	84.9±3.1	22.7±5.2	7.1±0.8	4.9±0.5	4.4±0.4
25	76.1±1.4 (77.8±0.03)	83.1±3.4	23.3±5.7	7.1±0.8	4.8±0.5	4.4±0.4

Table 5.4: Effects of noise in class labels. Concept complexity=32. Numbers in parentheses are for the optimal trees.

Tree sizes increase much slower with increasing concept complexity in this experiment, than in Experiment 1. In fact, the greedily induced trees are significantly *smaller* than the optimal trees when concept size = 64. This result, however, can not be taken in support of the greedy heuristic, for two reasons. First, for concepts of size 64, the training set size was about 500. It is possible that several decision regions are not even represented by such a small training set. Moreover, as all classes have the same number of training instances, larger classes (classes that span a lot of decision regions) have a large probability of having decision regions with no training instances in them. Both these reasons can cause overly general decision trees that are smaller than the optimal trees but much less accurate. Similar reasoning can be used to explain smaller values of maximum and average depth in this experiment. The expected depth, however, is very close to the expected depth of the optimal trees, as in Experiment 1.

Experiment 4: This experiment is intended to evaluate the effectiveness of greedy induction in the presence of noise in class labels. The training sets are all in 2-D, and consist of 100 instances per class, uniformly randomly distributed in each class. k% noise is added into each training set by incrementing by 1 the class labels of a random k% of the training points. All concepts are of size 32, so all optimal tree depth values are equal to 5.0. Pruning was used when noise level is greater than 0%. Table 5.4 summarizes the results.

Observations: As is expected, the classification and prediction accuracies decrease when the amount of noise is increased. The tree size and depth measurements vary significantly when the first 5% of noise is introduced (obviously because pruning is started), and remain steady thereafter.

algorithm (Section 5.1) were 3.43, 4.04, 4.55 and 5.02 for concept sizes 8, 16, 32 and 64. As we used 100 training points per class, the size of the training set increased by about 50 when concept complexity doubled.

Attribute Noise	Classification Accuracy	Prediction Accuracy	Tree Size	Depth		
				Maximum	Average	Expected
0	100.0 (100.0)	93.9±1.4	31.1±6.2	8.2±0.9	5.6±0.4	4.8±0.5
5	95.2±1.3 (98.0±0.4)	90.0±2.3	22.2±5.3	7.0±0.8	4.9±0.5	4.4±0.4
10	93.5±1.4 (96.0±0.7)	88.7±2.6	22.6±5.5	7.0±0.8	4.9±0.5	4.4±0.4
15	92.1±1.6 (94.1±1.0)	87.4±2.8	23.3±5.6	7.0±0.8	5.0±0.5	4.4±0.4
20	90.7±1.8 (92.2±1.3)	86.2±3.1	23.7±5.6	7.0±0.8	4.9±0.5	4.4±0.4
25	89.4±2.0 (90.6±1.6)	85.0±3.4	23.7±5.5	7.0±0.8	4.9±0.5	4.3±0.4

Table 5.5: Effects of noise in attribute values. Concept complexity=32.

One needs to be careful in analyzing the results of experiments 4 and 5, in order to separate out the effects of noise and the effect of the greedy search. What we want to investigate is whether the greedy heuristic becomes less and less effective as the noise levels increase, or if it is robust. For instance, the fact that the classification accuracy decreases linearly with increase in noise in Table 5.4 is perhaps *not* as significant as the fact that the prediction accuracy decreases more slowly than classification accuracy. This is because the former is an obvious effect of noise whereas the later indicates that greedy induction might be *compensating* for the noise.

Several patterns in Table 5.4 argue in favor of the effectiveness of pruning, which has come to be an essential part of greedy tree induction. Classification accuracies of the greedy trees are close to, and *less than*, those of the optimal trees for all the noise levels, indicating lack of overfitting. Prediction accuracies of greedily induced trees with pruning are *better* than their classification accuracies, again indicating that there is no strong overfitting. Tree size and depth measurements remained virtually unchanged in the presence of increasing noise, certifying to the robustness of pruning.

Experiment 5: This experiment is similar to the previous one, in that we measure the effectiveness of greedy induction as a function of noise in the training data. However, this time we consider noise in attribute measurements. The training sets again comprise 100 2-D instances per class, uniformly randomly distributed in each class. $k\%$ noise is introduced into each training set by choosing a random $k\%$ of the instances, and by adding an $\epsilon \in [-0.1, 0.1]$ to each attribute. All the concepts are of size 32, so all the optimal depth measurements are equal to 5.0. Cost complexity pruning was used in cases where the noise level was greater than 0%. The results are summarized in Table 5.5.

Observations: There results with attribute noise (Table 5.5) and noise in class labels (Table 5.4) are very similar, except for the classification accuracies. The values for prediction accuracy, tree size and depth measurements in the presence of $k\%$ noise are almost the same

#Dim.	Prediction Accuracy	Tree Size	Depth		
			Maximum	Average	Expected
2	98.7±0.3	20.7±3.3	7.2±1.0	5.0±0.4	3.9±0.4
4	98.3±0.7	23.9±6.0	6.6±0.9	5.0±0.5	4.0±0.4
8	98.0±0.8	24.5±6.5	6.3±0.9	4.9±0.5	4.1±0.2
12	97.9±0.9	25.4±6.8	6.3±0.9	4.9±0.5	4.1±0.2

Table 5.6: Effect of dimensionality. Training set size=1000. As there is no noise, all trees give 100.0% classification accuracy. Concept complexity=16.

whether the noise is in attribute values or class labels. The classification and prediction accuracies decrease with increasing noise. The tree size and depth measurements decrease when the first 5% of the noise is introduced (due to pruning) and remain steady thereafter.

However, introducing k% attribute noise is not equivalent to introducing k% class noise. Changing the attributes of an instance by a small amount affects the classification of only those instances lying near the borders of decision regions, whereas changing the class labels affects the classification of all the instances involved. This can be seen from the classification accuracies of the optimal trees in Tables 5.4 and 5.5. The classification accuracy of the greedy trees is quite close to, and less than that of the optimal trees in both tables. All the prediction accuracy values in Table 5.5, unlike those in Table 5.4, are less than the corresponding classification accuracies.

Experiment 6: Our final experiment attempts to quantify the effect of dimensionality on the greedy heuristic. All the training sets consist of 1000 uniformly randomly generated instances, with no noise, as in Experiment 1. No pruning was used. All concepts are of size 16, so the optimal tree depths are 4.0. Table 5.6 summarizes the results.

Observations: There is *no* statistically significant difference in *any* tree quality measure when dimensionality of the data is increased from 2 to 12. This result is surprising because, intuitively, higher dimensional concepts should be more difficult to learn than lower dimensional ones, when the amount of available training data does not change. Our experiments indicate that the effects due to dimensionality do not seem to be as pronounced as the effects due to concept complexity (Table 5.1) or noise. The quantity that does increase with increasing dimensionality is the variance. Both prediction accuracy and tree size fluctuate significantly more in higher dimensions than in the plane. This result suggests that methods that help decrease variance, such as combining the classifications of multiple decision trees (Section 2.5.7), may be useful in higher dimensions.

5.3 Discussion

In this chapter, we presented six experiments for evaluating the effectiveness of the greedy heuristic for decision tree induction. In each experiment, we generated thousands of random training sets, and compared the decision trees induced by C4.5 and CART to the corresponding optimal trees. The optimal trees were found using a novel graph coloring idea.

We summarize the main observations from our experiments below. [40] Where relevant, we briefly mention related work in the literature.

- *The expected depth of greedily induced decision trees was consistently very close to the optimal.* Garey and Graham (1974) showed that a recursive greedy splitting algorithm using information gain (*not* using pruning) can be made to perform arbitrarily worse than the optimal in terms of expected tree depth. Goodman and Smyth (1988) argued, by establishing the equivalence of decision tree induction and a form of Shannon-Fano prefix coding, that the average depth of trees induced by greedy one-pass (i.e., no pruning) algorithms is nearly optimal.
- *Cost complexity pruning* [44] *dealt effectively with both attribute and class noise.* However, the accuracies on the training set were overly optimistic in the presence of attribute noise.
- Greedily induced trees became less accurate as the concepts became harder, i.e., as the optimal tree size increased. However, *increasing the training data size linearly with concept complexity helped keep the accuracy stable.*
- Greedily induced trees were not much larger than the optimal, even for complex concepts. However, *the variance in tree size was more for higher dimensional and more complex concepts.* Dietterich and Kong (1995) empirically argued that even in terms of prediction accuracy, variance is the main cause for the failure of decision trees in some domains.
- *For a fixed training set size, increasing the dimensionality did not affect greedy induction as much as increasing concept complexity or noise did.* Several authors (e.g., [156]) have argued that for a finite sized data with no *a priori* probabilistic information, the ratio of training sample size to the dimensionality must be as large as possible. Our results are consistent with these studies. However, with a reasonably large training set (1000 instances), the drop in tree quality was quite small in our experiments, even for a 6-fold (2 to 12) increase in dimensionality.
- *The goodness measures of CART and C4.5 were identical in terms of the quality of trees they generated.* It has been observed earlier (e.g.,[44, 325]) that the differences

[40] As mentioned at the end of Chapter 4, experimental results with synthetic data are necessarily limited in their applicability. We had experimented with a spectrum of training data characteristics in this chapter, so we hope that our conclusions will prove to be general.

between these goodness criteria are not pronounced. Our observation that these measures consistently produced identical trees, in terms of six tree quality measures, in a large scale experiment (involving more than 130,000 synthetic data sets) strengthens the existing results. Note that the fact that we only used binary splits in real-valued domains may be one reason why information gain, Gini index and twoing rule behaved similarly.

Many researchers studied ways to improve upon greedy induction (Section 2.5.4), by using techniques such as limited lookahead search and more elaborate representations than trees. The results in the current chapter throw light on why it might be difficult to improve upon the simple greedy algorithm for decision tree induction.

Chapter 6

Domain specific data massaging: Two Illustrations from Astronomy

The goal of this thesis is to find techniques that help build "good" decision trees from data. Chapter 3, 4 and 5 described ways of achieving this aim through algorithmic extensions such as non axis-parallel splits and limited lookahead search. However, algorithmic enhancements alone are seldom adequate to grow good trees, especially if the data is not in the "right" form. For instance, if the features do not contain sufficient information about the concept to be learnt, or if the concept can not be concisely represented using the given features, no learning method can expect to do well. "Massaging" the data into a form suitable for the given learning method is a crucial step in building good real-world classifiers.

Although anyone that applied data exploration methods to real world data ought to be aware of the importance of data massaging, very few papers seem to explicitly discuss data massaging. The most detailed discussion about data massaging that the author is aware of is from two recent workshop proceedings [222, 332]. Both these forums discussed the problem of structuring, or engineering the data into a form suitable for inductive learning. Each paper at these two workshops, like the current chapter, argues data massaging mostly relies on domain knowledge. However, we feel that some domain-independent data massaging is possible, and pursue that issue in Chapter 7. More precisely, we discuss automated ways of engineering *any* data set into a form more appropriate for a particular learning method.

The data massaging procedure to use for a given problem depends heavily on the problem and on the learning method to be used. Data massaging encompasses tasks such as choosing a set of features that is likely to be useful for learning the concept, weeding out inconsistant/redundant instances and features, and sometimes even defining *what* to learn. Most of the time the classification problem needs to be understood using domain knowledge, so that the data can be re-represented in a suitable form.

The current chapter illustrates domain-specific data massaging in the context of two new classification problems in astronomy, namely:

1. Identifying cosmic rays in the Hubble Space Telescope images, and
2. Classifying stars and galaxies in Sloan Digital Sky Survey images.

The raw data for both problems consisted of images of parts of the sky, which were not directly useful to the classification method. The main effort in solving the problems was spent on finding the appropriate training data points (objects), and the appropriate features to represent the objects. The author worked as part of a group of three astronomers and two computer scientists [41] for about 3.5 years to solve these problems. We iterated the following (roughly defined) procedure several times to before finding the best data representation for each of these problems.

1. Using expert knowledge as well as empirical observation, define a set of features that are potentially good discriminators.
2. Build and test classifiers using these features. Stop if the astronomers are satisfied with the performance of the classifier.
3. Modify the feature set and the data. Go to step 1.

The rest of this chapter outlines this iterative process, and presents our results for the above problems. Our primary goal in this work has been to develop techniques that classify with high accuracy, in order to ensure that celestial objects are not stored in the wrong catalogs. In addition, classification time must be fast due to the large number of classifications and to future needs for on-line classification systems. For both problems, we obtained small, highly accurate decision tree classifiers using OC1 (Chapter 3).

Outline

Section 6.1 reports on our experiments for using decision tree classifiers to identify cosmic ray hits in Hubble Space Telescope (HST) images. This method produces classifiers with over 95% accuracy using data from a single, unpaired image. Accurate prediction was achieved both for the old and aberration-corrected images from the Hubble Telescope. Our experiments indicate that this accuracy will get even higher if methods for eliminating background noise improve.

Section 6.2 describes ongoing work on the classification of stars and galaxies in the Sloan Digital Sky Survey (SDSS) images. SDSS is a large-scale survey of the northern hemisphere, expected to produce photographic images of hundreds of millions of stars, galaxies and quasars, and high resolution spectra for millions of bright galaxies and quasars.

[41] The *Astroexplorer* group comprised of the author and Steven Salzberg from the Department of Computer Science, Johns Hopkins University; Holland Ford and Rupali Chandar from the Department of Physics and Astronomy, Johns Hopkins University; and Rick White from the Space Telescope Science Institute. The work described in this chapter and Appendix A was done jointly with these researchers.

We were able to obtain decision trees with very high classification accuracies in this domain also, using noisy and extremely faint object lists.

As our main emphasis in this thesis is on machine learning, we present most astronomy-specific details separately in Appendix A. This is also appropriate because the author's contributions are minor in the astronomy-specific parts of the project. As the purpose of this chapter is to illustrate the importance of data massaging, and not to find the best algorithm for the given data, we do *not* attempt comparison of different classification methods here — we use the default OC1 system (Chapter 3) for all our experiments. Some results with other classification methods on these data sets can be found in [424].

6.1 Cosmic ray hits in Hubble Space Telescope images

This section describes the results of an effort to identify accurately the types of noise commonly present in Hubble Space Telescope (HST) data, especially cosmic ray hits, which are very common in the Wide Field Planetary Camera (WFPC) images. Below we first outline the star/cosmic ray classification problem for HST images. We then present our experiments with the data from the aberrated HST images, describing in detail the data massaging involved. We describe in Section 6.1.7 our results on the aberration-corrected HST images, which are being obtained with the new WFPC (WFPC2) that was installed in the HST by the crew of the Space Shuttle Endeavor in December 1993.

6.1.1 The task

Cosmic rays (CR) are a common type of noise in HST images. During a typical 20-minute exposure, each of the four CCDs [42] detect approximately 2,000 CR events. Many of the low amplitude CR events look like faint stars, and need to be filtererd out before cataloging the stars.

The method commonly used to filter cosmic rays is to take two images of the same region in the sky at different times, and to identify as stars objects that appear above noise in both images. The goal of our work was to find classifiers that separated CRs and stars with high reliability in *individual* images. Eliminating the need for split exposures saves the significant amount of spacecraft time that is required to prepare the camera for the second exposure. We also wish to eliminate the noise penalty that is incurred by taking two exposures. (Whenever the exposure is split and then subsequently summed after CR removal, the noise is increased by $\sqrt{2}$. See Appendix A for details.)

We began our study by selecting two images of a field in the nearby galaxy M81 taken at two different times. Figure 6.1 shows part of an image from one of the four CCDs (CCD WF1). The entire field, which was centered on a portion of a spiral arm in M81,

[42] Loosely speaking, Charge Coupled Detectors (CCDs) are akin to photographic plates. Incident photons or rays leave electron-hole pairs on the CCDs. The magnitude of the current generated is a measure of the intensity of the incident light.

contains thousands of faint stars. Because the stars are faint and close to one another, the image is a difficult test case. Moreover, this image provides a difficult test case because some of the CCD's fall directly on the galactic bulge.

Our first objective was to create a highly reliable catalog of the positions and identities of stars and cosmic rays. We used standard astronomy software packages (see Appendix A for details) to detect and locate objects (stars in this case) that are statistically above the noise in each image, and reject objects (CRs) that appear in only one image. Note that we use the combined images **only** to find positions of the objects. Features are always extracted from single images.

A relatively small number of cosmic rays (287 out of 4689 objects) were superposed on the images of stars. These *blends* (stars plus CRs) are a third class of objects that we wish to classify at a later date when we have a sufficient number of examples. We removed these blends from our database for all the experiments described below.

Experimental Method

In some of our experiments, we used *cross-validation* to estimate accuracy. As a further check on these accuracy figures, we collected additional data from a different CCD and used a tree built on the first CCD to classify this data. The database consisted of two sets of objects, taken from two adjacent CCDs, WF1 and WF2, from a 4-CCD array. WF1 was used to develop the decision tree classifiers, and give initial estimates of accuracy. We reserved WF2 to provide an independent test of the accuracy of each classifier. We chose adjacent CCDs in order to equalize global conditions such as background sky brightness as much as possible. Including blends, WF1 contained 2430 objects and WF2 contained 2484 objects. After removing the blends, WF1 contained 2259 objects and WF2 contained 2368 objects. In both cases about 60% of the objects were cosmic rays.

To estimate the accuracy of a classifier on the WF1 data, we ran a five-fold *cross-validation* experiment. Each of the WF1 numbers given in below is an average of ten 5-fold cross-validation (CV) experiments. "Tree size" refers to the number of leaf nodes; i.e., the number of regions that the data is divided into by the tree. After training our program on the WF1 data, we used the same decision trees, without *any* further training, to classify the WF2 data. The experimental design here was simple: we built a tree using the entire WF1 data set of 2259 objects (rather than 4/5 of the objects, as was done in the cross-validation study above), and then calculated the classification accuracy of that tree on the WF2 data set of 2368 objects. Because OC1 is randomized, we randomly chose 80% of the WF1 data as the training set and reserved 20% as a test set. We then built a decision tree with OC1 on the training set and measured its accuracy on the test set. We repeated this procedure with different random training/test partitions ten times, and chose the best (in terms of overall accuracy on its test set) of the 10 trees as the tree to use for the WF2 data.

In all experiments, we report three accuracy figures: overall, stars, and cosmic rays. Overall accuracy is just the percentage of correct classifications over the whole test set, where "correct" means the decision tree agreed with the class label provided with the

Figure 6.1: A portion of a 900 second CCD WF1 image of M81 taken through a yellow filter (F555W), containing stars, cosmic rays, and other sources of CCD and sky noise.

Data	Accuracy (%)			Tree Size
	Overall	Stars	Cosmic Rays	
WF1	81.1	83.3	77.1	21.6
WF2	74.1	87.0	52.1	9

Table 6.1: OC1 accuracies: Using 9 raw pixel intensities. No blends

input data. Accuracy on stars (and respectively cosmic rays) is the percentage of the time that a prediction of "star" was correct.

6.1.2 Iteration 1: Using raw data

Any classification method uses a set of *features* to characterize each object; obviously, the features should be tailored to the task at hand. The first approach we took to extracting features has the appeal of maximum simplicity, and very liitle data massaging effort: simply give the classifier the raw data. A 3×3 array of pixels centered on a faint star contains most of the information about the star. Consequently, we extracted a 3×3 "postage stamp" centered on each star and each CR, and used the nine intensity values as our only features. Our results using the nine raw pixel values as the only features are presented in Table 6.1. These numbers obviously are not satisfactory, because not only are the accuracies low on WF1, but the trees built on WF1 do not generalize well on the WF2 data.

6.1.3 Iteration 2: An appropriate feature set

After the preliminary trials with the raw pixel intensities, we determined that we could improve our results substantially by using additional features relevant to the problem. The astronomers know in great detail how aberration and diffraction determine the HST intensity distribution from point light sources (i.e., stars), the so-called point spread function or PSF. If we fit a PSF to the stars and CRs, the parameters of the fit should be very different for the two classes. We used parameters from the PSF fit plus other knowledge about the differences between stars and CRs to define 11 additional features. This gave us a total of 20 parameters, listed below. Computation of these features is based entirely on an object's location on the CCD and on the pixel intensities in a 3x3 grid centered on the object.

Feature set for Star-Cosmic Ray discrimination:

- 9 **raw pixel values** in a 3x3 grid
- **x-moment**: computed from first intensity moments
- **y-moment**: computed from first intensity moments
- **ellipticity**: computed from a formula combining second intensity moments

Data	Accuracy (%)			Tree Size
	Overall	Stars	Cosmic Rays	
WF1	91.8	87.9	94.1	7.5
WF2	91.7	87.6	95.1	5

Table 6.2: OC1 accuracies: Using 20 features. No blends

ratio: average of 4 pixels (above, below, left, and right of central pixel)
divided by the intensity in the central pixel.
r1: magnitude calculated for a radius of 1 pixel
r1.5: magnitude (radius=1.5pixels) / magnitude (radius=1.0 pixels)
r2: magnitude (radius=2.0pixels)/ magnitude (radius=1.0 pixels)
peak intensity: intensity of the central pixel (radius=0.55 pixels)
Computed as: total counts (in this radius) - area*(avg. sky)
p2: *peak intensity*/ magnitude (radius=2.0 pixels)
mean: mean value of pixels in 3x3 postage stamp
stddev: standard deviation of pixels in 3x3 postage stamp

Significantly better results than those with the raw pixel intensities were obtained when the feature set included the PSF parameters, as shown in Table 6.2. Note that the trees produced using WF1 data were still very good classifiers for the WF2 data.

To give an example of what a particular classifier looks like, one relatively small tree that OC1 produced on the WF1 data is shown in Fig. 6.2. This tree gave 93.2% accuracy on the WF1 training set. The same tree applied to the WF2 data gave 91.9% overall accuracy.

In Figure 6.2, x_{20} is the standard deviation of the pixels in the 3x3 map, and x_{16} is the ratio of the magnitude at a radius of 2 pixels to the magnitude at 1 pixel. Because OC1 found a tree that used only two features, the partitioning induced by this tree can be viewed using those two features as the coordinate axes. As many trees produced by OC1 used these two features at the top levels, it might be a good idea to run OC1 using only these two features to describe the examples, an idea which we pursue in Section 6.1.5.

6.1.4 Iteration 3: Removing noise

Although 91% represents a very respectable accuracy, we wanted to improve this figure. With this goal in mind, we re-examined the CCD WF1 data in an attempt to locate other sources of error. One source of noise are the "hot" pixels, which occur where the "dark current" is large. The dark current appears when long exposures are taken with no light

Figure 6.2: Small, accurate tree produced by OC1-AP.

incident on the CCD.

The hot pixels occur in all of the images in our database (10 images that are within 1 pixel of each other). We combined all 10 images so that we would have better signal-to-noise, making it easier to see the hot pixels. We used a routine called DoPhot[43] which is fairly accurate at finding cosmic rays. Since the hot pixels are usually single pixel events resembling cosmic rays, all the objects DoPhot identified as cosmic rays in the combined image are presumably hot pixels. (Note that cosmic rays do not appear in the combined image.) We found between 100 and 225 hot pixels this way, depending on the CCD examined, but not all of them were bright enough to be detected.

After finding the hot pixels, we removed them from the WF1 and WF2 data sets, and re-ran our experiments to measure accuracy. For WF1, removing the hot pixels reduced the data set from 2259 objects to 2211, and for WF2 the number of objects dropped from 2368 to 2282. The accuracy on both data sets improved substantially when hot pixels were removed, as shown in Table 6.3. As in the first experiments, the WF1 accuracies were estimated using 10 five-fold cross-validation experiments. The WF2 accuracies were produced by using a single tree, created from WF1 data only, to classify the entire WF2 data set.

6.1.5 Iteration 4: Reducing the feature set

The performance of OC1 and many other machine learning methods can suffer when a data set is characterized by a large number of features (Section 2.5.1). Because OC1 searches an

[43] Thanks to Abhijit Saha of STScI for providing this software.

Data	Accuracy (%)			Tree Size
	Overall	Stars	Cosmic Rays	
WF1	94.0	91.8	95.3	8.4
WF2	95.1	95.1	95.0	4

Table 6.3: OC1 accuracies: 20 features, no blends, no hotpixels

Data	Accuracy (%)			Tree Size
	Overall	Stars	Cosmic Rays	
WF1	96.6	96.6	96.6	2
WF2	95.4	95.9	94.9	2

Table 6.4: OC1 accuracies: 2 features, no blends, no hot pixels

exponentially large space of hyperplanes, reducing the number of attributes dramatically reduces the size of the search space, improves the efficiency of OC1's search, and results in better oblique trees. We observed earlier that many of the trees used as few as two features: attributes 16 and 20. We therefore ran a final experiment in which we used the same data sets, but this time using just those two attributes. Our best results came from this final run using fewer features. These results appear in Table 6.4. As before, the results on WF2 were produced by training on the entire WF1 data set, and then testing on WF2. These results point out the importance of selecting the right features, which in this case was done by using a decision tree method as the selection mechanism.

The tree produced by OC1 in this last experiment is displayed in Fig. 6.3. This figure shows the data from WF1 displayed in two dimensions, with the tree induced by OC1 superimposed. A small number of outliers have been omitted from the figure in order to improve the resolution for the rest of the data. [44] We have also not shown some objects in the middle of the dense clusters in the figure, to help the clarity.

This very small tree (just two internal nodes) is easy to understand and even easier to interpret once displayed graphically. After the data massaging described here, a realistic estimate of accuracy for OC1 on the WF2 data increased from just over 92% to over 95%. We believe that this may represent the limit of accuracy for any method using the data we have available. However, larger data sets, which provide a more complete picture of the range of possible star and cosmic ray images, might lead to additional increases.

[44] More precisely, 68 objects out of a total of 2259 objects on WF1 lie far outside the bounding box of this graph. Of these, all except six are correctly classified by the OC1 tree shown. 5 of the misclassified objects are cosmic rays, and lie to the left of the oblique line. The remaining misclassified object is a star, and lies on the right of the oblique line, just above the line $X20 = 5.25$.

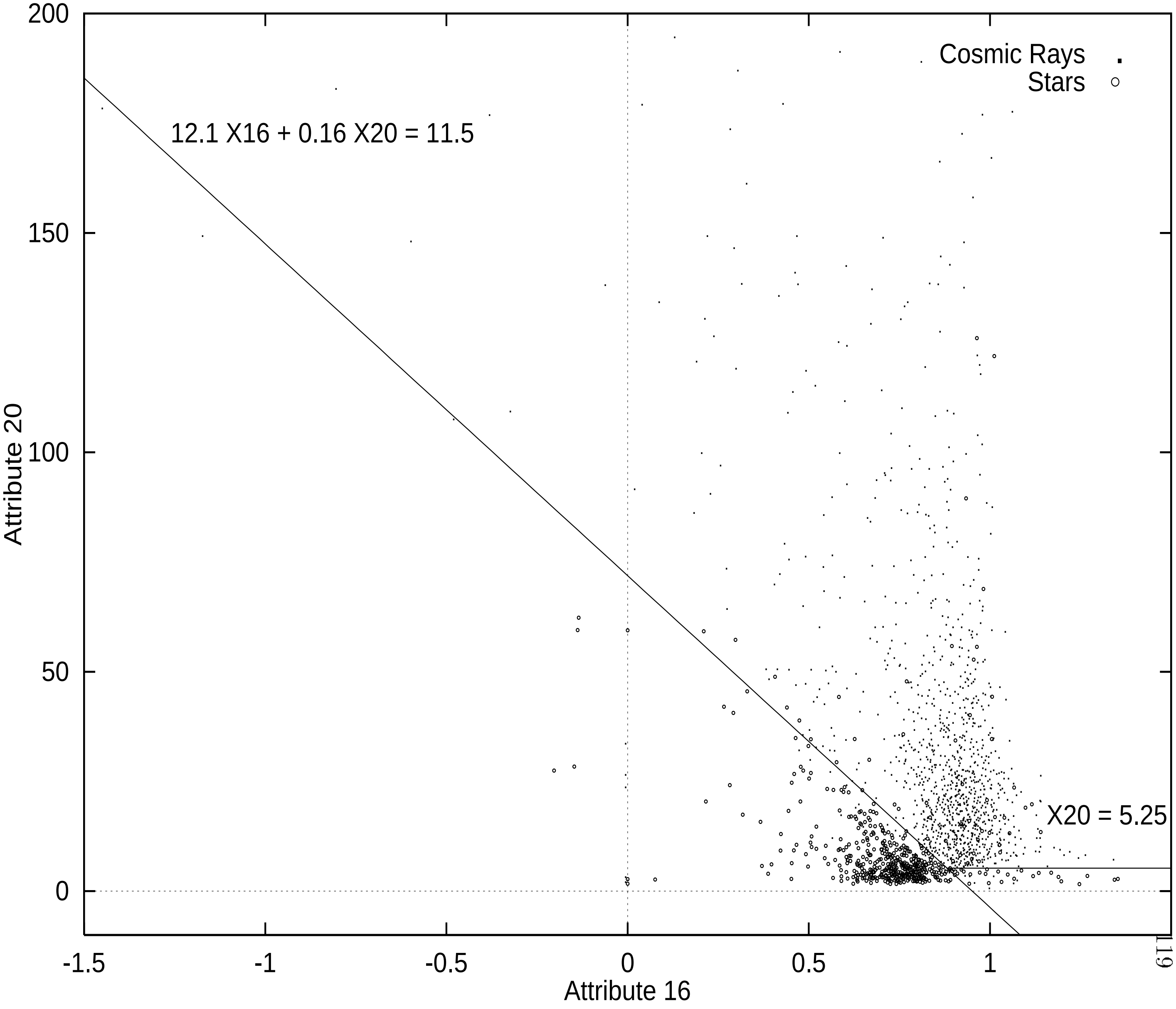

Figure 6.3: Data from WF1 displayed by Attributes 16 (magnitude at a radius of 2 pixels divided by magnitude at 1 pixel) and 20 (standard deviation of pixels in 3x3 postage stamp). The OC1 tree is superimposed on the data.

6.1.6 Using decision trees to confirm labelling

In a preliminary study using paired images, OC1 produced a decision tree that discovered errors made by a standard labelling procedure used as an HST analysis tool. For this study, we produced two new images from the pair. The first was a CR-clipped image produced by summing the original images with the STSDAS program Combine. In principle this image contains only stars. The second image was the difference of the original images, and contains only positive and negative CRs. By learning to recognize the difference in background for the stars in the summed image and the CRs in the difference image, OC1 produced a nearly perfect decision tree from a training set of 2,221 objects. (The only features used for this study were the nine intensity values from the 3x3 postage stamp centered on each object.) When we classified the remaining objects, 11 objects found in the combined image were classified as CRs. Inspection of these showed that 6 objects were in fact CRs that had survived the Combine program because they appeared in both images; independent cosmic rays had struck the same pixels in both images, thus fooling the Combine software. In other words, our decision tree detected errors in the labelling resulting from using Combine. After correcting the labelling in our catalogs, OC1 only mis-labelled (at most) four objects: one star and 3 CRs, for an overall accuracy of 99.8%.

6.1.7 The aberration-corrected images

All the work described so far was done before the First Servicing and Repair Mission of the HST during the December 1993. The images taken after the repair have very different characteristics than the ones taken before. Thus it was important to extend the above work to the aberration-corrected images.

The cosmic ray signature is essentially the same in the new camera as that seen in the old devices. However CR events impact scientific imaging differently with the new camera in two ways.

- The CCDs in the new camera have thickness around 10 micrometers, compared to the 8 micrometers for the old CCDs. This leads to a higher total number of electrons being deposited on the CCDs per event. In addition, the new camera has a significantly lower readnoise. Due to these “improvements”, a larger number of faint (low-amplitude) events are detected than before, leading to a larger number of pixels contaminated by CRs.
- As the new images are aberration-corrected, they have more light concentrated on a smaller number of pixels. So, small stars look more like cosmic rays in the new images.

In spite of the above difficulties, the distinctive signature of the point spread function (the sharp core surrounded by the first ring in the diffraction pattern) is more prominent in the new images, contributing to our confidence that our classification methods may be successful on the corrected imagery also.

Data	Accuracy (%)			Tree Size
	Overall	Stars	Cosmic Rays	
WF1	95.0	93.9	95.6	8
WF2	95.1	95.3	94.9	7
WF1 (new)	97.6	87.4	99.7	7

Table 6.5: OC1-AP accuracies: 2 features

Experimental design and results

We used four images (two separate pairs) of the M100 galaxy for classification. These two pairs are offset from each other by slightly more than one pixel. The first two images (set A) contain far more hotpixels than the second pair (Set B). In our previous set of experiments (Section 6.1.2), we combined two exposures to remove cosmic ray events. However, as the number of pixels contaminated by CRs per exposure have increased, there are higher chances that the same pixels are contaminated in both exposures. (See Appendix A for details.) We experimented with combining more than two images to identify the cosmic rays and to reduce contamination in star catalogs.

The star images proved to be a bit more difficult. The sky background varied widely in 3 of the 4 CCDs making accurate detection very difficult. Also the bulge of the galaxy falls in two CCDs. We tried different techniques (Appendix A) in order to find the one that gave the best star catalog on the the different CCDs.

We then ran the following experiment. We built decision trees on the data from (set A) CCD1 of the old camera using an axis-parallel version of OC1 and estimated their accuracies on (set A) CCD1 of the new camera. We used only features 16 and 20, which proved to be very good discriminants for the old images (Section 6.1.5). Table 6.5 shows the results of OC1-AP on the old WF1 and WF2, and on the new WF1. Note that trees were always built using the old WF1 data. Use of oblique hyperplanes did not improve results significantly over the axis-parallel ones. Similar results were obtained for data from the set B images, which contained fewer hotpixels.

CCD4 required the use of slightly different techniques than those described above, because it contained very bright as well as very dark sky regions. Appendix A gives details of how we extracted objects from CCD4. A 5-fold cross validation on the CCD4 data using OC1-AP and all 20 parameters gave the following accuracies: 92.4% overall, 95.1% for the cosmic rays and 86.9% for the stars. A significant problem for the CCD4 data set is that of blended objects. To deal with this problem, we defined a blend as a star and a cosmic ray that fall within two pixels of each other. There were approximately 290 blended objects on CCD4, which when deleted improved the accuracies as follows: **93.3%** overall, 96.5% for cosmic rays and 86.6% for stars.

Our analyses of the aberration-corrected data (by hand) indicates that the wide variations in the sky background are the major source of the difficulty of this data as opposed to the earlier one. For example, CCD3 contains part of the central bulge of the galaxy. We tried to eliminate most of the objects in that region from the data set, because it was very difficult to get accurate catalogs there. After we deleted spurious objects from the catalogs, the accuracies improved to about 94% overall, 93.5% on stars and 95.7% for the cosmic rays.

Although the M100 images obtained by the new camera appear quite difficult to classify, we found that once sources of noise are eliminated from the catalogs, OC1 is able to classify objects quite accurately, especially when the sky background does not vary too much across the CCD.

6.2 Star/galaxy classification for Sloan Digital Sky Survey

The Sloan Digital Sky Survey (SDSS), a large-scale digital survey of the northern galactic cap, is currently underway. The aim of the survey is to produce a detailed photometric map of half of the northern sky at about the 23rd magnitude. The detected objects would contain about 10^8 galaxies, similar number of stars and a million quasar candidates. From these, about one million bright galaxies and 100,000 bright quasars will be selected for which high-resolution spectra will be obtained. This project is being executed under the auspices of the Astrophysical Research Consortium by researchers at the University of Chicago, Fermi National Accelerator Laboratory, the Institute for Advanced Study, Johns Hopkins University, the National Observatory of Japan, and Princeton University. The survey work is expected to take five years, with a one year test period, and the total budget is $29.3M. The database that will result from SDSS will be enormous: a processed pixel map of the whole region will occupy about 8.2 tera bytes. The extracted spectra will occupy another 50 Gigabytes.

Because of the large amount of incoming data from the survey, efficient and robust data processing is a necessary component of the survey. An important part of SDSS's data processing pipeline is a classification module. Stars, galaxies and quasars in the photometric images need to be separated from each other, and from the various types of noise. We were able to build small decision tree classifiers for discriminating very accurately between star and galaxy images, down to the detection limit of the survey.

A set of simulated images and object catalogs are being made available to the SDSS researchers. The procedures for generating the simulated data are quite elaborate, and some details can be found in [114]. Considerable effort is being expended in making the simulated data as close to reality as possible, in terms of the noise levels, magnitudes, colors, atmospheric effects, distributions of objects etc. All the results reported in this section are on the simulated SDSS data. (The real survey data is not available yet. The first set will be released at the end of the second year of the survey, probably in 1997 or 1998.) We were able to get very high accuracies on this data, almost down to the detection limit of the survey.

6.2.1 The task

There are three basic categories of simulated data sets [114]: catalogs, images and tapes. A catalog is a list of objects, each of which has a position, magnitude, colors, shape parameters and so forth. An image represents a CCD frame, or a subsection of a frame. A tape (which might be a physical tape or a disk file) is a series of images, in the form of a data stream that looks as if it is coming from the telescope. Every image or tape has a corresponding catalog, so that one can run image or tape data through a pipeline module, and then compare the output to the "real" catalog of objects in the simulated universe. In the experiments reported below, we started with the positions of objects given in the catalogs, extracted from the images features we thought were relevant to classification (independently of the features listed in the catalogs), built decision trees using our features, and estimated accuracy against the "ground truth" (i.e., true class labels) provided by the catalogs.

The SDSS data is in five color bands, g, i, r, u and z. In each color, there are 20 frames. We used only eight of the 20 frames for our experiments. Each frame consists of 2048x1500 pixels. Each of the 8 frames we used contained approximately 1000 objects, together summing up to 9056 objects. The objects were classified into stars and galaxies. (Quasars have not been included in the simulated data yet.) [45] The SDSS images can be at five levels of increasingly more noise. All the data we used in the following experiments has noise level 5. This includes both local and global noise, encompassing such effects as readout noise, photon noise, ghosts, satellite tracks, airplanes , sky background fluctuation, atmospheric refraction, time delay of different color bands due to camera geometry and band-dependent positional shifts of objects (toward zenith) due to atmospheric differential refraction.

Throughout this section, we only report accuracies obtained with OC1-AP. We did try other methods such as OC1, CART, C4.5 and nearest neighbor, but the performances were not very different from that of OC1-AP. All the accuracy values given in this section are the averages of ten 10-fold cross validation experiments, each experiment using a different random seed for data partitioning.

6.2.2 Iteration 1: A "borrowed" feature set

For our first set of experiments, we tried to classify stars and galaxies using the same feature set that we found useful for cosmic ray identification (Section 6.1.2). Recall that there were a total of 20 features out of which 9 were raw pixel intensities. We did not use raw pixel values for star/galaxy classification, so there were 11 features. As the image features could be measured in any one of 5 color bands, this gave us a total of 55 features. We included some additional features that we thought might be helpful for differentiating between stars and galaxies — we included 3 new features per color band and 4 magnitude differences

[45] The SDSS data divides galaxies into elliptical galaxies, four types of spiral galaxies and irregular galaxies. For the experiments described in this section, we combined all of these into a single class *galaxy*. We may attempt subclassification of galaxies in the future.

across color bands. The total set of 74 features is listed below. The features that were not used for star-cosmic ray discrimination are shown in bold face.

Features for Iteration 1:

In color bands g,i,r,u and z:

x, y moments, ellipticity, ratio, r1, r1.5, r2

peak intensity, p2, mean, stddev

RMAG: magnitude

PEAK: intensity of the central pixel

FWHM: full width at half maximum: #pixels containing half the light.

magnitude differences **u-g, g-i, g-r** and **g-z**

Out of the 9056 objects in 8 frames, there were several very faint objects in the data. These had indefinite values for many features, because the object detection routines had difficulty in locating the faint objects and in measuring the features. For example, several objects had indefinite values for *RMAG* in one or more color bands, and several had *peak intensity* values ≤ 0. (Note that *peak intensity* is the intensity of the brightest pixel, so has to be positive.) There were also objects very close to the border of the image, whose features can not be accurately measured by any detection routine. The problem with near-border objects was easy to rectify: we used only the objects that were more than 20 pixels away from all four borders.

The problem of faint objects was much more difficult to solve. In fact, a main obstacle in solving the SDSS star-galaxy classification problem was to find a way of reliably quantifying magnitude of an object. It should be intuitively clear why a good measure of magnitude (brightness) is helpful: human experts classify brighter objects better, so one would expect automated classifiers to do the same. One can also divide objects into brightness ranges, and build separate classifiers for different brightness ranges. (The latter approach was taken in [367].)

At first, we tried several heuristic, empirically derived rules to filter out the faint objects. The following is an example of the kind of rules we considered:

> **Rule:** Retain only the objects that have a *peak intensity* value ≥ 20 in at least one color band. From these, remove objects that have an indefinite *RMAG* value in the u-band.

An application of the above rule resulted in retaining only 1754 objects out of the total 9056. Let us call this subset of presumably bright objects *Bright-R*. The accuracies obtained by OC1-AP on *Bright-R* are given in Table 6.6. Note that Table 6.6 has a new column, Coverage, which specifies what portion of the total data is used in the current experiment. In the case of *Bright-R*, the coverage is 1754 out of 9056, which is 19.4%.

Data	Coverage (%)	Accuracy (%)		
		Overall	Stars	Galaxies
Bright-R	19.4	91.5	86.7	94.7

Table 6.6: OC1-AP accuracies: 74 features

The accuracies in Table 6.6 are clearly not satisfactory. For instance, on *bright* stars, astronomers expect close to 100% accuracy, whereas the above numbers are in the higher 80s. A possible, and likely reason for this is that the objects in *Bright-R* are not really bright. The parameter *peak intensity*, which we used to identify the bright objects, measured the amount of light incident on the central pixel in a 3 × 3 pixel array. This measure is unreliable, as the objects can be off-center and the object centers vary from one color band to the other.

Moreover, none of our features measured the size of an object. In star/cosmic ray classification, a measure of size was not crucial because both the faint stars and cosmic rays are usually small. However, size can be an important discriminator between stars and galaxies, when combined with a peak intensity estimate. For instance, a star and a galaxy both having the same peak intensity values may be easily distinguished from the fact that the galaxy is larger than the star.

In addition, as machine learning methods are adversely affected by irrelevant or redundant features (Section 2.5.1), it may be beneficial to reduce the feature set from its current size of 75 by removing features that are clearly not useful.

6.2.3 Iteration 2: Refining the feature set

To address the problems outlined at the end of the last section, we added a new feature *peaksig*, which measures the intensity of the brightest pixel in the 3×3 pixel array. This is a more reliable brightness measure than *peak intensity*, which assumes that the central pixel is the brightest. In addition, we added two features, *radius* and *numpix*, which quantify size. We removed some features that were redundant/irrelevant. These included x and y *moments*, *RMAG, PEAK* and magnitude differences. We identified irrelevant features using standard feature subset selection methods (Section 2.5.1). We also studied several decision trees built in the previous experiments to identify, and remove, features that were never used. The modified feature set consisted of 61 features, listed below. σ denotes the sky background. New features are shown in bold face.

Features for Iteration 2:

In color bands g,i,r,u and z:

radius: average weighted distance between the object border to center.

numpix: #connected pixels above $1.5 * \sigma$, in a region of max. size 41×41.

Cutoff	Coverage (%)	Accuracy (%)		
		Overall	Stars	Galaxies
0	99.8	90.2	60.1	98.1
2.5	70.4	89.7	66.8	97.4
5	26.7	90.2	88.2	92.1
7.5	19.7	92.1	92.4	91.8
10	16.2	92.2	93.8	89.2

Table 6.7: Accuracy of OC1-AP. The larger the *peaksig* cutoff, the brighter the set of objects. 61 features.

ellipticity, ratio, r1, r1.5, r2, peak intensity, p2, mean, stddev
peaksig: intensity of the brightest pixel in a 3x3 box around the object
FWHM in the r-band

With this set of features, we used OC1-AP to construct decision trees on the data from 8 frames (9056 objects). At different cutoff thresholds for *peaksig*, we estimated the accuracy using ten 10-fold cross validation experiments with OC1-AP. The results are shown in Table 6.7. A cutoff threshold of k means that only objects that have a *peaksig* value more than k in at least one color band are retained in the data set. The format of Table 6.7 is similar to that of Table 6.6.

As seen from Table 6.7, *peaksig* is a reasonably good feature to quantify brightness, because the classification performance increased monotonically with brightness cutoffs. However, the classifier is guessing at the fainter end that all objects are galaxies, which is the more prevalent class. As the cutoff threshold increases, the proportion of stars in the training data increase, balancing the accuracies. The best results in Table 6.7 are better than the best results in iteration 1. At cutoff= 7.5, the coverage is about the same as that in Table 6.6 but the accuracies for stars and galaxies are better balanced.

6.2.4 Iteration 3: A multi-stage classifier

Assuming that *peaksig* is a good feature to quantify brightness, it may be beneficial to build separate classifiers for different brightness ranges. We divided the data into 3 subsets and built a different classifier for each subset.

- The **bright** objects have a *peaksig* value $\geq$ 15 in at least one color band. For these, we used the set of 61 attributes described in Section 6.2.3 and ran ten 10-fold cross validation experiments using OC1-AP.

- The **detectable** objects have a *peaksig* $\geq$ 5 in at least one color band, but have *peaksig* $\leq$ 15 in all color bands. We extracted a new set of features for classifying these objects, as described below.
- The **very faint** objects do not have a *peaksig* $\geq$ 5 in any color band. 73% of the objects in fields 1 through 8 are very faint. As a majority of these objects (88.8%) are galaxies, we extracted empirical rules that cover as many galaxies as possible. A concise and effective rule we found was the following.

Rule: if *peaksig* $<$ 1 in the g-band, then object is a galaxy.

Detectable Objects: Most parameters used so far were computed using a 3x3 postage stamp image. However, as dim objects can occupy larger areas, features need to be computed using the entire image of the object. We found the boundary of an object by starting at its center and spreading out in all directions until the intensity $\leq k * \sigma$ (σ is avg. sky brightness. We chose $k = 1.7$ and $k = 3.5$ on the recommendation of the astronomers). All pixels inside the boundary were used to compute the features. This method of computing features is clearly more powerful than the postage stamp method, and produced good parameters for the bright and very faint objects also (Section 6.2.5). The 51 features used in the classification of detectable objects are listed below. As before, new features are shown in bold face. We ran ten 10-fold cross validation experiments using OC1-AP and these 51 features.

Features for Detectable objects:
In color bands g,i,r and z:
for $k = 1.7$ and $k = 3.5$
npix: #pixels above $k * \sigma$
s-parm: "Gaussian-ness" of the core, for pixels above $k * \sigma$
mean: average transmission for pixels above $k * \sigma$
stddev: std. dev. of transmission for pixels above $k * \sigma$
r2, peak intensity
r3: magnitude (radius=3.0) / magnitude(radius=1.0)
r4: magnitude (radius=4.0)
magnitude: Equal to $-2.5 * \log_{10} flux$
sharpness: (Roughly) equal to $objectwidth^2 - PSFwidth^2$
chisquare: Diff. between the object and the PSF. Should be high for galaxies.

Table 6.8 summarizes the results of the multiple classifiers described above. These numbers are better than the accuracies we got with one-stage classifiers so far, because the

Data	Coverage (%)	Accuracy (%)		
		Overall	Stars	Galaxies
Bright	12.6	92.7	95.1	85.8
Detectable	14.1	89.3	75.4	93.6
Very Faint	9.9	93.9	0	100
Total	36.6	91.3	85.5	94.6

Table 6.8: Multiple classifiers based on brightness cutoffs.

3-stage classifier has a much larger coverage. However, these numbers were also not very satisfactory to the astronomers.

Though our multi-stage classifier was not very successful, we stumbled upon two very useful things in trying to come up with a feature set for the "Detectable" objects.

1. The method for computing feature values from the whole image of an object (which was used for the "Detectable" objects above) is clearly more powerful than the method we have been using, which computes features from a 3X3 postage stamp.
2. The features *magnitude, sharpness and chisquare* appear very powerful. *Magnitude* has an almost linear relation with the catalog magnitude of the objects in the simulated SDSS, as shown in Fig. 6.4. So it provides a better basis for brightness cutoffs than either *peak intensity* or *peaksig*. *Sharpness* estimates the intrinsic angular size of the measured object outside the atmosphere. It should have values close to zero for single stars, large positive values for blended doubles and partially resolved galaxies, and large negative values for cosmic rays and blemishes. The *chisquare* parameter should have near zero values for stars and high values for galaxies.

With the help of three new features and the whole-object method of extracting features, we built star/galaxy classifiers that achieved very high accuracies upto the detection limit of the survey. We will describe these experiments next.

6.2.5 The final iteration

At the end of the previous iteration, we found that the features *magnitude, sharpness and chisquare* have a variety of useful characteristics. We had also empirically observed that the feature *FWHM* was used in the highest levels of many trees for the "Bright" objects (Section 6.2.4). Drawing from these two pieces of evidence, we formed our final feature set.

Final Feature set

In color band r:

magnitude

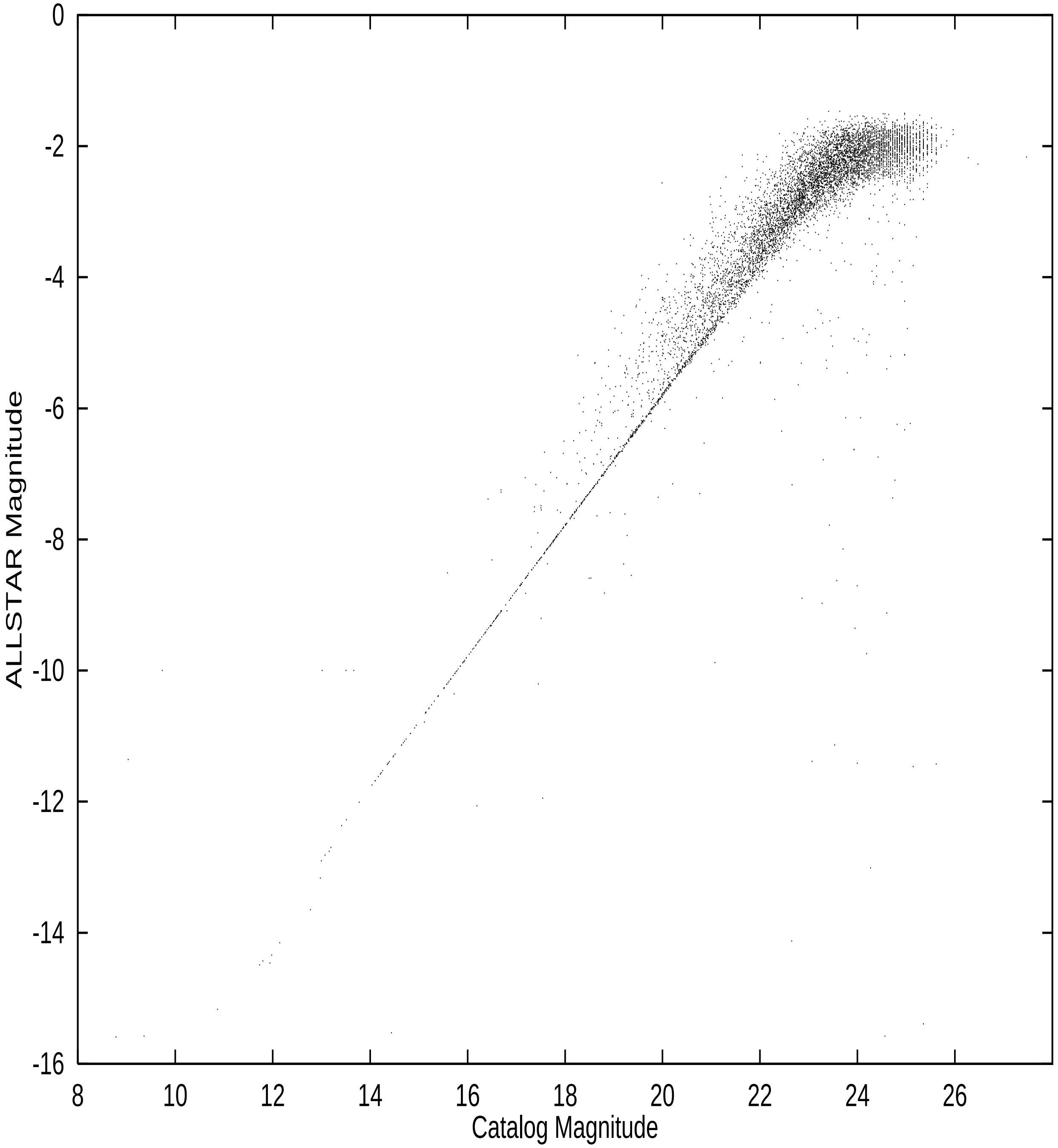

Figure 6.4: ALLSTAR magnitude varies almost linearly with the catalog magnitude. Hence it can be used to divide objects into brightness ranges reliably. This figure plots catalog versus ALLSTAR magnitudes for all objects (stars and galaxies) in fields 1–8

Magnitude Cutoff	Coverage (%)	Accuracy (%)		
		Overall	Stars	Galaxies
-2	85.4	89.8	64.2	98.0
-2.5	56.4	89.4	73.9	96.3
-3	39.7	91.5	85.0	95.4
-3.5	29.3	94.6	95.2	94.1
-4	22.4	96.6	97.4	95.7
-4.5	16.9	98.7	98.7	98.7
-5	12.6	99.1	98.9	99.6
-5.5	9.5	99.1	99.0	99.3
-6	7.5	98.9	98.8	98.9

Table 6.9: Accuracies with OC1-AP. Objects get brighter at lower cutoffs. 4 features.

sharpness
chisquare
FWHM

We used OC1-AP to build decision trees using these 4 features, and estimated accuracy using the average of ten 10-fold cross validation experiments, at different *magnitude* cutoffs. At a cutoff of k, only objects whose *magnitude* is $\leq k$ are retained. (Note that *magnitude* is a negative number, and the smaller the magnitude, the brighter the objects.) Out of the 9056 objects in frames 1 through 8, *magnitude* could be measured for 7319 objects. The other objects were too faint to be detected by the ALLSTAR program, which we used to measure features.

Very bright objects "saturate" the CCD pixels, so it is difficult to tell the difference between two very bright objects. There were many bright stars that were saturated in the r-band. When a star saturates the CCD pixels, it tends to have high values for *FWHM*, which makes it more similar to galaxies. We discovered that most of the stars that are saturated in the r-band are not saturated in the u-band. So, we used the parameters computed in the u-band for all objects whose catalog magnitude was greater than 13.6.

Table 6.9 shows the results of this experiment. At the faintest end, the classifier guesses that everything is a galaxy, giving poor accuracies on stars. But this situation improves rapidly. The accuracies on both the stars and galaxies increase significantly when the *magnitude* cutoff is raised. The accuracies for both stars and galaxies are $> 94\%$ at *magnitude* cutoff -3.5. ALLSTAR magnitude -3.5 is equivalent to catalog magnitude 22.35 (see Fig. 6.4), which is close to the detection limit of the SDSS survey. (Recall that the detection limit on the photometric survey in SDSS is at about the 23rd magnitude.) One main purpose of the photometric survey in SDSS is to identify the bright objects, for which high-resolution spectral analysis can be performed. SDSS aims to produce high resolution spectra for galaxies brighter than catalog magnitude 18.3. Even at a *magnitude* cutoff of -4.5, which is equivalent to catalog magnitude 21.35 and considerably fainter than the limit for spectroscopic survey, our classifier gets 98.7% accuracy on stars and galaxies.

It is interesting to note that the accuracies on both stars and galaxies go *down* slightly at the brightest ends. The reason for this is that there are fewer objects at the bright end. We observed that the same 5 stars and one galaxy were being misclassified at magnitude cutoffs -5, -5.5 and -6. We are currently looking into why these six objects are difficult to classify.

6.2.6 Confidence estimation

A common feature of all the experimental results we presented on SDSS data is that predictive accuracy decreases as objects become fainter. Though this is similar to the fact that the classification performance of humans deteriorates as the objects become fainter, there is a crucial difference. Humans typically know that they don't know — astronomers do not attempt to classify very faint objects, and even if they do, their *confidence* in the classification will be very little. It is desirable to be able to compute some estimate of confidence for pipelined systems like the Simulated Digital Sky Survey — if one classifier can accurately identify objects on which it is not confident, other, more elaborate, types of processing could be used on these objects.

A way of quantifying confidence in decision tree classification is by outputting class probabilities. Most decision tree methods used so far in this thesis output crisp decisions such as "this object is a star" or "this object is a galaxy". Instead, if the decision tree outputs "this object is a star with probability 0.6 and a galaxy with probability 0.4", then one identify objects for which the classifier is very certain, say, $probability \geq 0.95$. Some techniques to augment decision tree methods to output class probabilities have been explored in the literature (Section 2.5.6). We experimented with one such technique [62] on the SDSS data.

The most obvious way of determining class probabilities is to compute them directly from the counts of training examples at the leaf nodes. That is, if an object O that is being classified ends up at a leaf node which has 100 objects (90 stars + 10 galaxies) from the training set, then the probability of O being a star may be output as 0.9. This estimate is often not be reliable because (1) the number of training examples at individual leaf nodes is typically quite small, not allowing reliable probability estimates, and (2) the distribution at the training data at a leaf node may be particular to the choice of the training and pruning partitions used.

We improved the naive probability estimates at the leaf nodes using two techniques, *smoothing* and *averaging.* Smoothing is the process of combining the probability estimates at the leaf nodes with those at nodes higher up in the tree, whereas averaging is done across trees, combining the probability estimates of several trees built on the same data.

We started by dividing the data into a training set and a test set. Our training set consisted of all the objects from frames 1 through 4, and the test set had all objects from frames 5 through 8. (So far, we are doing cross validation experiments using the entire data — data from frames 1 through 8. An alternate way of getting reliable estimate of the accuracy on unseen data is to divide the data into training and test sets, as we had done in Section 6.1.) We first built ten trees on the training data. The only difference between the trees was the way the training data was partitioned into training and pruning sets. [46] In each tree, we smoothed the probabilities at each leaf by combining them along the paths from the root to the leaf, giving different weights to different ancestor nodes. (We do not discuss the smoothing algorithm in detail here. The interested reader is referred to [62].) Once the probability estimates are smoothed in each tree, we classify the test instances using the ten trees. Each tree outputs two probabilities for each test instance: the confidence of the object being a star and the confidence of it being a galaxy. We average these numbers across the ten trees, to get our confidence measures for each test instance.

Fig. 6.5 and Fig. 6.6 show the confidence measures outputted by the above method, as a function of ALLSTAR *magnitude*, for stars and galaxies respectively. There is one line (ruler) for each object, and the length of the ruler is 1.0, the total probability. In the figure for stars (Fig. 6.5), the length of the ruler above $Y = 0$ is the confidence that the object is a star, and the length below $Y = 0$ is the confidence that the object is a galaxy. In the figure for galaxies (Fig. 6.6), the length of the ruler above $Y = 0$ is the confidence that the object is a galaxy, and the length below $Y = 0$ is the confidence that the object is a star. So, a long bar above $Y = 0$ indicates that the object is correctly classified with confidence; a long bar below $Y = 0$ indicates that the object is misclassified with confidence, and a bar that is half-way through $Y = 0$ indicates that the classifier is not very sure of its classification. Note that the objects become brighter as we move towards the Y-axis.
Figures 6.5 and 6.6 offer several interesting insights.

- There are much fewer galaxies than stars at the brighter end. More bright galaxies are being misclassified than bright stars, however the bright stars are more confidently being misclassified. For example, if we decide to not take into account any object for which the confidence in both categories is less than 90%, we will not misclassify any galaxies (we will just drop them as being “unsure”), but we will misclassify both bright and faint stars.
- All the stars below magnitude of about -3.0 are confidently (80–90% confidence) classified as galaxies. However, all galaxies at this magnitude are still being confidently ($> 90\%$) correctly classified. These facts are in accordance with our earlier observation that, at very faint magnitudes, the classifiers are just guessing that everything is a galaxy.

[46] Note that OC1 being a randomized method produces a different tree each time it is executed. However, as we used the axis-parallel version, OC1-AP, for all our experiments in this section, we used generated different trees by randomly setting aside 30% of the training data each time for pruning.

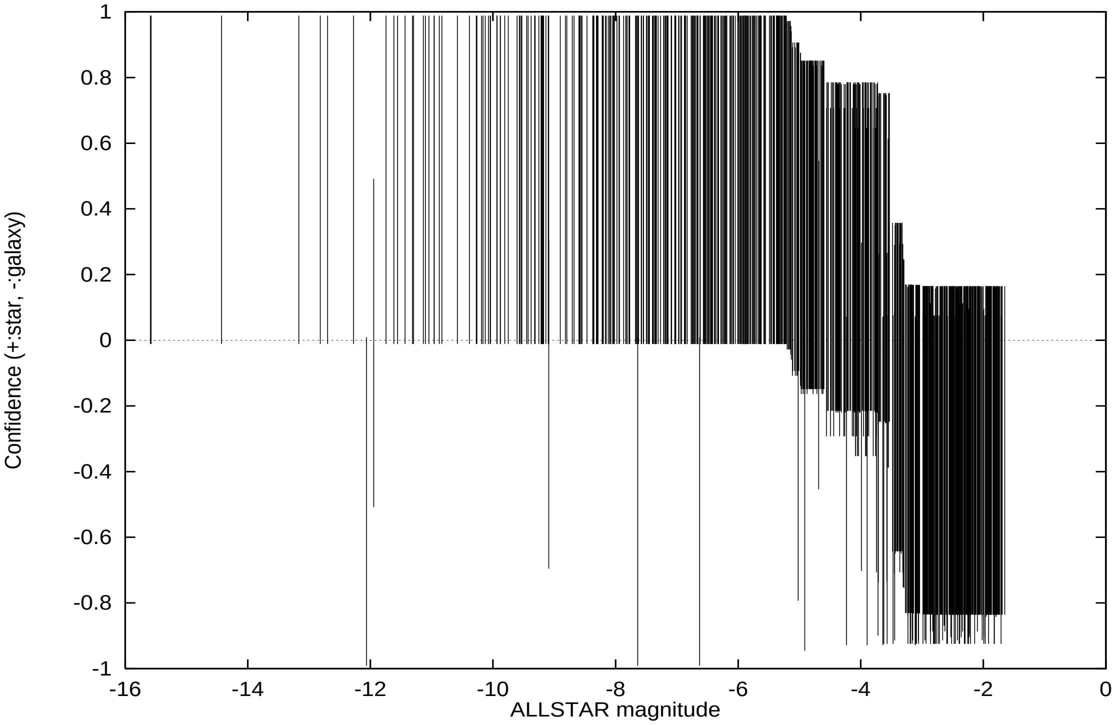

Figure 6.5: Confidence in classifying stars in fields 5–8, as a function of magnitude. The decision trees are trained using objects in fields 1–4.

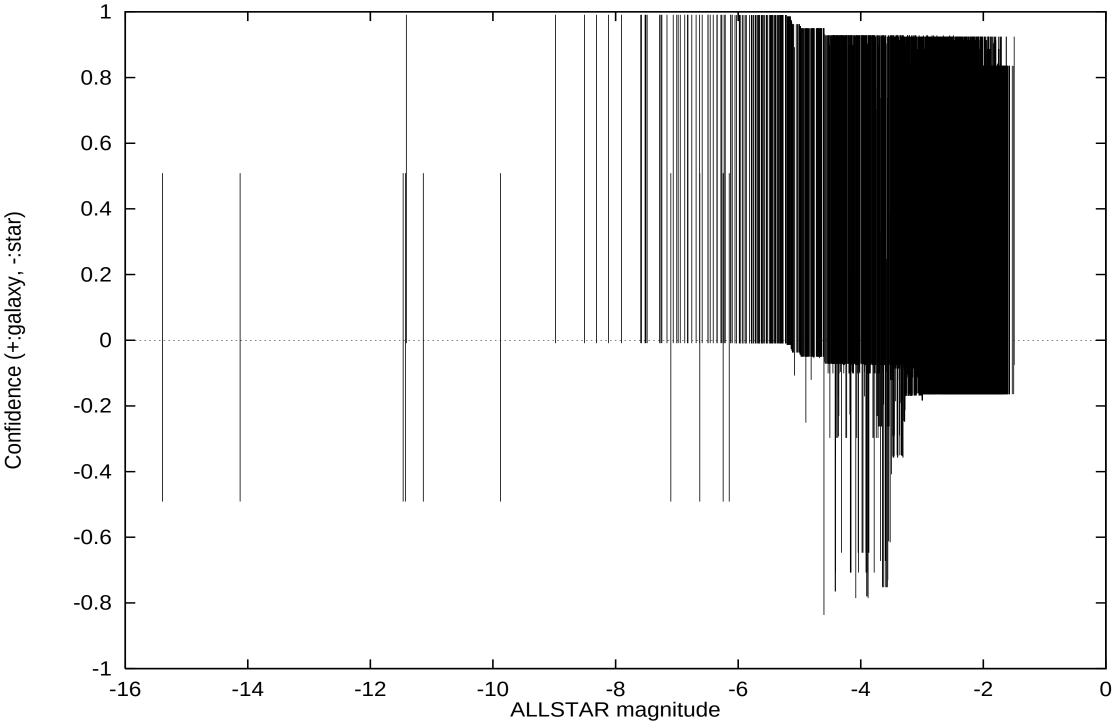

Figure 6.6: Confidence in classifying galaxies in fields 5–8, as a function of magnitude. The decision trees are trained using objects in fields 1–4.

In both Figure 6.5 and 6.6, there are a lot of objects in the magnitude range -3 to -4.5, for which the classifiers are "wavering" — the confidences are 70/30 or 60/40 for these objects. We found that some of the bright galaxies and stars are misclassified by the classifier because the fainter objects in the training set confuse it. By excluding the fainter stars and galaxies from the training set, we may be able to avoid this confusion, at the expense of losing some accuracy at the faint end. This is one of the directions we would like to explore. Another is to use the error in feature measurements to compute the confidence. The program ALLSTAR, which computed the features *magnitude, sharpness and chisquare*, outputs a measure of error in each of these features. We are currently trying to effectively use these error measures for building reliable classifiers.

Chapter 7

Domain-independent data massaging: Statistical Preprocessing

The previous chapter argued that (1) data massaging is a crucial step in building classifiers from real-world data, and (2) most data massaging requires domain knowledge. In the current chapter, we argue that it is posible to (partially) automate some data massaging tasks. We argue that, irrespective of the specific problem under consideration, decision tree methods can benefit by the use of "structure" information in numeric domains. We propose statistical preprocessing as a means to furnish decision tree methods with structure information.

The criterion used to measure the goodness of a split (*goodness measure* or *feature evaluation criterion*) is clearly an important factor in tree construction. Most existing goodness measures (Section 2.3.1) may be inadequate for some numeric domains for the following reasons:

- A majority of the existing goodness measures are either information theory based or class distance based. Both these kinds of measures compute the goodness of a split based solely on the discrete counts of each class in each partition. Other factors which may be relevant, such as distribution of the objects in the attribute space, are not taken into account at all.
- Several decision tree methods, especially in the machine learning literature, were originally proposed for symbolic domains. The "adoption" of these methods into numeric domains may be less than ideal, as numeric domains have their own peculiarities [485, 245, 486, 481].

In this chapter, we present a framework for augmenting decision tree induction so it can take advantage of patterns in numeric attribute spaces that would otherwise be ignored. We suggest a way preprocess (*massage*) data in numeric domains to extract some

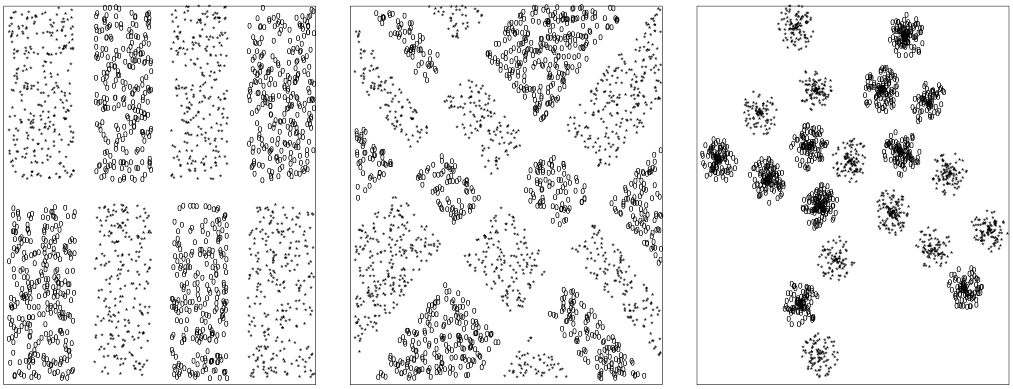

Figure 7.1: The CB, RCB and RGC data sets

"structure" information, which can in turn be used by any tree induction program. As a case study, we demonstrate empirically that clustering, when used as a preprocessing step, can improve the quality of decision trees induced.

Scatterd attempts exist in literature which qualify as examples of domain-independent data massaging. Nearest Neighbor classification is used to preprocess training data before using a neural network, in [362]. Flach [143] discusses inductive data engineering, an interactive process of restructuring a knowledge base by means of rule induction.

Section 7.1 presents some "simple" artificial data sets for which several common goodness measures fail to produce good trees. Section 7.2 suggests a framework in which data is processed by statistical methods prior to tree induction. As an example of the use of this framework, Section 7.3 uses Minimum Spanning Tree clustering as a preprocessing step with univariate and multivariate decision tree methods.

7.1 Three simple data sets

Fig. 7.1 displays three synthetic, no-noise 2-D data sets, each having 2000 objects belonging to two classes, 0 or $*$. The CB (checker board) data set can be described perfectly by an axis-parallel decision tree with 8 leaves. The RCB (rotated checkerboard) data can be described exactly by an oblique decision tree (Section 2.3.2) of 16 leaves. The RGC (randomly generated clusters) data consists of 20 circular clusters, and need not necessarily have a clear decision tree partitioning. But since the generation process produced clusters, trees that separate each cluster into a distinct region are clearly preferable.

Each of these artificial data sets has well-separated, dense, homogeneous regions of the attribute space that *call out* to be separated. Now consider typical decision trees induced on these data sets by existing tree induction methods. Figure 7.2 displays the decision trees generated for the CB data by C4.5 [398]. Figure 7.3 shows the trees generated by multivariate CART [44] and multivariate OC1 on the RCB data. Figure 7.4 displays the trees induced by C4.5 and multivariate CART on the RGC data.

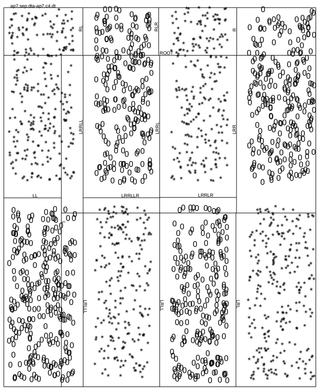

Figure 7.2: Tree induced on CB data by C4.5 using information gain

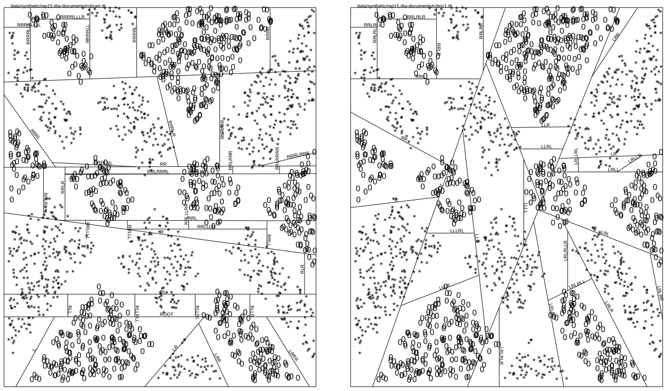

Figure 7.3: Trees induced on RCB data by multivariate CART and OC1, using Gini index

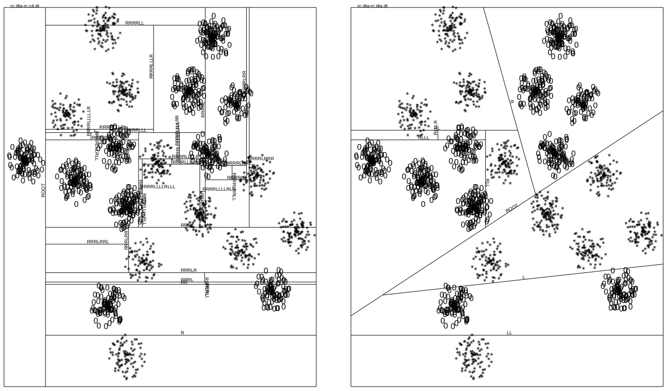

Figure 7.4: Trees induced on RGC data by C4.5 (information gain) and multivariate CART (Gini index)

These figures show that some otherwise successful tree induction methods have trouble in these apparently simple domains. The source of this difficulty is that the *only* information available to the goodness measures used is the distribution of object classes across the splits. However, building the ideal tree requires knowing that there are well-defined homogeneous clusters in the attribute space. Existing decision tree methods cannot use any such "structure" information. Some thought should convince the reader that this problem is specific to numeric attribute spaces. In nominal-valued domains, attribute similarity is used for generating the partitions and class similarity follows due to instance space proximity. But in numeric attribute spaces, this process is reversed. Class similarity guides the generation of decision regions and the proximity of instances is a side effect of the divide-and-conquer process.

One solution to this problem is to augment the definition of the goodness measures, to somehow take into account the "structure" of the examples in addition to the class distribution. Van de Merckt [486] used this approach to define a selection criterion that combines proximity with class entropy. Though this certainly is a step towards using structure, it leaves open some potential problems.

- de Merckt considers only one kind of structure information, namely clusters. It is not clear how to deal with other important kinds of structure information, for e.g., empty regions in attribute space, in his framework.

- de Merckt uses unsupervised clustering. This approach fails when each class is clearly multimodal, but the entire set of examples is not. (Consider, for example, variations of fig. 7.1 data sets with no "space" between clusters.)

- As de Merckt incorporates structure information into the definition of the goodness measure, this information needs to be calculated once for every split considered. This can be very expensive, especially for multivariate tree methods that consider large numbers of candidate splits.

An alternative way of incorporating structure, that overcomes the above problems, is presented in the next section.

7.2 A Framework

Given that structure information is important for constructing good decision trees in some numeric domains, an effective strategy is to find the structure using other (e.g., statistical) methods and to incorporate it into the information that the tree induction methods *can* use – namely, attributes or classes. This suggests the two-layered architecture in Fig. 7.5, in which tree induction is preceded by a data massaging step, namely, statistical preprocessing.

The training data is first fed into the statistical "structure extraction" module, which outputs information about patterns in the data. Patterns can include:

Figure 7.5: Statistical Preprocessing for Decision Tree Induction

- clusters,
- attributes irrelevant to inducing the classifier,
- instances that are useless/crucial in inducing the classifier,
- large empty regions in attribute space,
- sudden variations in the distribution of instances, etc.

The above information, referred to here as "structure" information for lack of a better phrase, is incorporated into the training data by modifying attributes and/or class labels. For example, if we know which subsets of instances form well-separated clusters, we can change the training set by marking each homogeneous cluster as a distinct class, to ensure that the decision tree separates out the clusters. If it is known that there exist large "voids" or empty spaces in the attribute space, we can generate "null" instances in these voids, so that the decision tree is forced not to overgeneralize.

An advantage of the above framework is the clear separation between the structure extraction and tree induction stages. Structure extraction methods of varying complexity can be used in conjunction with univariate, multivariate and/or incremental decision tree methods in this model. The complexity of the resulting system is only a sum of the complexities of the preprocessing and tree building stages, as opposed to a product as in [486].

7.3 A Concrete Example

This section illustrates that five decision tree methods (three axis-parallel and two oblique) benefit by using Euclidean minimum spanning tree (EMST) clustering as a preprocessing step, on the CB, RCB and RGC domains and two real-world data sets taken from the UCI machine learning repository [346].

7.3.1 Minimum Spanning Tree Clustering

A host of unsupervised clustering methods have been developed in the fields of psychology, statistics and machine learning. (See [127, 241] for a tutorial.) The choice of which clustering technique to use for a given data set is often very difficult. Most techniques require the user to define the number of clusters in advance, and those that do not, often require tuning of various parameters. Most clustering methods also require well-formed, convex clusters if they are to do a good job. In our experiments, we use a clustering method based on minimum spanning trees. The criteria we used to select this method from the many alternatives were (1) MST-clustering is intuitive and easy to implement, and (2) it has been researched extensively, and shown to work well on a variety of distributions [178, 513, 232].

A minimum spanning tree of a weighted graph G is the minimum-weight connected acyclic subgraph G' of G containing all vertices of G. Many robust and efficient algorithms are available to compute MSTs. An optimal algorithm runs in $O(E \log V)$ time and $O(E+V)$ space, where E and V are the number of edges and number of vertices in G, respectively. The variation of MSTs that we use is the Euclidean minimum spanning tree (EMST). An EMST of a set of points S is the minimum length acyclic graph spanning all points of S, where "length" of a graph is defined as the sum of the edge lengths. Efficient algorithms for MSTs do not do very well for EMSTs, because if we let $V = |S|$, then $E = O(V^2)$ for EMSTs. EMSTs in two dimensions can be constructed in an optimal $O(V \log V)$ time and $O(V)$ space by constructing the Delaunay triangulation of the points [386]. Unfortunately, this technique does not extend to higher-dimensional spaces, because the size of the Delaunay triangulation grows exponentially in the number of dimensions.

It is easy to see how EMSTs can be a basis for clustering. By definition, each edge in the EMST of a set of points S is the smallest edge connecting two partitions A and $S-A$ of S. Thus points in different clusters in S should intuitively be connected by longer edges in the EMST than points in the same cluster. So, cutting the largest edges in the EMST may separate the most prominent clusters of points. Cutting by edge length alone may not be sufficient to split the correct clusters, because the clusters thus obtained may not be of reasonable size and/or concentration. To obtain good clusters, one needs, in most cases, to impose some additional constraints on the edges to be cut.

Our method computes the MST of labeled examples and artificially sets the edge lengths between examples of different classes to be much *greater* than edges between examples of the same class. In this way, the MST connects all regions of like examples before making any connections between unlike examples. When the heaviest edges in an MST are cut, we obtain homogeneous clusters. (For more details of our clustering method, see [352].) In most of our experiments, the clustering algorithm did not do a perfect job, but our clusters were distinct enough that the tree induction methods did quite well. Obviously, the success of the clustering method will affect the quality of the decision tree induced.

7.3.2 Experiments

We have experimented with the synthetic data sets CB, RCB, RGC (Fig. 7.1), as well as two real-world data sets from UC Irvine machine learning repository [346]. The BUPA liver disorders data contains patients that have specific liver disorders. It has 345 instances, each described using 6 numeric attributes. The ION data [446] contains classifications of radar returns from the ionosphere. 351 observations, each with 34 continuous attributes, were classified as good or bad, depending on whether they were genuine or erroneous signals.

The decision tree induction programs used in our experiments were C4.5 [398], CART [44], and OC1 (Chapter 3). Both the univariate and multivariate versions of CART and OC1 were used, unless we knew the correct bias for a data set in advance. For example, on the CB data only univariate algorithms were considered. C4.5 used information gain (IG) as its goodness measure, and CART and OC1 used the twoing rule (TW). Additional experiments (not reported here due to space limitations) indicated that no other impurity measures did significantly better for the data sets used. The univariate CART and C4.5 implementations are part of the IND2.0 package developed by Wray Buntine. We implemented our own version of multivariate CART based on [44], with error complexity pruning using a separate test set. All the methods were run using default parameters.

All the classification accuracies and tree sizes (given as number of leaves) reported are averages of ten 5-fold cross validation experiments. A 5-fold cross validation consists of first dividing the training set randomly into 5 disjoint partitions of equal size. Then, for each partitions p, we built a tree using the rest of the data as training, and tested it on p. We report the error rate on the entire data set (number of examples misclassified/total number of examples), and the average size of the five trees built.

Table 7.1 summarizes the results of our experiments, giving classification accuracies and tree sizes (number of leaves) with and without the use of clusters as a pre-processing step. Each entry lists the mean and standard deviation of ten 5-fold cross-validation experiments.

The results on artificial data were unequivocal: the use of clustering allowed all the methods to get much smaller, more accurate trees. For the CB data, all three methods found the perfect tree every time, where without clustering they never found the right tree. These tree contains just eight leaf nodes, corresponding to the eight regions in Fig. 1. For the RCB data, only the oblique methods were used because we knew the tree required oblique splits. These methods showed a similarly dramatic improvement with clustering: in many cases they found the minimal tree with 16 nodes, and occasionally the tree was slightly larger. OC1 only had one non-essential leaf node on average. For the RGC data the results were improved but not as dramatic. For the two real data sets, the accuracies

	OC1		C4.5	CART	
Data Set	Univariate	Multivariate		Univariate	Multivariate
With No Preprocessing					
CB	99.7±1.08 13.6±3.63		99.7±0.27 23.6±2.70	99.9±0.13 16.0±3.50	
RCB		97.1±0.76 25.7±1.75			97.0±0.46 33.7±2.9
RGC	98.5±0.34 18.1±2.87	97.4±1.55 17.3±2.65	98.8±0.21 26±3.54	98.3±1.21 18±1.22	96.6±1.64 17.8±2.77
Bupa	62.1±3.66 4.2±2.28	67.5±2.13 3.4±2.63	67.3±6.69 37.2±7.19	62.9±3.15 8.4±2.5	62.7±2.06 3.6±1.35
ION	88.6±1.8 3.5±0.92	86.0±1.1 3.4±0.82	92.0±2.92 12.2±2.39	88.7±2.13 3.8±1.3	79.2±4.31 4.6±1.9
With Clustering as a preprocessing step					
CB	100±0.0 8±0.0		100±0.0 8±0.0	100±0.0 8±0.0	
RCB		99.6±0.44 17.2±1.2			99.5±0.7 21.0±2.53
RGC	99.4±0.2 17.7±1.39	99.6±0.29 17.2±0.45	99.6±0.16 17.1±1.89	99.5±0.21 17.6±1.33	99.4±0.22 17.3±1.2
Bupa	62.9±3.09 5.0±2.13	67.0±1.2 3.4±1.7	64.4±1.2 5.6±2.67	62.8±2.77 5.0±2.21	62.9±2.12 3.5±1.22
ION	89.2±1.43 3.7±0.63	87.6±1.39 3.5±1.3	91.4±1.95 7.8±2.16	89.3±1.4 3.7±0.67	79.1±1.6 4.8±1.55

Table 7.1: Effect of clustering as a data massaging step

remained roughly the same but in some cases the trees were much smaller. For example, the tree size for C4.5 dropped from 37.2 to 5.6 leaf nodes. These data sets do not contain such well-formed clusters as our synthetic data.

Of course, accuracy and tree size alone do not completely capture the quality of a decision tree in a continuous attribute space. For example, some trees displayed in Figures 7.2 and 7.4 are quite small and accurate, but impose a counterintuitive structure on the data. Alternative ways to quantify the goodness of decision trees would be useful in numerical domains [486]. In our experiments, after clustering information was provided to the tree induction programs, the trees induced were consistently identical to the original concept descriptions for the synthetic data. This indicates that using structure information helps induce better decision trees in some domains. However, in higher dimensional real-world domains, clusters obtained may be arbitrary, confusing the decision tree method. The results reported in this chapter are of a preliminary nature. We believe that the idea of statistical preprocessing in the context of decision trees has merit, and needs to be explored.

Chapter 8

Conclusions

In this dissertation, we explored a specific paradigm for data exploration and classification, namely, decision trees. We started by undertaking an extensive, multi-disciplinary survey of existing work on decision trees. We then studied in detail two algorithmic extensions to tree building methods, namely (1) using more general classes of splits at tree nodes, and (2) using less greedy search to determine the splits. Finally, we illustrated the importance of data massaging (domain-specific and domain-independent) for inducing good decision trees.

We recapitulate in Section 8.1 the main results in the dissertation, chapter by chapter. In Section 8.2, we point out some interesting directions for future research.

8.1 Summary

Chapter 2 presented a multi-disciplinary survey of work on automatically constructing decision trees from data. We gave pointers to work in fields such as pattern recognition, statistics, decision theory, machine learning, mathematical programming, neural networks, signal processing etc. We attempted to provide a self-contained, concise description of the directions which decision tree work has taken over the years. In addition to familiarizing the reader with the diversity and extent of work on decision trees, the larger goal of this survey is to help avoid some redundant, *ad hoc* effort in decision tree work, both from researchers and from system developers.

Chapter 3 described OC1, a new system for constructing oblique decision trees. We have shown experimentally that OC1 can produce good classifiers for a range of real-world and artificial domains. We have also shown how the use of randomization improves upon the original algorithm by Breiman *et al.*[44], without significantly increasing the computational cost of the algorithm. If a domain is best captured by a tree that uses oblique hyperplanes, it is desirable to have a system that can generate that tree. OC1 is the first such publicly available system.

Chapter 4 considered the implications of increasing the amount of search for inducing axis-parallel splits, beyond the commonly used greedy heuristic. Using experiments on a very large number of synthetic and real-world data sets, we demonstrated that limited lookahead search does not improve decision tree quality significantly. The only advantage of lookahead search in our experiments was to produce trees with slightly smaller worst-case classification cost than greedy search. However, lookahead produced *worse* trees than greedy search in a significant number of cases, in terms of prediction accuracy, tree size and depth (classification cost), exhibiting *pathology.* We observed instances of pathology both for real and synthetic data sets.

Chapter 5 is a continuation of the work in Chapter 4. After observing that limited lookahead search is not particularly beneficial for decision tree induction, we explored the question of whether this was because greedy induction itself produced trees so close to the optimal that any improvement was difficult. We built decision trees on thousands of synthetic data sets using CART [44] and C4.5 [398], and compared each one to the respective optimal tree. We found that, for a wide range of data characteristics, the greedy heuristic (along with pruning) produced decision trees whose expected classification cost was *very* close to the optimal.

We changed our focus after Chapter 5. Algorithmic extensions alone are not adequate for building “better” decision trees. Another important, if not crucial, step is to *massage*, or re-represent, the data in an appropriate fashion. We distinguished between domain-specific and domain-independent data massaging in this thesis.

We illustrated domain-specific data massaging in detail in Chapter 6, where we described applications of decision trees to two real-world classification problems. The first problem was to identify cosmic rays in Hubble Space Telescope images, and the second was to classify stars and galaxies in Sloan Digital Sky Survey images. For both these problems, we could not work with off-the-shelf data sets, as is common practice in machine learning research. In collaboration with researchers in astronomy, we went through several iterations of extracting useful data and features for classification. In both the astronomical classification problems we worked on, we were able to obtain very high accuracies using small decision tree classifiers.

Finally, in Chapter 7, we suggested a simple framework for domain-independent data massaging. Many existing tree induction methods can not take into account certain kinds of “structure” information in numeric attribute spaces. We suggested to use statistical methods to extract structure information from data, and represent that information in a form usable to tree induction. We gave a simple illustration of the effectiveness of this approach by using clustering as a preprocessing step for tree induction, on both real and artificial datasets. The results were uniformly excellent on the artificial data, which is not surprising given that these data sets were designed with this problem in mind. The results on real data were less convincing.

8.2 Research directions

One of the ideas we put forth in this thesis is the use of randomization for inducing oblique decision trees (Chapter 3). Randomization can also be beneficial for axis-parallel tree methods. Note that although greedy axis-parallel tree methods do find the optimal test (with respect to an impurity measure) for each node of a tree, they are necessarily suboptimal [174]. It will be interesting to explore the uses of randomization to build optimal or near-optimal axis-parallel trees. Randomized search techniques, such as genetic programming [264] and simulated annealing [55, 303] have already been used to improve axis-parallel decision trees. These methods search the space of all decision trees using random perturbations, additions and deletions of the splits. We believe that randomization is a powerful tool in the context of decision trees, and our experiments in Chapter 3 are just one example of how it might be exploited.

It will be interesting to design algorithms to efficiently find splits which are more general than oblique splits, say, a class of hypercurves. However, caution must be exercised in such an effort. A main advantage of decision trees in particular, and hierarchical methods in general, is that they divide the classification problem into a sequence of subproblems which are, in principle, simpler to solve than the original problem. Allowing more and more general splits implies that some advantages of the divide-and-conquer paradigm may be lost.

In addition, axis-parallel splits are simpler than oblique splits or hypercurves. It can be argued that a n node oblique tree in d dimensions is no smaller than a $n * d$ node axis parallel tree. In other words, it is desirable to use complicated splits only when the extra complexity of the split is commensurate with its contribution to tree quality. OC1 uses oblique splits only when their impurity is less than a user-defined constant k times the impurity of the best axis-parallel split; however, we do not know how to determine the best value for k. Finding a good compromise between the complexity and effectiveness of a split remains an area for further research.

Observing incidences of pathology (as we did in Chapter 4) is only the first step in several interesting research directions. Concept classes for which a particular goodness measure exhibits pathology can be studied, analytically or quantitatively, to determine *when* pathology might occur. Conversely, one can attempt to isolate characteristics of data which have bearing on when lookahead is likely to help. As we only studied lookahead in the context of two concept classes $\mathcal{C}$ and $\mathcal{C}_\mathcal{S}$ in Chapter 4, several other interesting concepts remain to be explored. In addition, we considered only one-level lookahead. One can attempt to evaluate the benefits of lookahead as a function of search depth. We feel that such a systematic evaluation is not only going to be computationally prohibitive, but also probably not very useful. Norton [365] presents experiments comparing one and two level lookahead for decision tree induction, on one particular data set. Another interesting question for further study is whether there exist effective goodness measures that guarantee no pathology.

Section 4.3.4 described a decision tree rebalancing method. We used rotation operations to locally adjust the structure of the tree, in order to reduce its classification cost. Note that, in addition to attempting to refine the structure of the tree, rotations provide an opportunity for recomputing splits at some tree nodes. This is because they change the portions of the training set associated with some nodes. We are in general interested in finding methods that can systematically refine the splits and structure of greedily induced trees.

Greedy decision tree induction is effective under many conditions (Chapter 5) and is quite efficient. For the situations in which greedy induction is not effective, we may need to augment it with appropriate preprocessing or postprocessing methods. We described, in Chapter 7 and Section 4.3.4, preliminary attempts at augmenting greedy decision tree induction. We suggested the use of statistical preprocessing to extract and use structure information in finding splits, and rebalancing greedily induced trees. Clearly more work needs to be done in these directions. The effectiveness of these approaches needs to be evaluated on more real-world data. Our experiments only explored incorporating structure information into class distributions. It will be interesting to incorporate structure into attributes instead. By doing this in conjunction with feature selection, it may be possible to identify what kinds of structure information are most useful for specific problems.

One premise of this work in Chapter 7 is that information about structure should be useful to learning algorithms. We do not yet know what other kinds of structural information besides clusters might be beneficial for decision tree induction, or to what extent. Besides clusters, descriptions of empty regions of attribute space might help build trees that do not overgeneralize (*underfit*). Sudden changes in the density of examples in the attribute space may tell us something about possible splits. One direction for future work is to identify and quantify the kinds of structure information useful in decision tree induction, and then explore the question of how to take advantage of this structure in building decision trees and other classifiers.

Last, but certainly not the least, it is always very educational and exciting to apply data exploration techniques like decision trees in new application areas. Each significant real-world classification problem has its own peculiarities, requirements and challenges. Algorithmic improvements often take place in the context of a particular application. We feel that continued application of existing techniques to new problems is a prerequisite for progress in data exploration technology.

A final word of caution. The hierarchical, recursive tree construction methodology itself is simple and intuitively appealing. However, the simplicity of the methodology should not lead a practitioner to take a slack attitude towards using decision trees. Experimental researchers should guard against the tendency to “try out” interesting ideas as and when they occur. There is a large body of previous work on decision trees, and significant extensions to this work are not easy to come up with.

Appendix A

Processing HST images

This chapter presents some details of the image processing techniques we used to detect objects and measure features from the Hubble Space Telescope images. It should be read in conjunction with Section 6.1 where we describe the process of building classifiers on the extracted data. Most of the material presented in this chapter is astronomy-related, and was originally prepared by astronomers Holland Ford, Richard White and Rupali Chandar, with whom the author had collaborated.

A.1 The aberrated images

Cosmic ray (CR) primaries and secondaries striking the CCD detectors in the HST's first Wide Field and Planetary Camera (WF/PC-1) created electron-hole pairs in the silicon that were detected along with the electron-hole pairs created by the photons striking the CCD.

The traditional way to filter cosmic rays is to work with twin exposures of the same region of the sky. Split exposures increase the readnoise in the images, in addition to needing the extra time required to prepare the spacecraft for an additional exposure. More precisely, the quadratic sum of the root mean square (rms) read noise and preflash noise in CCD WF1 was $\sim$ 20 electrons per pixel. The rms read noise in WFPC2 is $\sim$ 5.2 electrons per pixel. Whenever the exposure is split and then subsequently summed after CR removal, the read noise is increased by $\sqrt{2}$. This noise competes with the noise in the sky-background in short exposures or in exposures made through narrow band filters.

Our images for the cosmic ray identification project are taken from HST "Key Project" aimed at measuring the Hubble Constant, i.e., the rate at which the universe is expanding. The images are two 900-second WFC exposures through a yellow filter (F555W). Four CCDs (CCD WF1, WF2,WF3 and WF4) were exposed simultaneously when a picture was taken by WF/PC1.

We combined the two exposures using the task "Combine" in the Space Telescope Institute's software package STSDAS, with one-sided, three sigma cutoff. We next used the program DAOFIND to catalog the (x, y) positions of the stars in the image. [47] The mean background of the combined image varied from 27 to 33 counts in WF1, with an average of 31 counts. The standard deviation varied from 2.1 to 2.8, with an average of 2.55. Similar means and standard deviations were found for the other three CCDs. The threshold used for object detection by DAOFIND was 9 counts above the background, which along with the 2.55 standard deviation of the background gives a 3.5σ detection limit. By subtracting the combined image from each of the individual images, we obtained two images which contain

[47] The expected number of corresponding pixels with CR hits in both images is (roughly) $1574^2/800^2 \approx 4$. These can not be found by the above procedure of combining two images.

only CRs. We then used DAOFIND on each of the CR images to catalog the positions of the CRs.

As an external check on the completeness of this CR catalogue, we found the surface density of the CRs and compared it to the values recently determined in [505]. The Final Orbital/Science Verification Report by the WF/PC-1 Investigation Definition Team lists zeropoint offsets for the M81 field for WF2 [214]. Consequently, we used the same CCD for our CR surface density determination. We defined a cosmic ray to be a single pixel in the CR image with a threshold of $4.0 \times \sigma_{local}$ in ADU flux. A fairly high threshold was used because the noise in the two input images was heavily correlated, leading to an artificially low rms noise in the difference (CR) image. The (x, y) catalog of these CR positions on WF2 was used as input to the IRAF task DAOPHOT.PHOT, which performed 0.55 pixel aperture photometry. This essentially gives the counts in a single pixel, which is appropriate for cosmic rays. The V magnitude given by PHOT treats CRs as if they were real objects on the sky, and is defined as $C_v - 2.5 \log ADU$, where ADU is the ADU flux a CR would have generated in the image during the total exposure time. We histogramed the V magnitudes from both images.

Before plotting the histogram, we needed to find the correct zeropoint magnitude for our data, in order to use it as an offset. We followed a procedure similar to that outlined in [151]. We located the 5 Cepheids listed for CCD WF2 and did core fitting with an aperture of 2.5 pixels. In order to perform the aperture correction, we used a theoretical PSF generated by TinyTim for WF2. From this we found that $\sim 14.3\%$ of the light falls within a 2.5 pixel radius. We ignored color corrections as was done in [214], but we did perform aperture corrections. Comparing our F555W V magnitudes to the V magnitudes listed by the IDT, we confirmed the zeropoint magnitude of 23.0 to within 0.1 magnitudes (or 10%), and used this number as our offset. Figure A.1 plots the (log of the) number of pixels affected by cosmic rays as a function of V magnitude. The peak of our CR surface density occurs around magnitude 26, whereas it is around magnitude 27 in Windhorst et al.'s paper. This difference is most likely due to the much longer exposure times used in Windhorst et al.'s paper. We find that the differential CR "magnitude counts" closely follow an apparent power law, down to $V \simeq 27$ magnitude, with a slope $\gamma \simeq 0.57$ (when counted as $N(m) \propto m^{\gamma}$), compared to $\gamma \simeq 0.6$ in Windhorst et al.'s paper. Our observed surface density of CR hits is 3.96 pix/sec/CCD. This corresponds to ~ 3 events $cm^{-2}sec^{-1}$. (Note that we have defined a CR event to consist of one pixel which has signal above a certain threshold in the image containing only CRs.) In general, however, our results are consistent with theirs.

A.2 Hot Pixels

Hot pixels are caused due to the thermal generation (i.e., the CCD is not at 0° Kelvin) of electron-hole pairs at the interfaces between the silicon and oxide layers of the CCD causes the dark currents. At normal CCD temperature, the dark current is about 0.01 electrons/pixel/second for WF/PC. The hot pixels are caused by hits from high energy CRs which damage the lattice in the bulk silicon near the CCD channel stops. Because of

Figure A.1: HST/WFC Cosmic Ray “counts” (F555W)

the high electric fields near the channel stops, the lattice damage results in the creation of electron-hole pairs at the interface between the bulk silicon and the overlying silicon oxide layer.

Some of the damaged sites in the silicon lattice anneal whenever the temperature of the CCD is raised from the cold operating temperature of $-87°$ C to the warm decontamination temperature of $\sim 5°$ C; consequently, dark frames taken months before the most recent CCD temperature cycling are not a reliable guide for finding hot pixels. We used the following method to identify hot pixels. We combined all 10 images using STSDAS "combine" so that we would have better signal-to-noise, making it easier to see the hot pixels. Fig. A.2 shows a portion of the combined WF1 image (not the same portion as shown in Fig. 6.1). We ran DoPhot on the combined image using a threshold of 10 counts above the background. The mean sky varied from 33 to 42 counts with an average of 39, while the standard deviation varied from 1.6 to 2.5 with an average of 2.2, giving a detection level of $\sim 4.5\sigma$. We found between 100 and 225 hot pixels this way, depending on the CCD examined. However not all of these were sufficiently intense to be detected by DAOFIND in the image from which we made the catalogue.

The last step, in order to find the (x, y) values of the hot pixels and remove them from the images, was to match the (x, y) coordinates of the hot pixels given by DoPhot with our star catalogue using a radius of 1.5 pixels. (The reason that we had to do this matching is that DAOFIND and DoPhot give slightly different coordinates for detections, and we needed the exact coordinates of objects to delete.)

Figure A.2: A 900 sec WF1 image of M81 which has been combined to improve signal-to-noise. This picture shows a portion of the WF1 image.

A.3 The aberration-corrected images

WFPC II CCDs have a thickness around 10 micrometers, compared to the 8 micrometers for the WF/PC I CCDs, and a recombination length of 8–10 microns in the substrate. This leads to a higher total number of electrons being deposited on the CCDs per event. WFPC II also has a significantly lower readnoise than WF/PC I [48], so low amplitude events are detected leading to a larger apparent number of CR events. This leads to a larger number of pixels that are contaminated by CR in a WFPC II image, even though the underlying event rate is similar to that experienced by WF/PC I.

Stellar images are still undersampled, but with the correction of the spherical aberration, stars and cosmic rays look more alike and are more difficult to tell apart. Faint stellar images and low amplitude CRs are indistinguishable. Long observations are thus broken up into at least two exposures (CR-SPLIT) to ensure that CR events can be identified.

Approximately 3.7% of the pixels in WFPC2 CCDs are affected by cosmic rays in a 2000 sec. exposure. Since cosmic rays hit random locations on the CCDs, about 1000 pixels per chip may be hit in both the exposures to be combined, making CR identification for these pixels impossible. These affected pixels obviously can contaminate the data set.

The first step to classification was to create accurate catalogs of star and cosmic ray positions on the CCDs. We used the STSDAS.COMBINE routine in IRAF to combine each pair of images. This algorithm retains objects that are statistically above the noise in

[48] The quadratic sum of the root mean square (rms) read noise and preflash noise in CCD WF1 was $\sim$ 20 electrons per pixel. The rms read noise in WFPC2 is $\sim$ 5.2 electrons per pixel.

both images (stars in our case) and rejects objects that appear in only one image (cosmic rays). If the resulting image (containing stars, hotpixels, and CR events that occurred in the same place in both images) is subtracted from one of the original images, the result is an image containing only cosmic rays. We ran DAOFIND on this cosmic ray image, using a 4 sigma detection limit, to get a catalog of cosmic rays.

Set A images were taken at the higher instrument temperature of around -77 degrees Celsius. Set B images was taken with instrument temperature around -88 degrees Celsius, and were the first chronologically taken images at that lower temperature. Images taken later at the lower temperature contain more and more hotpixels since long term radiation from cosmic rays leads to an increase in the dark noise, primarily in the creation of hotpixels. The solution to this in HST has been to anneal the CCD's periodically to wipe out these hotpixels. All four of our images are 1800 second exposures taken at different times through a yellow filter (F555W).

Let the combination of the two images in set A (with lots of hot pixels) created to get rid of cosmic rays be called C1 and the combination of the images in set B (with fewer hotpixels) be called C2. We first created C1 and C2 for all the CCDs. We then used a technique which took advantage of the fact that C2 is shifted by about one pixel from C1. We shifted and combined C1 and C2 (using IRAF task PROTO.IMALIGN and STSDAS.COMBINE), to get rid of most of the hotpixels. We then subtracted the combined image from C1, and ran a detection routine to get a map of all of the hotpixels. Some stars leaked through to this image, and we deleted them by hand. This gave us a map of all the hotpixels in the first set of images, which were matched and deleted from the star catalog created by running DAOFIND on the combined image.

CCD4 required the use of slightly different techniques than those described above, because it contained very bright as well as very dark sky regions. The sky varied from approximately 30 counts to 60 counts depending on location in the chip. The standard deviation of the sky also varied by location. One of the different methods we used for CCD4 is the following: we took C1 and C2 for this CCD and aligned the images in the same way as discussed above. The shifted images were put through STSDAS.COMBINE again. However, instead of using this to catalog the hotpixels and then delete them from the data set, we used the combined image of all four images (sets A and B) to catalog the stars. The stars are a bit dimmer, but this technique has the advantage that not only does it eliminate the hotpixels, but it also gets rid of many of the cosmic ray events that occurred in the same location in both the images that were used to make C1 or C2. The chances that cosmic ray hits were sustained in all four CCD's in the same location is decreased significantly. We ran DAOFIND on this combined image. These coordinates were shifted when used on the images of set B. We ended up with 1751 stars for CCD 4.

Bibliography

[1] E.H.L. Aarts, P.J.M. Van Laarhoven, J.K.Lenstra, and N.L.J.Ulder. A computational study of local search algorithms for job shop scheduling. *ORSA Journal on Computing*, **6**(2):118–125, Spring 1994.

[2] J. Aczel and J. Daroczy. *On measures of information and their characterizations.* Academic Publishers, New York, 1975.

[3] David W. Aha and Richard L. Bankert. A comparitive evaluation of sequential feature selection algorithms. In AI&Statistics-95 [5], pages 1–7.

[4] *AI&Stats-93: Preliminary Papers of the Fourth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, 3rd–6th, January 1993. Society for AI and Statistics.

[5] *AI&Stats-95: Preliminary Papers of the Fifth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, 4–7th, January 1995. Society for AI and Statistics.

[6] Hussein Almuallim and Thomas G. Dietterich. Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, **69**:279–305, 1994.

[7] American Association for Artificial Intelligence. *AAAI-92: Proceedings of the Tenth National Conference on Artificial Intelligence*, San Jose, CA, 12–16th, July 1992. AAAI Press / The MIT Press.

[8] American Association for Artificial Intelligence. *AAAI-93: Proceedings of the Eleventh National Conference on Artificial Intelligence*, Washington, DC, 11–15th, July 1993. AAAI Press / The MIT Press.

[9] American Association for Artificial Intelligence. *AAAI-94: Proceedings of the Twelfth National Conference on Artificial Intelligence*, volume 1, Seattle, WA, 31st July - 4th August 1994. AAAI Press / The MIT Press.

[10] Peter Argentiero, Roland Chin, and Paul Beaudet. An automated approach to the design of decision tree classifiers. *IEEE Transactionson Pattern Analysis and Machine Intelligence*, **PAMI-4**(1):51–57, January 1982.

[11] Sunil Arya and David M. Mount. Asymptotically efficient randomized algorithm for nearest neighbor searching. Technical Report CS-TR-3011 or UMIACS-TR-92-135, Computer Science, University of Maryland, College Park, MD, December 1992.

[12] Les Atlas, Ronald Cole, Yeshwant Muthuswamy, Alan Lipman, Jerome Connor, Dong Park, Muhammed El-Sharkawi, and Robert J. Marks II. A performance comparison of trained multilayer perceptrons and trained classification trees. *Proceedings of the IEEE*, **78**(10):1614–1619, 1990.

[13] Peter Auer, Robert C. Holte, and Wolfgang Maass. Theory and applications of agnostic PAC-learning with small decision trees. In ML-95 [333], pages 21–29. Editor: Jeffrey Schlimmer.

[14] Haldun Aytug, Siddhartha Bhattacharya, Gary J. Koehler, and Jane L. Snowdon. A review of machine learning in scheduling. *IEEE Transactions on Engineering Management*, **41**(2):165–171, May 1994.

[15] A. Babic, E. Krusinska, and J.-E. Stromberg. Extraction of diagnostic rules using recursive partitioning systems: A comparison of two approaches. *Artificial Intelligence in Medicine*, 4(5):373–387, October 1992.

[16] J.L. Baer and B. Schwab. A comparison of tree-balancing algorithms. *Communications of the ACM*, **20**(5):322–330, 1977.

[17] L. Bahl, P.F.Brown, P.V. de Souza, and R. L. Mercer. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Accoustics, Speech and Signal Processing*, **37**(7):1001–1008, 1989.

[18] Eard Baker and A. K. Jain. On feature ordering in practice and some finite sample effects. In *Proceedings of the Third International Joint Conference on Pattern Recognition*, pages 45–49, San Diego, CA, 1976.

[19] F. A. Baker, David L. Verbyla, C. S. Hodges Jr., and E. W. Ross. Classification and regression tree analysis for assessing hazard of pine mortality caused by hetero basidion annosum. *Plant Disease*, **77**(2):136, February 1993.

[20] D.A. Belsley. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity.* Wiley & Sons, New York, 1980.

[21] W. A. Belson. Matching and prediction on the principle of biological classification. *Applied Statistics*, **8**:65–75, 1959.

[22] Moshe Ben-Bassat. Myopic policies in sequential classification. *IEEE Transactions on Computing*, **27**(2):170–174, February 1978.

[23] Moshe Ben-Bassat. Use of distance measures, information measures and error bounds on feature evaluation. In Krishnaiah and Kanal [265], pages 773–791.

[24] J. A. Benediktsson and P. H. Swain. Consensus theoretic classification methods. *IEEE Transactions on Systems, Man and Cybernetics*, **22**(4):688–704, 1992.

[25] K.P. Bennett and O.L. Mangasarian. Robust linear programming discrimination of two linearly inseparable sets. *Optimization Methods and Software*, **1**:23–34, 1992.

[26] K.P. Bennett and O.L. Mangasarian. Multicategory discrimination via linear programming. *Optimization Methods and Software*, **3**:29–39, 1994.

[27] K.P. Bennett and O.L. Mangasarian. Serial and parallel multicategory discrimination. *SIAM Journal on Optimization*, **4**(4), 1994.

[28] Kristin P. Bennett. Decision tree construction via linear programming. In *Proceedings of the 4th Midwest Artificial Intelligence and Cognitive Science Society Conference*, pages 97–101, 1992.

[29] Kristin P. Bennett. *Machine Learning via Mathematical Programming.* PhD thesis, Department of Computer Science, University of Wisconsin-Madison, 1993.

[30] Kristin P. Bennett. Global tree optimization: A non-greedy decision tree algorithm. In *Proceedings of Interface 94: The 26th Symposium on the Interface*, Research Triangle, North Carolina, 1994.

[31] John A. Bentrup and Sylvian R. Ray. An examination of inductive learning algorithms for the classification of sleep signals. Report. UIUCDCS-R-93-1792, Department of Computer Science, University of Illinois at Urbana-Champaign, 1304 Springfield Avenue, Urbana, Il 61801, February 1993.

[32] A. Blum and R. Rivest. Training a 3-node neural network is NP-complete. In *Proceedings of the 1988 Workshop on Computational Learning Theory*, pages 9–18, Boston, MA, 1988. Morgan Kaufmann.

[33] Marko Bohanec and Ivan Bratko. Trading accuracy for simplicity in decision trees. *Machine Learning*, **15**:223–250, 1994.

[34] David Bowser-Chao and Debra L. Dzialo. Comparison of the use of binary decision trees and neural networks in top quark detection. *Physical Review D: Particles and Fields*, **47**(5):1900, March 1993.

[35] D. Boyce, A. Farhi, and R. Weishedel. *Optimal Subset Selection.* Springer-Verlag, 1974.

[36] Anna Bramanti-Gregor and Henry W. Davis. The statistical learning of accurate heuristics. In IJCAI-93 [221], pages 1079–1085. Editor: Ruzena Bajcsy.

[37] Y. Brandman, A. Orlitsky, and J. Hennessy. A spectral lower bound technique for the size of decision trees and two-level AND/OR circuits. *IEEE Transactions on Computers*, **39**(2):282–286, February 1990.

[38] Breast cancer data. Available in the UCI ML Repository. Obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Data provided by Matjaz Zwitter and Milan Soklic.

[39] Y. Breibart and A. Reiter. A branch-and-bound algorithm to obtain an optimal evaluation tree for monotonic boolean functions. *Acta Informatica*, **4**:311–319, 1975.

[40] Leo Breiman. The π method for estimating multivariate functions from noisy data. *Technometrics*, **33**(2):125–143, 1991.

[41] Leo Breiman. Stacked regressions. Technical Report TR-367, Department of Statistics, University of California at Berkeley, 1992.

[42] Leo Breiman. Hinging hyperplanes for regression, classification and function approximation. *IEEE Transactions on Information Theory*, **39**(3):999–1013, May 1993.

[43] Leo Breiman. Bagging predictors. Technical report, Department of Statistics, University of California, Berkeley, CA, 1994.

[44] Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone. *Classification and Regression Trees.* Wadsworth International Group, 1984.

[45] Michael R. Brent, Sreerama K. Murthy, and Andrew Lundberg. Discovering morphemic suffixes: A case study in minimum description length induction. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, Ft. Laudersdale, FL, January 1995.

[46] Richard P. Brent. Fast training algorithms for multilayer neural nets. *IEEE Transactions on Neural Networks*, **2**(3):346–354, May 1991.

[47] Carla E. Brodley. *Recursive Automatic Algorithm Selection for Inductive Learning.* PhD thesis, University of Massachusetts, Amherst, MA, 1994.

[48] Carla E. Brodley and Paul E. Utgoff. Multivariate versus univariate decision trees. Technical Report COINS-CR-92-8, Dept. of Computer Science, University of Massachusetts, Amherst, MA, January 1992.

[49] Carla E. Brodley and Paul E. Utgoff. Multivariate decision trees. *Machine Learning*, **19**:45–77, 1995.

[50] Donald E. Brown, Vincent Corruble, and Clarence Louis Pittard. A comparison of decision tree classifiers with backpropagation neural networks for multimodal classification problems. *Pattern Recognition*, **26**(6):953–961, 1993.

[51] Donald E. Brown and Clarence Louis Pittard. Classification trees with optimal multivariate splits. In *Proceedings of the International Conference on Systems, Man and Cybernetics*, volume 3, pages 475–477, Le Touquet, France, 17–20th, October 1993. IEEE, New York.

[52] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computing*, **C-35**(8):677–691, August 1986.

[53] Randal E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. Technical Report CMU-CS-92-160, Carnegie Mellon University, School of Computer Science, Pittsburgh, PA 15213., July 1992. Accepted to ACM Computing Surveys.

[54] Nader H. Bshouty. Exact learning via monotone theory. In *Proceedings. 34th Annual Symposium on Foundations of Computer Science*, pages 302–311, New York, NY, 1993. IEEE.

[55] R.S. Bucy and R.S. Diesposti. Decision tree design by simulated annealing. *Mathematical Modieling and Numerical Analysis*, **27**(5):515–534, 1993. A RAIRO Journal.

[56] C. Bull, M. Chiogna, R. Franklin, and D. Spiegelhalter. Expert derived and automatically generated classification trees: an example from pediatric cardiology. In *Proceedings: Computers in Cardiology*, pages 217–220, Los Alamitos, CA, September 5th–8th 1993. IEEE Computer Society Press.

[57] M. E. Bullock, D. L. Wang, Fairchild S. R., and T. J. Patterson. Automated training of 3-D morphology algorithm for object recognition. *Proceedings of SPIE – The International Society for Optical Engineering*, **2234**:238–251, 1994. Issue title: Automatic Object Recognition IV.

[58] W. Buntine. Tree classification software. Technology 2002: The third national technology transfer conference and exposition, December 1992.

[59] W. Buntine and T. Niblett. A further comparison of splitting rules for decision-tree induction. *Machine Learning*, **8**:75–85, 1992.

[60] Wray Buntine. Classifiers: A theoretical and empirical study. In IJCAI-91 [220], pages 638–644. Editors: John Mylopoulos and Ray Reiter.

[61] Wray Buntine. *A theory of learning classification rules.* PhD thesis, University of Technology, Sydney, Australia, 1991.

[62] Wray Buntine. Learning classification trees. *Statistics and Computing*, **2**:63–73, 1992.

[63] Wray Buntine and Rich Caruana. Introduction to IND and recursive partitioning. Technical Report FIA-91-28, RIACS and NASA Ames Research Center, Moffett Field, CA, 1991.

[64] G. Burger and U. Jútting. Specimen classification in cytometry: An intercomparison of various means of decision making. In Gelsema and Kanal [163], pages 509–519.

[65] A. Buzo, A.H. Gray Jr., Robert M. Gray, and J.D. Markel. Speech coding based upon vector quantization. *IEEE Transactions on Accoustics, Speech and Signal Processing*, **28**:562–574, October 1980.

[66] Janice D. Callahan and Stephen W. Sorensen. Rule induction for group decisions with statistical data - an example. *Journal of the Operational Research Society*, **42**(3):227–234, March 1991.

[67] Jan M. Van Campenhout. Topics in measurement selection. In Krishnaiah and Kanal [265], pages 793–803.

[68] C. Carter and Jason Catlett. Assessing credit card applications using machine learning. *IEEE Expert*, **Fall**:71–79, 1987.

[69] Rich Caruana and Dayne Freitag. Greedy attribute selection. In ML-94 [331], pages 28–36. Editors: William W. Cohen and Haym Hirsh.

[70] Richard G. Casey and George Nagy. Decision tree design using a probabilistic model. *IEEE Transactions on Information Theory*, **IT-30**(1):93–99, January 1984.

[71] Jason Catlett. *Megainduction.* PhD thesis, Basser Department of Computer Science, University of Sydney, Australia, 1991.

[72] Jason Catlett. Tailoring rulesets to misclassification costs. In AI&Statistics-95 [5], pages 88–94.

[73] Gert Cauwenberghs. A fast stochastic error-descent algorithm for supervised learning and optimization. In *Proceedings of Neural Information Processing Systems*, 1992.

[74] G. Cestnik, I. Kononenko, and I. Bratko. Assistant 86: A knowledge acquisition tool for sophisticated users. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning.* Sigma Press, 1987.

[75] C. Chan and J. Bao. On the design of a tree classifier and its application to speech recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, **5**(5):677–692, December 1991.

[76] B. Chandrasekaran. From numbers to symbols to knowledge structures: Pattern Recognition and Artificial Intelligence perspectives on the classification task. In Gelsema and Kanal [163], pages 547–559.

[77] B. Chandrasekaran and A. K. Jain. Quantization complexity and independent measurements. *IEEE Transactions on Computers*, **C-23**(1):102–106, January 1974.

[78] C.-L. Chang. Finding prototypes for nearest neighbor classifiers. *IEEE Transactions on Computers*, **23**(11):1179–1184, November 1974.

[79] Hsi Chang and Sitharama Iyengar. Efficient algorithms to globally balance a binary search tree. *Communications of the ACM*, **27**(7):695–702, July 1984.

[80] A. R. Chaturvedi and D. L. Nazareth. Investigating the effectiveness of conditional classification: an application to manufacturing scheduling. *IEEE Transactions on Engineering Management*, **41**(2):183–193, May 1994.

[81] M. R. Chmielewski and J. W. Grzymala-Busse. Global discretization of continuous attributes as preprocessing for machine learning. In T. Y. Lin, editor, *RSSC-94: The Third International Workshop on Rough Sets and Soft Computing*, pages 294–301, San Jose, CA, November 1994. American Association of Artificial Intelligence, San Jose State University.

[82] Philip A. Chou. *Applications of Information Theory to Pattern Recognition and the Design of Decision Trees and Trellises.* PhD thesis, Stanford University, 1988.

[83] Philip A. Chou. Optimal partitioning for classification and regression trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**(4):340–354, April 1991.

[84] Philip A. Chou and Robert M. Gray. On decision trees for pattern recognition. In *Proceedings of the IEEE Symposium on Information Theory*, page 69, Ann Arbor, MI, 1986.

[85] Philip A. Chou, Tom Lookabaugh, and Robert M. Gray. Optimal pruning with applications to tree-structured source coding and modeling. *IEEE Transactions on Information Theory*, **35**(2):299–315, March 1989.

[86] Athene Chuk-Kwan Choy. *Decision Rules and Decision Tree Approach in Pattern Recognition Theory.* PhD thesis, University of Missouri, Columbia, MO, 1978.

[87] Krzysztof J. Cios and Ning Liu. A machine learning method for generation of a neural network architecture: A continuous ID3 algorithm. *IEEE Transactions on Neural Networks*, **3**(2):280–291, March 1992.

[88] I. Cleote and H. Theron. CID3: An extension of ID3 for attributes with ordered domains. *South African Computer Journal*, **4**:10–16, March 1991.

[89] Cleveland heart disease database. Available in the UCI ML Repository. Collected by Roberto Detrano, M.D., Ph.D., V.A. Medical center, Long Beach and Cleveland Clinic Foundation.

[90] J.R.B. Cockett and J.A. Herrera. Decision tree reduction. *Journal of the ACM*, **37**(4):815–842, October 1990.

[91] W.W. Cohen. Efficient pruning methods for separate-and-conquer rule learning systems. In IJCAI-93 [221], pages 988–994. Editor: Ruzena Bajcsy.

[92] Douglas Comer and Ravi Sethi. The complexity of trie index construction. *Journal of the ACM*, **24**(3):428–440, July 1977.

[93] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 1990.

[94] T.M. Cover and J.M. Van Campenhout. On the possible orderings in the measurement selection problems. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-7**(9), 1977.

[95] Louis Anthony Cox. Using causal knowledge to learn more useful decision rules from data. In AI&Statistics-95 [5], pages 151–160.

[96] Louis Anthony Cox and Yuping Qiu. Minimizing the expected costs of classifying patterns by sequential costly inspections. In AI&Statistics-93 [4].

[97] Louis Anthony Cox, Yuping Qiu, and Warren Kuehner. Heuristic least-cost computation of discrete classification functions with uncertain argument values. *Annals of Operations Research*, **21**(1):1–30, 1989.

[98] Stuart L. Crawford. *Resampling strategies for recursive partitioning classification using the CART algorithm*. PhD thesis, Stanford University, 1987.

[99] Stuart L. Crawford. Extensions to the CART algorithm. *International Journal of Man-Machine Studies*, **31**(2):197–217, August 1989.

[100] Stephen P. Curram and John Mingers. Neural networks, decision tree induction and discriminant analysis: An empirical comparison. *Journal of the Operational Research Society*, **45**(4):440–450, April 1994.

[101] K.T. Dago, R. Luthringer, R. Lengelle, G. Rinaudo, and J. P. Matcher. Statistical decision tree: A tool for studying pharmaco-EEG effects of CNS-active drugs. *Neuropsychobiology*, **29**(2):91–96, 1994.

[102] Florence D́Alché-Buc, Didier Zwierski, and Jean-Pierre Nadal. Trio learning: A new strategy for building hybrid neural trees. *International Journal of Neural Systems*, **5**(4):259–274, December 1994.

[103] S.K. Das and S. Bhambri. A decision tree approach for selecting between demand based, reorder and JIT/kanban methods for material procurement. *Production Planning and Control*, **5**(4):342, 1994.

[104] Belur V. Dasarathy, editor. *Nearest neighbor (NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, Los Alamitos, CA, 1991.

[105] Belur V. Dasarathy. Minimal consistent set (MCS) identification for optimal nearest neighbor systems design. *IEEE transactions on systems, man and cybernetics*, **24**(3):511–517, 1994.

[106] G. R. Dattatreya and Laveen N. Kanal. Decision trees in pattern recognition. In Kanal and Rosenfeld, editors, *Progress in Pattern Recognition*, volume 2, pages 189–239. Elsevier Science, 1985.

[107] G. R. Dattatreya and V. V. S. Sarma. Bayesian and decision tree approaches to pattern recognition including feature measurement costs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-3**(3):293–298, 1981.

[108] L. Devroye. Automatic pattern recognition : A study of the probability of error. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **10**(4):530–543, 1988.

[109] Thomas G. Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of Artificial Intelligence Research*, **2**:263–286, January 1995.

[110] Thomas G. Dietterich, Hermann Hild, and Ghulum Bakiri. A comparison of ID3 and backpropagation for english text-to-speech mapping. *Machine Learning*, **18**:51–80, 1995.

[111] Thomas G. Dietterich and Eun Bae Kong. Machine learning bias, statistical bias and statistical variance of decision tree algorithms. In ML-95 [333]. to appear.

[112] Thomas G. Dietterich and Ryszard S. Michalski. A comparitive view of selected methods for learning from examples. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning, an Artificial Intelligence Approach*, volume 1, pages 41–81. Morgan Kaufmann, San Mateo, CA, 1983.

[113] Justin Doak. An evaluation of search algorithms for feature selection. Technical report, Graduate Group in Computer Science, University of California at Davis; *and* Safeguards Systems Group, Los Alamos National Laboratory, January 1994.

[114] M. Doi, J. Gunn, and D. Weinberg. Simulated data for the SDSS: User's guide. Technical report, Princeton University, Princeton, NJ, 1994.

[115] B. A. Draper, Carla E. Brodley, and Paul E. Utgoff. Goal-directed classification using linear machine decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **16**(9):888, 1994.

[116] N. R. Draper and H. Smith. *Applied Regression Analysis.* Wiley, New York, 1966. 2nd edition in 1981.

[117] R. Duda and P. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[118] Gunter Dueck and Tobias Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. *Journal of computational physics*, **90**:161–175, 1990.

[119] Eades and Staples. On optimal trees. *Journal of Algorithms*, **2**(4):369–384, 1981.

[120] Bradley Efron. Estimating the error rate of a prediction rule: improvements on cross-validation. *Journal of American Statistical Association*, **78**(382):316–331, June 1983.

[121] A. Ehrenfeucht and David Haussler. Learning decision trees from random examples. *Information and Computation*, **82**:231–246, 1989.

[122] John F. Elder, IV. Heuristic search for model structure. In AI&Statistics-95 [5], pages 199–210.

[123] Tapio Elomaa. In defence of C4.5: Notes on learning one-level decision trees. In ML-94 [331], pages 62–69. Editors: William W. Cohen and Haym Hirsh.

[124] A. Ercil. Classification trees prove useful in nondestructive testing of spotweld quality. *Welding Journal*, **72**(9):59, September 1993. Issue Title: Special emphasis: Rebuilding America's roads, railways and bridges.

[125] Floriana Esposito, Donato Malerba, and Giovanni Semeraro. A further study of pruning methods in decision tree induction. In AI&Statistics-95 [5], pages 211–218.

[126] Bob Evans and Doug Fisher. Overcoming process delays with decision tree induction. *IEEE Expert*, pages 60–66, February 1994.

[127] Brian Everitt. *Cluster Analysis - 3rd Edition.* E. Arnold Press, London., 1993.

[128] Final orbital/science verification report: Wide field/planetary camera investigation definition team, February 1991. Principal Investigator: James A. Westphal.

[129] Judith A. Falconer, Bruce J. Naughton, Dorothy D. Dunlop, Elliot J. Roth, and Dale C. Strasser. Predicting stroke inpatient rehabilitation outcome using a classification tree approach. *Archives of Physical Medicine and Rehabilitation*, **75**(6):619, June 1994.

[130] A. Famili. Use of decision tree induction for process optimization and knowledge refinement of an industrial process. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing (AI EDAM)*, **8**(1):63–75, Winter 1994.

[131] R. M. Fano. *Transmission of Information.* MIT Press, Cambridge, MA, 1961.

[132] Usama M. Fayyad and Keki B. Irani. What should be minimized in a decision tree? In *AAAI-90: Proceedings of the National Conference on Artificial Intelligence*, volume 2, pages 749–754. American Association for Artificial Intelligence, 1990.

[133] Usama M. Fayyad and Keki B. Irani. The attribute specification problem in decision tree generation. In AAAI-92 [7], pages 104–110.

[134] Usama M. Fayyad and Keki B. Irani. On the handling of continuous-valued attributes in decision tree generation. *Machine Learning*, **8**(2):87–102, 1992.

[135] Usama M. Fayyad and Keki B. Irani. Multi-interval discretization of continuous valued attributes for classification learning. In IJCAI-93 [221], pages 1022–1027. Editor: Ruzena Bajcsy.

[136] Usama M. Fayyad, Nicholas Weir, and D. Djorgovski. SKICAT: A machine learning system for automated cataloging of large scale sky surveys. In ML-93 [330], pages 112–119. Editor: Paul E. Utgoff.

[137] Edward A. Feigenbaum. Expert systems in the 1980s. In A. Bond, editor, *State of the Art in Machine Intelligence*. Pergamon-Infotech, Maidenhead, 1981.

[138] C. Feng, A. Sutherland, R. King, S. Muggleton, and R. Henery. Comparison of machine learning classifiers to statistics and neural networks. In AI&Statistics-93 [4], pages 41–52.

[139] A. Fielding. Binary segmentation: the automatic interaction detector and related techniques for exploring data structure. In O'Muircheartaigh and Payne [370], pages 221–257.

[140] P. E. File, P. I. Dugard, and A. S. Houston. Evaluation of the use of induction in the develeopment of a medical expert system. *Computers and Biomedical Research*, **27**(5):383–395, October 1994.

[141] Douglas Fisher. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, **2**:130–172, 1987.

[142] Douglas Fisher and Kathleen McKusick. An empirical comparison of ID3 and back propagation. In IJCAI-89 [219]. Editor: N. S. Sridharan.

[143] P. A. Flach. Predicate invention in inductive data engineering. In P. B. Brazdil, editor, *ECML: European Conference on Machine Learning*, Berlin, Germany, 1993. Springer-Verlag. Conf. held in Vienna, Austria, 5-7 April 1993.

[144] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Computer Journal*, **6**(ISS.2):163–168, 1963.

[145] D. H. Foley. Considerations of sample and feature size. *IEEE Transactions on Information Theory*, **IT-18**:618–626, 1972.

[146] F. Forouraghi, L. W. Schmerr, and G. M. Prabhu. Induction of multivariate regression trees for design optimization. In AAAI-94 [9], pages 607–612.

[147] Iman Foroutan. *Feature Selection for Piecewise Linear Classifiers.* PhD thesis, University of California, Irvine, CA, 1985.

[148] Iman Foroutan and Jack Sklansky. Feature selection for automatic classification of non-Gaussian data. *IEEE Transactions on Systems, Man and Cybernetics*, **17**(2):187–198, March/April 1987.

[149] Richard S. Forsyth, David D. Clarke, and Richard L. Wright. Overfitting revisited: an information-theoretic approach to simplifying discrimination trees. *Journal of Experimental and Theoretical Artificial Intelligence*, **6**(3):289–302, July–September 1994.

[150] M. Frean. *Small Nets and Short Paths: Optimising neural computation.* PhD thesis, Centre for Cognitive Science, University of Edinburgh, 1990.

[151] W. L. Freedman, S. M. Hughes, B. F. Madore, J. R. Mould, M. G. Lee, P. Stetson, R. C. Kennicutt, A. Turner, L. Ferrarese, H. C. Ford, J. A. Graham, R. Hill, J. G. Hoessel, J. Huchra, and G. D. Illingworth. The Hubble Space Telescope extragalactic distance scale key project. I. The discovery of cepheids and a new distance to M81. *Astrophysical Journal*, **427**:628, 1994.

[152] Jerome H. Friedman. A recursive partitioning decision rule for nonparametric classifiers. *IEEE Transactions on Computers*, **C-26**:404–408, April 1977.

[153] Jerome H. Friedman. Flexible metric nearest neighbor classification. Technical report, Department of Statistics and Stanford Linear Accelerator Center, Stanford University, Stanford, CA 94305, November 1994.

[154] M. Fujita, H. Fujisawa, and Y. Matsunaga. Variable ordering algorithms for ordered binary decision diagrams and their evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, **12**(1):6–12, January 1993.

[155] Keinosuke Fukanaga. *Introduction to Statistical Pattern Recognition.* Academic Press, 1990.

[156] Keinosuke Fukanaga and R. A. Hayes. Effect of sample size in classifier design. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **11**:873–885, 1989.

[157] Truxton K. Fulton, Simon Kasif, and Steven Salzberg. An efficient algorithm for for finding multi-way splits for decision trees. In ML-95 [333]. to appear.

[158] G. M. Furnival. Regression by leaps and bounds. *Technometrics*, **16**(4):499–511, 1974.

[159] Michael R. Garey and Ronald L. Graham. Performance bounds on the splitting algorithm for binary testing. *Acta Informatica*, **3**(Fasc. 4):347–355, 1974.

[160] Michael R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the theory of NP-Completeness.* Freeman and Co., San Francisco, CA, 1979.

[161] S. B. Gelfand and C. S. Ravishankar. A tree-structured piecewise-linear adaptive filter. *IEEE Transactions on Information Theory*, **39**(6):1907–1922, November 1993.

[162] Saul B. Gelfand, C. S. Ravishankar, and Edward J. Delp. An iterative growing and pruning algorithm for classification tree design. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, **13**(2):163–174, February 1991.

[163] Edward S. Gelsema and Laveen N. Kanal, editors. *Pattern Recognition in Practice*, volume 2. Elsevier Science, Amsterdam, The Netherlands, 1986.

[164] Edzard S. Gelsema and Laveen S. Kanal, editors. *Pattern Recognition in Practice IV: Multiple paradigms, Comparative studies and hybrid systems*, volume 16 of *Machine Intelligence and Pattern Recognition. Series editors: Kanal, L. S. and Rozenfeld, A.* Elsevier, 1994.

[165] G. H. Gennari, Pat Langley, and Douglas Fisher. Models of incremental concept formation. *Artificial Intelligence*, **40**(1–3):11–62, September 1989.

[166] Ian P. Gent and Toby Walsh. An empirical analysis of search in GSAT. *Journal of Artificial Intelligence Research*, **1**:47–59, September 1993.

[167] Allen Gersho and Robert M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1991.

[168] Casimiro Giampaolo, Andrew T. Gray, Richard A. Olshen, and Sandor Szabo. Predicting chemically induced duodenal ulcer and adrenal necrosis with classification trees. *Proceedings of the National Academy of Sciences of the USA*, **88**(14):6298–6302, July 1991.

[169] W. J. Gibb, D. M. Auslander, and J. C. Griffin. Selection of myocardial electrogram features for use by implantable devices. *IEEE Transactions on Biomedical Engineering*, **40**(8):727–735, August 1993.

[170] M. W. Gillo. MAID: A Honeywell 600 program for an automatised survey analysis. *Behavioral Science*, **17**:251–252, 1972.

[171] Elizabeth A. Giplin, Richard A. Olshen, Kanu Chatterjee, John Kjekshus, Arthur J. Moss, Harmut Henning, Robert Engler, A. Robert Blacky, Howard Dittrich, and John Ross Jr. Predicting 1-year outcome following acute myocardial infarction. *Computers and biomedical research*, **23**(1):46–63, February 1990.

[172] Malcolm A. Gleser and Morris F. Collen. Towards automated medical decisions. *Computers and Biomedical Research*, **5**(2):180–189, April 1972.

[173] M. Golea and M. Marchand. A growth algorithm for neural network decision trees. *EuroPhysics Letters*, **12**(3):205–210, June 1990.

[174] Rodney M. Goodman and Padhraic J. Smyth. Decision tree design from a communication theory standpoint. *IEEE Transactions on Information Theory*, **34**(5):979–994, September 1988.

[175] Rodney M. Goodman and Padhriac J. Smyth. Decision tree design using information theory. *Knowledge Acquisition*, **2**:1–19, 1990.

[176] Michael T. Goodrich, Vincent Mirelli, Mark Orletsky, and Jeffery Salowe. Decision tree conctruction in fixed dimensions: Being global is hard but local greed is good. Technical Report TR-95-1, Johns Hopkins University, Department of Computer Science, Baltimore, MD 21218, May 1995.

[177] L. Gordon and R. A. Olshen. Asymptotically efficient solutions to the classification problem. *Annals of Statistics*, **6**(3):515–533, 1978.

[178] R. Graham and P. Hell. On the history of minimum spanning tree problem. *Annals of History of Computing*, **7**, 1985.

[179] N. A. B. Gray. Capturing knowledge through top-down induction of decision trees. *IEEE Expert*, **5**(3):41–50, June 1990.

[180] Robert M. Gray. Vector quantization. *IEEE ASSP Magazine*, pages 4–28, April 1984.

[181] Gabriel Groner. *Statistical Analysis of Adaptive Linear Classifiers.* PhD thesis, Stanford University, 1964.

[182] A. Guenoche. Representation of classifications as trees. *RAIRO Recherche Operationelle*, **20**(4):341–353, November 1986.

[183] R.K. Gulati, R. Gupta, P. Gothoskar, and S. Khobragade. Ultraviolet stellar spectral classification using multilevel tree neural networks. *Vistas in Astronomy*, **38**:293, 1993. Part 3: Neural Networks in Astronomy.

[184] Heng Guo and Saul B. Gelfand. Classification trees with neural network feature extraction. *IEEE Transactions on Neural Networks.*, **3**(6):923–933, November 1992.

[185] Y. Guo and K.J. Dooley. Distinguishing between mean, variance and autocorrelation changes in statistical quality control. *International Journal of Production Research*, **33**(2):497–510, February 1995.

[186] R. Gupta, S.A. Smolka, and S. Bhaskar. On randomization in sequential and distributed algorithms. *ACM Computing Surveys*, **26**(1):7–86, March 1994.

[187] Ouzden Guur-Ali and William A. Wallace. Induction of rules subject to a quality constraint: Probabilistic inductive learning. *IEEE Transactions on Knowldge and Data Engineering*, **5**(6):979–984, December 1993. Special Issue on Learning and Discovery in Knowledge-based databases.

[188] S.E. Hampson and D.J. Volper. Linear function neurons: Structure and training. *Biological Cybernetics*, **53**(4):203–217, 1986.

[189] Thomas R. Hancock. Learning kμ decision trees on the uniform distribution. In *Proceedings of the Sixth Annual Workshop on Computational Learning Theory*, pages 352–360, July 1993.

[190] D. J. Hand. *Discrimination and Classification.* Wiley, Chichester, UK, 1981.

[191] D. J. Hand, editor. *Artificial Intelligence Frontiers in Statistics III.* Chapman & Hall, London, 1993.

[192] W. Hanisch. Design and optimization of a hierarchical classifier. *Journal of new Generation Computer Systems*, **3**(2):159–173, 1990.

[193] L. K. Hansen and P. Salomon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **12**(10):993–1001, 1990.

[194] D. Harrison and D.L. Rubinfeld. Hedonic prices and the demand for clean air. *Journal of Environmental Economics and Management*, **5**:81–102, 1978.

[195] A. Hart. Experience in the use of an inductive system in knowledge engineering. In M. Bramer, editor, *Research and Development in Expert Systems.* Cambridge University Press, Cambridge, MA, 1984.

[196] P. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, **14**(3), May 1968.

[197] Carlos R. P. Hartmann, Pramod K. Varshney, Kishan G. Mehrotra, and Carl L. Gerberich. Application of information theory to the construction of efficient decision trees. *IEEE Transactions on Information Theory*, **IT-28**(4):565–577, July 1982.

[198] R. E. Haskell and A. Noui-Mehidi. Design of hierarchical classifiers. In N. A. Sherwani, E. de Doncker, and J. A. Kapenga, editors, *Computing in the 90's: The First Great Lakes Computer Science Conference Proceedings*, pages 118–124, Berlin, 1991. Springer-Verlag. Conference held in Kalamazoo, MI on 18th-20th, October 1989.

[199] N.D. Hatziargyriou, G.C. Contaxis, and N.C. Sideris. A decision tree method for on-line steady state security assessment. *IEEE Transactions on Power Systems*, **9**(2):1052, 1994.

[200] Patrick Hayes and Kenneth Ford. Turing test considered harmful. In IJCAI-95 [223], pages 972–977. Invited talk.

[201] Mark A. Heap and M. R. Mercer. Least upper bounds on OBDD sizes. *IEEE Transactions on Computers*, **43**(6):764–767, June 1994.

[202] D. Heath. *A Geometric Framework for Machine Learning.* PhD thesis, Johns Hopkins University, Baltimore, MD, 1992.

[203] D. Heath, S. Kasif, and S. Salzberg. k-DT: A multi-tree learning method. In *Proceedings of the Second International Workshop on Multistrategy Learning*, pages 138–149, Harpers Ferry, WV, 1993. George Mason University.

[204] D. Heath, S. Kasif, and S. Salzberg. Learning oblique decision trees. In IJCAI-93 [221], pages 1002–1007. Editor: Ruzena Bajcsy.

[205] David P. Helmhold and Robert E. Schapire. Predicting nearly as well as the best pruning of a decision tree. In *Proceedings of the 8th Annual Conference on Computational Learning Theory*, pages 61–68, New York, NY, 1995. ACM Press.

[206] Ernest G. Henrichon Jr. and King-Sun Fu. A nonparametric partitioning procedure for pattern classification. *IEEE Transactions on Computers*, **C-18**(7):614–624, July 1969.

[207] Gabor T. Herman and K.T. Daniel Yeung. On piecewise-linear classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(7):782–786, July 1992.

[208] Klaus-U Hoeffgen, Hans-U Simon, and Kevin S. Van Horn. Robust trainability of single neurons. *Journal of Computer System Sciences*, **50**(1):114–125, 1995.

[209] R. Holte. Very simple classification rules perform well on most commonly used datasets. *Machine Learning*, **11**(1):63–90, 1993.

[210] Ellis Horowitz and Sartaj Sahni. *Fundamentals of Computer Algorithms.* Computer Science Press, Rockville, MD, 1984.

[211] A. S. Houston, R. J. Iorns, and M. A. Macleod. The use of thyroid function studies. *Nuclear Medicine Communications*, **12**:497–506, 1991.

[212] G. E. Hughes. On the mean accuracy of statistical pattern recognition. *IEEE Transactions on Information Theory*, **IT-14**(1):55–63, January 1968.

[213] K. J. Hunt. Classification by induction: Applications to modelling and control of non-linear dynamic systems. *Intelligent Systems Engineering*, **2**(4):231–245, Winter 1993.

[214] D. Hunter, S. Faber, R. Light, and E. Shaya. Stellar photometry and zero-points. In S. Faber, editor, *Final Orbital/Science Verification Report*, 1991.

[215] Laurent Hyafil and Ronald L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, **5**(1):15–17, 1976.

[216] Toshihide Ibaraki and Saburo Muroga. Adaptive linear classifiers by linear programming. Technical Report 284, Department of Computer Science, University of Illinois, Urbana-Champaign, 1968.

[217] M. Ichino and Jack Sklansky. Optimum feature selection by zero-one integer programming. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-14**:737–746, September/October 1984.

[218] Y. Iikura and Y. Yasuoka. Utilization of a best linear discriminant function for designing the binary decision tree. *International Journal of Remote Sensing*, **12**(1):55–67, January 1991.

[219] *IJCAI-89: Proceedings of the Eleventh International Joint Conference on Artificial Intelligence.* Morgan Kaufmann Publishers Inc., San Mateo, CA, 1989. Editor: N. S. Sridharan.

[220] *IJCAI-91: Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, volume 2, Darling Harbour, Sydney, Australia, 24–30th, August 1991. Morgan Kaufmann Publishers Inc., San Mateo, CA. Editors: John Mylopoulos and Ray Reiter.

[221] *IJCAI-93: Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, volume 2, Chambery, France, 28th August–3rd September 1993. Morgan Kaufmann Publishers Inc., San Mateo, CA. Editor: Ruzena Bajcsy.

[222] *Data Engineering for Inductive Learning*, Montreal, Canada, 16th–21st, August 1995. Morgan Kaufmann Publishers Inc., San Mateo, CA. Workshop organized by Peter Turney. http://ai.iit.nrc.ca/DEIL/.

[223] *IJCAI-95: Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, Montreal, Canada, 16th–21st, August 1995. Morgan Kaufmann Publishers Inc., San Mateo, CA. Editor: Chris Mellish.

[224] I. F. Imam and Ryszard S. Michalski. Should decision trees be learned from examples or from decision rules? In *Methodologies for Intelligent Systems: 7th International Symposium. ISMIS '93*, volume 689 of *Lecture Notes in Computer Science*, pages 395–404. Springer-Verlag, Trondheim, Norway, June 1993.

[225] Keki B. Irani, Cheng Jie, Usama M. Fayyad, and Qian Zhaogang. Applying machine learning to semiconductor manufacturing. *IEEE Expert*, **8**(1):41–47, February 1993.

[226] P. Israel and C. Koutsougeras. A hybrid electro-optical architecture for classification trees and associative memory mechanisms. *International Journal on Artificial Intelligence Tools (Architectures, Languages, Algorithms)*, **2**(3):373–393, September 1993.

[227] A. K. Jain and B. Chandrasekaran. Dimensionality and sample size considerations in pattern recognition. In Krishnaiah and Kanal [265], pages 835–855.

[228] Mike James. *Classification Algorithms.* Wiley-Interscience Publications, 1985.

[229] George H. John. Robust linear discriminant trees. In AI&Statistics-95 [5], pages 285–291.

[230] George H. John, Ron Kohavi, and Karl Pfleger. Irrelevant features and the subset selection problem. In ML-94 [331], pages 121–129. Editors: William W. Cohen and Haym Hirsh.

[231] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *Journal of Computer and System Sciences*, **37**:79–100, 1988.

[232] S.C. Johnson. Hierarchical clustering schemes. *Psychometrika*, **32**(3), September 1967.

[233] Michael I. Jordan and R. A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, **6**:181–214, 1994.

[234] J. Judmaier, P. Meyersbach, G. Weiss, H. Wachter, and G. Reibnegger. The role of Neopterin in assessing disease activity in Crohn's disease: Classification and regression trees. *The American Journal of Gastroenterology*, **88**(5):706, May 1993.

[235] G. Kalkanis. The application of confidence interval error analysis to the design of decision tree classifiers. *Pattern Recognition Letters*, **14**(5):355–361, May 1993.

[236] Laveen N. Kanal. Patterns in pattern recognition: 1968-1974. *IEEE Transactions in Information Theory*, **20**:697–722, 1974.

[237] Laveen N. Kanal. Problem solving methods and search strategies for pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-1**:193–201, 1979.

[238] Laveen N. Kanal and B. Chandrasekaran. On dimensionality and sample size in statistical pattern classification. *Pattern Recognition*, **3**:225–234, 1971.

[239] Younkyung Cha Kang. *Randomized Algorithms for Query Optimization.* PhD thesis, Department of Computer Sciences, University of Wisconsin-Madison, October 1991. Computer Sciences Report #1053.

[240] G. V. Kass. An exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, **29**(2):119–127, 1980.

[241] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis.* Wiley-Interscience Publication, 1990.

[242] Michael J. Kearns and Umesh Virkumar Vazirani. *An Introduction to Computational Learning Theory.* MIT Press, Cambridge, MA, 1994.

[243] Davis M. Kennedy. Decision tree bears fruit. *Products Finishing*, **57**(10):66, July 1993.

[244] J. D. Kennefick, R. R. Carvalho, S. G. Djorgovski, M. M. Wilber, E. S. Dickson, N. Weir, U. Fayyad, and J. Roden. The discovery of five quasars at $z > 4$ using the second Palomar Sky Survey. *The Astronomical Journal*, **110**(1):78, 1995.

[245] Randy Kerber. Chimerge: Discretization of numeric attributes. In AAAI-92 [7], pages 123–128.

[246] B. Khoshnevis and S. Parisay. Machine learning and simulation: Application in queuing systems. *Simulation*, **61**(5):294–302, 1993.

[247] Byungyong Kim and David Landgrebe. Hierarchical decision tree classifiers in high-dimensional and large class data. *IEEE Transactions on Geoscience and Remote Sensing*, **29**(4):518–528, July 1991.

[248] Hyunsoo Kim. *PAC Learning: A Decision Tree with Pruning.* PhD thesis, University of Florida, 1992.

[249] Hyunsoo Kim and G. J. Koehler. An investigation on the conditions of pruning an induced decision tree. *European Journal of Operational Research*, **77**(1):82, August 1994.

[250] Sung-Ho Kim. A general property among nested, pruned subtrees of a decision support tree. *Communications in Statistics—Theory and Methods*, **23**(4):1227–1238, April 1994.

[251] Kenji Kira and Larry A. Rendell. The feature selection problem: Traditional methods and a new algorithm. In AAAI-92 [7], pages 129–134.

[252] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecci. Optimization by simulated annealing. *Science*, **220**(4598):671–680, May 1983.

[253] Y. Kodratoff and M. Manago. Generalization and noise. *International Journal of Man-Machine Studies*, **27**:181–204, 1987.

[254] Y. Kodratoff and R.S. Michalski, editors. *Machine learning, and Artificial Intelligence approach*, volume 3. Morgan Kaufmann, San Mateo, CA, 1990.

[255] Y. Kodratoff and S. Moscatelli. Machine learning for object recognition and scene analysis. *Internationa Journal of Pattern recognition and AI*, **8**(1):259–304, 1994.

[256] Ron Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection. In IJCAI-95 [223], pages 1137–1143. Editor: Chris Mellish.

[257] Ron Kohavi and Chia-Hsin Li. Oblivious decision trees, graphs and top-down pruning. In IJCAI-95 [223], pages 1071–1077. Editor: Chris Mellish.

[258] P. Kokol, M. Mernik, J. Zavrsnik, and K. Kancler. Decision trees based on automatic learning and their use in cardiology. *Journal of Medical Systems*, **18**(4):201, 1994.

[259] Igor Kononenko. Inductive and bayesian learning in medical diagnosis. *Applied Artificial Intelligence*, **7**(4):317–337, October-December 1993.

[260] Igor Kononenko. On biases in estimating multi-valued attributes. In IJCAI-95 [223], pages 1034–1040. Editor: Chris Mellish.

[261] Igor Kononenko and Ivan Bratko. Information based evaluation criterion for classifier's performance. *Machine Learning*, **6**(1):67–80, January 1991.

[262] J. A. Kors and J. H. Van Bemmel. Classification methods for computerized interpretation of the electrocardiogram. *Methods of Information in Medicine*, **29**(4):330–336, September 1990.

[263] V. A. Kovalevsky. The problem of character recognition from the point of view of mathematical statistics. In V. A. Kovalevsky, editor, *Character Readers and Pattern Recognition*. Spartan, New York, 1968.

[264] J. R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In H. P. Schwefel and R. Männer, editors, *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, volume 496 of *Lecture Notes in Computer Science*, pages 124–128, Dortmund, Germany, October 1991. Springer-Verlag, Berlin, Germany.

[265] Paruchuri Rama Krishnaiah and Laveen N. Kanal, editors. *Classification, Pattern Recognition and Reduction of Dimensionality*, volume 2 of *Handbook of Statistics*. North-Holland Publishing Company, Amsterdam, 1987.

[266] Srinivasan Krishnamoorthy and Douglas Fisher. Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering*, **21**(2):126–137, February 1995.

[267] M. Kubat, G. Pfurtscheller, and D. Flotzinger. AI-based approach to automatic sleep classification. *Biological Cybernetics*, **70**(5):443–448, 1994.

[268] Ashok K. Kulkarni. On the mean accuracy of hierarchical classifiers. *IEEE Transactions on Computers*, **C-27**(8):771–776, August 1978.

[269] Michael J. Kurtz. Astronomical object classification. In E. S. Gelsema and Laveen N. Kanal, editors, *Pattern Recognition and Artificial Intelligence*, pages 317–328. Elsevier Science Publishers, Amsterdam, 1988.

[270] M. W. Kurzynski. The optimal strategy of a tree classifier. *Pattern Recognition*, **16**:81–87, 1983.

[271] M. W. Kurzynski. On the multi-stage Bayes classifier. *Pattern Recognition*, **21**(4):355–365, 1988.

[272] M. W. Kurzynski. On the identity of optimal strategies for multi-stage classifiers. *Pattern Recognition Letters*, **10**(1):39–46, July 1989.

[273] Eyal Kushilevitz and Yishay Mansour. Learning decision trees using the fourier spectrum. *SIAM Journal of Computing*, **22**(6):1331–1348, 1993. Earlier version in STOC-91.

[274] S.W. Kwok and Carter. C. Multiple decision trees. In R.D. Schachter, T.S. Levitt, L.N. Kanal, and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, volume 4, pages 327–335. Elsevier Science, Amsterdam, 1990.

[275] G. Landeweerd, T. Timmers, E. Gersema, M. Bins, and M. Halic. Binary tree versus single level tree classification of white blood cells. *Pattern Recognition*, **16**:571–577, 1983.

[276] P. Langley and S. Sage. Scaling to domains with many irrelevant features. Unpublished manuscript. Learning Systems Department, Siemens Corporate Research, Princeton, NJ, 1993.

[277] M. Lebowitz. Categorizing numeric information for generalization. *Cognitive Science*, **9**:285–308, 1985.

[278] Chulhee Lee and David Landgrebe. Decision boundary feature extraction for nonparametric classification. *IEEE Transactions on Systems, Man and Cybernetics*, **23**(2):433–444, March/April 1993.

[279] C.Y. Lee. Representation of switching circuits by binary decision programs. *Bell Systems Technical Journal*, **38**:985–999, July 1959.

[280] Kun Chang Lee and Sung Joo Park. PRTSM: Pattern recognition based time series modeler. *Computer Science in Economics and Management*, **2**(3):239–254, 1989.

[281] Seong-Whan Lee. Noisy Hangul character recognition with fuzzy tree classifier. *Proceedings of SPIE*, **1661**:127–136, 1992. Volume title: Machine vision applications in character recognition and industrial inspection. Conference location: San Jose, CA. 10th–12th February, 1992.

[282] Wendy Lehnert, Stephen Soderland, David Aronow, Fangfang Feng, and Avinoam Shmueli. Inductive text classification for medical applications. *Journal of Experimental and Theoretical Artificial Intelligence*, **7**(1):49–80, January-March 1995.

[283] Debra A. Lelewer and Daniel S. Hirschberg. Data compression. *ACM Computing Surveys*, **19**(3):261–296, 1987.

[284] David D. Lewis and Jason Catlett. Heterogeneous uncertainty sampling for supervised learning. In ML-94 [331]. Editors: William W. Cohen and Haym Hirsh.

[285] P.M. Lewis. The characteristic selection problem in recognition systems. *IRE Transactions on Information Theory*, **IT-18**:171–178, 1962.

[286] Xiaobo Li and Richard C. Dubes. Tree classifier design with a permutation statistic. *Pattern Recognition*, **19**(3):229–235, 1986.

[287] Jianhia Lin and L.A. Storer. Design and performance of tree structured vector quantizers. *Information Processing and Management*, **30**(6):851–862, 1994.

[288] Jianhua Lin, J. A. Storer, and M. Cohn. Optimal pruning for tree-structured vector quantizers. *Information Processing and Management*, **28**(6):723–733, 1992.

[289] Jyh-Han Lin and J. S. Vitter. Nearly optimal vector quantization via linear programming. In J. A. Storer and M. Cohn, editors, *DCC 92. Data Compression Conference*, pages 22–31, Los Alamitos, CA, March 24th–27th 1992. IEEE Computer Society Press.

[290] Y. K. Lin and King-Sun Fu. Automatic classification of cervical cells using a binary tree classifier. *Pattern Recognition*, **16**(1):69–80, 1983.

[291] D. V. Lindley. On a measure of the information provided by an experiment. *Annals of Mathematical Statistics*, **27**(4):986–1005, December 1956.

[292] W. Z. Liu and A. P. White. The importance of attribute selection measures in decision tree induction. *Machine Learning*, **15**:25–41, 1994.

[293] Zhen-Ping Lo and B. Bavarian. Development of a two-stage neural network classifier. *Journal of Artificial Neural Networks*, **1**(3):307–327, 1994.

[294] Wei-Yin Loh and Nunta Vanichsetakul. Tree-structured classification via generalized discriminant analysis. *Journal of the American Statistical Association*, **83**(403):715–728, September 1988.

[295] William J. Long, John L. Griffith, Harry P. Selker, and Ralph B. D'Agostino. A comparison of logistic regression to decision tree induction in a medical domain. *Computers and Biomedical Research*, **26**(1):74–97, February 1993.

[296] D.W. Loveland. Performance bounds for binary testing with arbitrary weights. *Acta Informatica*, **22**:101–114, 1985.

[297] David Lubinsky. Algorithmic speedups in growing classification trees by using an additive split criterion. In AI&Statistics-93 [4], pages 435–444.

[298] David Lubinsky. *Bivariate splits and consistent split criteria in dichotomous classification trees.* PhD thesis, Department of Computer Science, Rutgers University, New Brunswick, NJ, 1994.

[299] David Lubinsky. Classification trees with bivariate splits. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks and Complex Problem-Solving Technologies*, **4**(3):283–296, July 1994.

[300] David Lubinsky. Tree structured interpretable regression. In AI&Statistics-95 [5], pages 331–340.

[301] Paul Lukowitz, Ernst A. Heinz, Lutz Prechelt, and Walter F. Tichy. Experimental evaluation in computer science: A quantitative study. *Journal of Systems and Software*, January 1995. Anonymous FTP at ftp.ira.uka.de in the file /pub/uni-karlsruhe/papers/techreports/1994/1994-17.ps.Z.

[302] Ren C. Luo, Ralph S. Scherp, and Mark Lanzo. Object identification using automated decision tree construction approach for robotics applications. *Journal of Robotic Systems*, **4**(3):423–433, June 1987.

[303] J. F. Lutsko and B. Kuijpers. Simulated annealing in the construction of near-optimal decision trees. In AI&Statistics-93 [4].

[304] Lymphography data. Available in the UCI ML Repository. Obtained from the University Medical Centre, Institute of Oncology, Ljubljana, Yugoslavia. Data provided by Matjaz Zwitter and Milan Soklic.

[305] F. Maffioli, M.G. Speranza, and C. Vercellis. Randomized algorithms. In M. OhEigeartaigh, J.K. Lenstra, and A.H.G. Rinnooy Kan, editors, *Combinatorial Optimization - Annotated Bibliographies*, pages 89–105. 1985.

[306] John Makhoul, Salim Roucos, and Herbert Gish. Vector quantization in speech coding. *Proceedings of the IEEE*, **73**:1551–1588, November 1985. Invited paper.

[307] Olvi Mangasarian. Mathematical programming in neural networks. *ORSA Journal on Computing*, **5**(4):349–360, Fall 1993.

[308] Olvi L. Mangasarian. Misclassification minimization, 1994. Unpublished manuscript.

[309] Olvi L. Mangasarian, R. Setiono, and W. Wolberg. Pattern recognition via linear programming: Theory and application to medical diagnosis. In *SIAM Workshop on Optimization*, 1990.

[310] López de Màntaras. Technical note: A distance-based attribute selection measure for decision tree induction. *Machine Learning*, **6**(1):81–92, 1991.

[311] J. Kent Martin. Evaluating and comparing classifiers: complexity measures. In AI&Statistics-95 [5], pages 372–378.

[312] J. Kent Martin. An exact probability metric for decision tree splitting and stopping. In AI&Statistics-95 [5], pages 379–385.

[313] Dean P. McKenzie and Lee Hun Low. The construction of computerized classification systems using machine learning algorithms: An overview. *Computers in Human Behaviour*, **8**(2/3):155–167, 1992.

[314] Dean P. McKenzie, P. D. McGorry, C. S. Wallace, Lee Hun Low, D. L. Copolov, and B. S. Singh. Constructing a minimal diagnostic decision tree. *Methods of Information in Medicine*, **32**(2):161–166, April 1993.

[315] K. L. McMillan. *Symbolic model checking: an approach to the state explosion problem.* PhD thesis, Carnegie Mellon University, School of Computer Science, 1992.

[316] R.J. McQueen, S. R. Garner, C.G. Nevill-Manning, and I.H. Witten. Applying machine learning to agricultural data. *Computers and Electronics in Agriculture*, **12**(4):275–293, June 1995.

[317] Nimrod Megiddo. On the complexity of polyhedral separability. *Discrete and Computational Geometry*, **3**:325–337, 1988.

[318] William S. Meisel and Demetrios A. Michalopoulos. A partitioning algorithm with application in pattern classification and the optimization of decision trees. *IEEE Transactions on Computers*, **C-22**(1):93–103, January 1973.

[319] Joseph J. Mezrich. When is a tree a hedge? *Financial Analysts Journal*, pages 75–81, November-December 1994.

[320] Donald Michie. The superarticulatory phenomenon in the context of software manufacture. *Proceedings of the Royal Society of London*, **405A**:185–212, 1986.

[321] Donald Michie. Current developments in expert systems. In J. Ross Quinlan, editor, *Applications of Expert Systems*, pages 137–156. Addison-Wesley, Reading, MA, 1987.

[322] A. J. Miller. *Subset Selection in Regression.* Chapman and Hall, 1990.

[323] John Mingers. Expert systems — rule induction with statistical data. *Journal of the Operational Research Society*, **38**(1):39–47, 1987.

[324] John Mingers. An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, **4**(2):227–243, 1989.

[325] John Mingers. An empirical comparison of selection measures for decision tree induction. *Machine Learning*, **3**:319–342, 1989.

[326] M. Minsky and S. Papert. *Perceptrons.* MIT Press, Cambridge, MA, 1969.

[327] Tom Mitchell, Rich Caruana, Dayne Freitag, John McDermott, and David Zabowski. Experience with a learning personal assistant. *Communications of the ACM*, July 1994.

[328] Masahiro Miyakawa. Optimum decision trees – an optimal variable theorem and its related applications. *Acta Informatica*, **22**(5):475–498, 1985.

[329] Masahiro Miyakawa. Criteria for selecting a variable in the construction of efficient decision trees. *IEEE Transactions on Computers*, **38**(1):130–141, January 1989.

[330] *Machine Learning: Proceedings of the Tenth International Conference*, University of Massachusetts, Amherst, MA, 27–29th, June 1993. Morgan Kaufmann Publishers Inc. Editor: Paul E. Utgoff.

[331] *Machine Learning: Proceedings of the Eleventh International Conference*, Rutgers University, New Brunswick, NJ, 10–13th, July 1994. Morgan Kaufmann Publishers Inc. Editors: William W. Cohen and Haym Hirsh.

[332] *Applying Machine Learning in Practice*, Tahoe City, CA, 10–13th, July 1995. Morgan Kaufmann Publishers Inc., San Mateo, CA. Workshop organized by David Aha. http://www.aic.nrl.navy.mil/ aha/imlc95-workshop/.

[333] *Machine Learning: Proceedings of the Twelfth International Conference*, Tahoe City, CA, 10–13th, July 1995. Morgan Kaufmann Publishers Inc., San Mateo, CA. Editor: Jeffrey Schlimmer.

[334] Dunja Mladenic. Combinatorial optimization in inductive concept learning. In ML-93 [330], pages 205–211. Editor: Paul E. Utgoff.

[335] Advait Mogre, Robert McLaren, James Keller, and Raghuram Krishnapuram. Uncertainty management for rule-based systems with application to image analysis. *IEEE Transactions on Systems, Man and Cybernetics*, **24**(3):470–481, March 1994.

[336] Andrew W. Moore and Mary S. Lee. Efficient algorithms for minimizing cross validation error. In ML-94 [331], pages 190–198. Editors: William W. Cohen and Haym Hirsh.

[337] Bernard M. E. Moret, M. G. Thomason, and R. C. Gonzalez. The activity of a variable and its relation to decision trees. *ACM Transactions on Programming Language Systems*, **2**(4):580–595, October 1980.

[338] Bernard M.E. Moret. Decision trees and diagrams. *Computing Surveys*, **14**(4):593–623, December 1982.

[339] J. N. Morgan and R. C. Messenger. THAID: a sequential search program for the analysis of nominal scale dependent variables. Technical report, Institute for Social Research, University of Michigan, Ann Arbor, MI, 1973.

[340] D. T. Morris and D. Kalles. Decision trees and domain knowledge in pattern recognition. In Gelsema and Kanal [164], pages 25–36.

[341] Paul Morris. The breakout method for escaping from local minima. In AAAI-93 [8], pages 40–45.

[342] A. N. Mucciardi and E. E. Gose. A comparison of seven techniques for choosing subsets of pattern recognition properties. *IEEE Transactions on Computers*, **C-20**(9):1023–1031, September 1971.

[343] W. Muller and F. Wysotzki. Automatic construction of decision trees for classification. *Annals of Operations Research*, **52**:231, 1994.

[344] O. J. Murphy and R. L. McCraw. Designing storage efficient decision trees. *IEEE Transactions on Computers*, **40**(3):315–319, March 1991.

[345] Patrick M. Murphy. An empirical analysis of the benefit of decision tree size biases as a function of concept distribution. Submitted to the Machine Learning journal, July 1994.

[346] Patrick M. Murphy and David Aha. UCI repository of machine learning databases – a machine-readable data repository. Maintained at the Department of Information and Computer Science, University of California, Irvine. Anonymous FTP from ics.uci.edu in the directory pub/machine-learning-databases, 1994.

[347] Patrick M. Murphy and Michael J. Pazzani. Exploring the decision forest: An empirical investigation of Occam's Razor in decision tree induction. *Journal of Artificial Intelligence Research*, **1**:257–275, 1994.

[348] Sreerama K. Murthy. Data exploration using decision trees: A survey. In preparation, 1995. http://www.cs.jhu.edu/grad/murthy.

[349] Sreerama K. Murthy. Using structure to improve decision trees. In AI&Statistics-95 [5], pages 403–409.

[350] Sreerama K. Murthy, S. Kasif, S. Salzberg, and R. Beigel. OC1: Randomized induction of oblique decision trees. In AAAI-93 [8], pages 322–327.

[351] Sreerama K. Murthy, Simon Kasif, and Steven Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, **2**:1–33, August 1994.

[352] Sreerama K. Murthy and Steven Salzberg. Clustering astronomical objects using minimum spanning trees. Technical report, Dept. of Computer Science, Johns Hopkins University, July 1992.

[353] Sreerama K. Murthy and Steven Salzberg. Decision tree induction: How effective is the greedy heuristic? In *Proceedings of the First International Conference on Knowledge Discovery in Databases*, Montreal, Canada, August 1995.

[354] Sreerama K. Murthy and Steven Salzberg. Lookahead and pathology in decision tree induction. In IJCAI-95 [223]. to appear.

[355] R. Musick, Jason Catlett, and S. Russell. Decision theoretic subsampling for induction on large databases. In ML-93 [330], pages 212–219. Editor: Paul E. Utgoff.

[356] D. Mutchler. The multi-player version of minimax displays game pathology. *Artificial Intelligence*, **64**(2):323–336, December 1993.

[357] Y. Nakamura, S. Abe, Y. Ohsawa, and M. Sakauchi. A balanced hierarchical data structure for multidimensional data with highly efficient dynamic characteristics. *IEEE Transactions on Knowledge and Data Engineering*, **5**(4):682–694, August 1993.

[358] P. M. Narendra and K. Fukanaga. A branch and bound algorithm for feature subset selection. *IEEE Transactions on Computers*, **C-26**(9):917–922, 1977.

[359] S. C. Narula and J. F. Wellington. The minimum sum of absolute errors regression: a state of the art survey. *International Statistical Review*, **50**:317–326, 1982.

[360] Dana S. Nau. Decision quality as a function of search depth on game trees. *Journal of the Association of Computing Machinery*, **30**(4):687–708, October 1983.

[361] G. E. Naumov. NP-completeness of problems of construction of optimal decision trees. *Soviet Physics, Doklady*, **36**(4):270–271, April 1991.

[362] V. Nedeljkovic and M. Milosavljevic. On the influence of training set data preprocessing on neural networks training. In *Proceedings of the 11th International Conference on Pattern Recognition*, volume II, pages 33–36, Los Alamitos, CA, September 1992. International Association for Pattern Recognition, IEEE Computer Society Press.

[363] T. Niblett. Constructing decision trees in noisy domains. In I. Bratko and N. Lavrac, editors, *Progress in Machine Learning*. Sigma Press, England, 1986.

[364] N.J. Nilsson. *Learning Machines*. Morgan Kaufmann, 1990.

[365] Steven W. Norton. Generating better decision trees. In IJCAI-89 [219], pages 800–805. Editor: N. S. Sridharan.

[366] M. Núñez. The use of background knowledge in decision tree induction. *Machine Learning*, **6**:231–250, 1991.

[367] S.C. Odewahn, E.B. Stockwell, R.L. Pennington, R.M. Humphreys, and W.A. Zumach. Automated star-galaxy discrimination with neural networks. *Astronomical Journal*, **103**(1):318–331, 1992.

[368] J. Oliver. Decision graphs—an extension of decision trees. In AI&Statistics-93 [4].

[369] Colm A. O'Muircheartaigh. Statistical analysis in the context of survey research. In O'Muircheartaigh and Payne [370], pages 1–40.

[370] Colm A. O'Muircheartaigh and Clive Payne, editors. *The analysis of survey data*, volume I. John Wiley & Sons, Chichester, UK, 1977.

[371] Giulia M. Pagallo. *Adaptive Decision Tree Algorithms for Learning from Examples.* PhD thesis, University of California, Computer Research Laboratory, Santa Cruz, CA, June 1990.

[372] Giulia M. Pagallo and D. Haussler. Boolean feature discovery in empirical learning. *Machine Learning*, **5**(1):71–99, March 1990.

[373] Shailendra C. Palvia and Steven R. Gordon. Tables, trees and formulas in decision analysis. *Communications of the ACM*, **35**(10):104–113, October 1992.

[374] Youngtae Park. A comparison of neural net classifiers and linear tree classifiers: Their similarities and differences. *Pattern Recognition*, **27**(11):1493–1503, 1994.

[375] Youngtae Park and Jack Sklansky. Automated design of linear tree classifiers. *Pattern Recognition*, **23**(12):1393–1412, 1990.

[376] Yountae Park and Jack Sklansky. Automated design of multiple-class piecewise linear classifiers. *Journal of Classification*, **6**:195–222, 1989.

[377] A. Patterson and T. Niblett. ACLS user manual. Technical report, MIRU, ITL, Universty of Edinburgh, 1982.

[378] Krishna R. Pattipati and Mark G. Alexandridis. Application of heuristic search and information theory to sequential fault diagnosis. *IEEE Transactions on Systems, Man and Cybernetics*, **20**(4):872–887, July/August 1990.

[379] R. W. Payne and D. A. Preece. Identification keys and diagnostic tables: A review. *Journal of the Royal Statistical Society: series A*, **143**:253, 1980.

[380] Judea Pearl. On the connection between the complexity and credibility of inferred models. *International Journal of General Systems*, **4**:255–264, 1978.

[381] R. A. Pearson and P. E. Stokes. Vector evaluation in induction algorithms. *International Journal of High Speed Computing*, **2**(1):25–100, March 1990.

[382] F. Pipitone, K. A. De Jong, and W. M. Spears. An artificial intelligence approach to analog systems diagnosis. In Ruey-wen Liu, editor, *Testing and Diagnosis of Analog Circuits and Systems.* Van Nostrand-Reinhold, New York, 1991.

[383] Selwyn Piramuthu, Narayan Raman, and Michael J. Shaw. Learning-based scheduling in a flexible manufacturing flow line. *IEEE Transactions on Engineering Management*, **41**(2):172–182, May 1994.

[384] N. J. Pizzi and D. Jackson. Comparitive review of knowledge engineering and inductive learning using data in a medical domain. *Proceedings of the SPIE: The International Society for Optical Engineering*, **1293**(2):671–679, April 1990.

[385] Lutz Prechelt. A study of experimental evaluations of neural network algorithms: Current research practice. Technical Report 19/94, Fakultät für Informatik, Universität Karlsruhe, 76128 Karlsruhe, Germany, August 1994. anonymous FTP: /pub/papers/techreports/1994/1994-19.ps.Z on ftp.ira.uka.de.

[386] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[387] Shi Qing-Yun and King-Sun Fu. A method for the design of binary tree classifiers. *Pattern Recognition*, **16**:593–603, 1983.

[388] John Ross Quinlan. Discovering rules by induction from large collections of examples. In Donald Michie, editor, *Expert Systems in the Micro Electronic Age.* Edinburgh University Press, Edinburgh, UK, 1979.

[389] John Ross Quinlan. Learning efficient classification procedures and their application to chess end games. In R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach.* Morgan Kaufmann, San Mateo, CA, 1983.

[390] John Ross Quinlan. The effect of noise on concept learning. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2. Morgan Kauffman, San Mateo, CA, 1986.

[391] John Ross Quinlan. Induction of decision trees. *Machine Learning*, **1**:81–106, 1986.

[392] John Ross Quinlan. Generating production rules from decision trees. In *Proceedings of Tenth International Joint Conference on Artificial Intelligence*, pages 304–307, Milan, Italy, 1987. Morgan Kaufmann.

[393] John Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, **27**:221–234, 1987.

[394] John Ross Quinlan. An empirical comparison of genetic and decision tree classifiers. In *Fifth International Conference on Machine Learning*, pages 135–141, Ann Arbor, Michigan, 1988. Morgan Kaufmann.

[395] John Ross Quinlan. Unknown attribute values in induction. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 164–168, San Mateo, CA, 1989. Morgan Kaufmann.

[396] John Ross Quinlan. Decision trees and decisionmaking. *IEEE Transactions on Systems, Man, and Cybernetics*, **20**(2):339–346, March-April 1990.

[397] JOHN ROSS QUINLAN. Probabilistic decision trees. In R.S.Michalski and Y. Kodratoff, editors, *Machine Learning: An Artificial Intelligence Approach - Volume 3*. Morgan Kaufmann, San Mateo, CA, 1990.

[398] JOHN ROSS QUINLAN. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, San Mateo, CA, 1993.

[399] JOHN ROSS QUINLAN. Combining instance-based and model-based learning. In ML-93 [330], pages 236–243. Editor: Paul E. Utgoff.

[400] JOHN ROSS QUINLAN. Comparing connectionist and symbolic learning methods. In S. Hanson, G. Drastal, and R. Rivest, editors, *Computational Learning Theory and Natural Learning Systems: Constraints and Prospects*. MIT Press, 1993.

[401] JOHN ROSS QUINLAN AND R. M. CAMEROON-JONES. Oversearching and layered-search in empirical learning. In IJCAI-95 [223], pages 1019–1024. Editor: Chris Mellish.

[402] JOHN ROSS QUINLAN AND RONALD L. RIVEST. Inferring decision trees using the minimum description length principle. *Information and Computation*, **80**(3):227–248, March 1989.

[403] HARISH RAGAVAN AND LARRY RENDELL. Lookahead feature construction for learning hard concepts. In ML-93 [330], pages 252–259. Editor: Paul E. Utgoff.

[404] C. R. Rao, editor. *Computational Statistics*, volume 9 of *Handbook of Statistics*, Amsterdam, 1993. Elsevier Publications, North-Holland Publishing Company.

[405] LARRY RENDELL AND HARISH RAGAVAN. Improving the design of induction methods by analyzing algorithm functionality and data-based concept complexity. In IJCAI-93 [221], pages 952–958. Editor: Ruzena Bajcsy.

[406] ALFRED RENYI AND LASZLO VEKERDI. *Probability Theory*. North-Holland Publishing Company, Amsterdam, 1970.

[407] P. RIDDLE, R. SEGAL, AND O. ETZIONI. Representation design and brute-force induction in a Boeing manufacturing domain. *Applied Artificial Intelligence*, **8**(1):125–147, January-March 1994.

[408] JORMA RISANNEN. *Stochastic Complexity in Statistica Enquiry*. World Scientific, 1989.

[409] EVE A. RISKIN AND ROBERT M. GRAY. A greedy tree growing algorithm for the design of variable rate vector quantizers. *IEEE Transactions on Signal Processing*, **39**(11):2500–2507, November 1991.

[410] Eve A. Riskin and Robert M. Gray. Lookahead in growing tree-structured vector quantizers. In *ICASSP 91: International Conference on Accoustics, Speech and Signal Processing*, volume 4, pages 2289–2292, Toronto, Ontario, May 14th–17th 1991. IEEE.

[411] G. Ritter, H. Woodruff, S. Lowry, and T. Isenhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, **21**(6):665–669, 1975.

[412] Richard H. Roth. An approach to solving linear discrete optimization problems. *Journal of the ACM*, **17**(2):303–313, April 1970.

[413] E. Rounds. A combined non-parametric approach to feature selection and binary decision tree design. *Pattern Recognition*, **12**:313–317, 1980.

[414] Steven Rovnyak, Stein Kretsinger, James Thorp, and Donald Brown. Decision trees for real time transient stability prediction. *IEEE Transactions on Power Systems*, **9**(3):1417–1426, August 1994.

[415] Ron Rymon. An SE-tree based characterization of the induction problem. In ML-93 [330], pages 268–275. Editor: Paul E. Utgoff.

[416] Ron Rymon and N. M. Short, Jr. Automatic cataloging and characterization of earth science data using set enumeration trees. *Telematics and Informatics*, **11**(4):309–318, Fall 1994.

[417] S. Rasoul Safavin and David Landgrebe. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man and Cybernetics*, **21**(3):660–674, May/June 1991.

[418] M. Sahami. Learning non-linearly separable boolean functions with linear threshold unit trees and madaline-style networks. In AAAI-93 [8], pages 335–341.

[419] Sartaj Sahni. Approximate algorithms for the 0/1 knapsack problem. *Journal of the ACM*, **22**:115–124, 1975.

[420] Steven Salzberg. Distance metrics for instance-based learning. In *Methodologies for Intelligent Systems: 6th International Symposium*, pages 399–408, 1991.

[421] Steven Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, **6**:251–276, 1991.

[422] Steven Salzberg. Combining learning and search to create good classifiers. Technical Report JHU-92/12, Department of Computer Science, Johns Hopkins University, Baltimore MD, 1992.

[423] STEVEN SALZBERG. Locating protein coding regions in human DNA using a decision tree algorithm. *Journal of Computational Biology*, 1995. To appear in Fall.

[424] STEVEN SALZBERG, RUPALI CHANDAR, HOLLAND FORD, SREERAMA MURTHY, AND RICK WHITE. Decision trees for automated identification of cosmic-ray hits in Hubble Space Telescope images. *Publications of the Astronomical Society of the Pacific*, **107**:1–10, March 1995.

[425] ANANT SANKAR AND RICHARD J. MAMMONE. Growing and pruning neural tree networks. *IEEE Transactions on Computers*, **42**(3):291–299, March 1993.

[426] U. K. SARKAR, P. P. CHAKRABARTI, S. GHOSE, AND S. C. DESARKAR. Improving greedy algorithms by lookahead-search. *Journal of Algorithms*, **16**(1):1–23, January 1994.

[427] LAWRENCE SAUL AND MICHAEL I. JORDAN. Learning in Boltzmann trees. *Neural Computation*, **6**(6):1174–1184, November 1994.

[428] CULLEN SCHAFFER. Overfitting avoidance as bias. *Machine Learning*, **10**:153–178, 1993.

[429] CULLEN SCHAFFER. A conservation law for generalization performance. In ML-94 [331], pages 259–265. Editors: William W. Cohen and Haym Hirsh.

[430] CULLEN SCHAFFER. Conservation of generalization: A case study. Technical report, Department of Computer Science, CUNY/Hunter College, February 1995.

[431] T. M. SCHMIDL, P. C. COSMAN, AND ROBERT M. GRAY. Unbalanced non-binary tree-structured vector quantizers. In A. Singh, editor, *Conference Record of the Twenty-Seventh Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1519–1523, Los Alamitos, CA, November 1st–3rd 1993. IEEE Computer Society Press. Conf. held at Pacific Grove, CA.

[432] J. SCHUERMANN AND W. DOSTER. A decision-theoretic approach in hierarchical classifier design. *Pattern Recognition*, **17**:359–369, 1984.

[433] RICHARD W. SELBY AND ADAM A. PORTER. Learning from examples: Generation and evaluation of decision trees for software resource analysis. *IEEE Transactions on Software Engineering*, **14**(12):1743–1757, December 1988.

[434] BART SELMAN AND HENRY A. KAUTZ. An empirical study of greedy local search for satisfiability testing. In AAAI-93 [8], pages 46–51.

[435] ISHWAR KRISHNAN SETHI. Entropy nets: From decision trees to neural networks. *Proceedings of the IEEE*, **78**(10), October 1990.

[436] Ishwar Krishnan Sethi and B. Chatterjee. Efficient decision tree design for discrete variable pattern recognition problems. *Pattern Recognition*, **9**:197–206, 1977.

[437] Ishwar Krishnan Sethi and G.P.R. Sarvarayudu. Hierarchical classifier design using mutual information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-4**(4):441–445, July 1982.

[438] Ishwar Krishnan Sethi and J. H. Yoo. Design of multicategory, multifeature split decision trees using perceptron learning. *Pattern Recognition*, **27**(7):939–947, 1994.

[439] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, **27**:379–423,623–656, 1948.

[440] Jude W. Shavlik, R. J. Mooney, and G. G. Towell. Symbolic and neural learning algorithms: An empirical comparison. *Machine Learning*, **6**(2):111–144, 1991.

[441] Sheldon B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, **C-27**(6):509–516, June 1978.

[442] S. Shimozono, A. Shinohara, T. Shinohara, S. Miyano, S. Kuhara, and S. Arikawa. Knowledge acquisition from amino acid sequences by machine learning system BONSAI. *Transactions of the Information Processing Society of Japan*, **35**(10):2009–2018, October 1994.

[443] Seymour Shlien. Multiple binary decision tree classifiers. *Pattern Recognition*, **23**(7):757–763, 1990.

[444] Seymour Shlien. Nonparametric classification using matched binary decision trees. *Pattern Recognition Letters*, **13**(2):83–88, February 1992.

[445] W. Siedlecki and J. Skalansky. On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, **2**(2):197–220, 1988.

[446] V.G. Sigiletto, S.P. Wing, L.V. Hutton, and K.B. Baker. Classification of radar returns from the ionosphere using neural networks. In *Johns Hopkins APL Technical Digest*, pages 262–266, 1989.

[447] Herbert Simon. Artificial intelligence as an experimental science. In AAAI-93 [8], page 853. Invited Talk.

[448] J.A. Sirat and J.-P. Nadal. Neural trees: A new tool for classification. *Network: Computation in Neural Systems*, **1**(4):423–438, October 1990.

[449] Jack Sklansky and Leo Michelotti. Locally trained piecewise linear classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-2**(2):101–111, March 1980.

[450] Jack Sklansky and Gustav Nicholas Wassel. *Pattern classifiers and trainable machines*. Springer-Verlag, New York, 1981.

[451] John Smith. Public key cryptography. *Byte*, January 1983.

[452] J.W. Smith, J.E. Everhart, W.E. Dickson, W.C Knowler, and R.S. Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Symposium on Computer Applications and Medical Care*, pages 261–265. IEEE Computer Society Press, 1988.

[453] Padhraic Smyth, Alex Gray, and Usama M. Fayyad. Retrofitting decision tree classifiers using kernel density estimation. In ML-95 [333]. to appear.

[454] J. A. Sonquist, E. L. Baker, and J. N. Morgan. *Searching for Structure*. Institute for Social Research, University of Michigan, Ann Arbor, MI, 1971.

[455] S.P.Brooks and B.J.T.Morgan. Automatic starting point selection for function optimization. *Statistics and Computing*, **4**:173–177, 1994.

[456] Lilly Spirkovska. Three dimensional object recognition using similar triangles and decision trees. *Pattern Recognition*, **26**(5):727, May 1993.

[457] Sreejit Chakravarty. A characterization of binary decision diagrams. *IEEE Transactions on Computers*, **42**(2):129–137, February 1993.

[458] S.Schwartz, J. Wiles, I. Gough, and S. philips. Connectionist, rule-based and bayesian decision aids: An empirical comparison. In Hand [191], pages 264–278.

[459] B.S. Stewart, Ching-Fang Liaw, and C.C. White. A bibliography of heuristic search research through 1992. *IEEE Transactions on Systems, Man and Cybernetics*, **24**(2):268–293, 1994.

[460] L. Stewart. Hierarchical bayesian analysis using monte carlo integration: computing posterior distributions when there are many possible models. *The Statistician*, **36**:211–219, 1987.

[461] Quentin F. Stout and Bette L. Warren. Tree rebalancing in optimal time and space. *Communications of the ACM*, **29**(9):902–908, September 1986.

[462] C. Y. Suen and Qing Ren Wang. ISOETRP – an interactive clustering algorithm with new objectives. *Pattern Recognition*, **17**:211–219, 1984.

[463] Xiaorong Sun, Yuping Qiu, and Louis Anthony Cox. A hill-climbing approach to construct near-optimal decision trees. In AI&Statistics-95 [5], pages 513–519.

[464] P. Swain and H. Hauska. The decision tree classifier design and potential. *IEEE Transactions on Geoscience and Electronics*, **GE-15**:142–147, 1977.

[465] JAN L. TALMON. A multiclass nonparametric partitioning algorithm. *Pattern Recognition Letters*, **4**:31–38, 1986.

[466] JAN L. TALMON. A multiclass nonparametric partitioning algorithm. In Gelsema and Kanal [163].

[467] JAN L. TALMON, WILLEM R. M. DASSEN, AND VINCENT KARTHAUS. Neural nets and classification trees: A comparison in the domain of ECG analysis. In Gelsema and Kanal [164], pages 415–423.

[468] JAN L. TALMON AND P. MCNAIR. The effect of noise and biases on the performance of machine learning algorithms. *International Journal of Bio-Medical Computing*, **31**(1):45–57, July 1992.

[469] MING TAN. Cost-sensitive learning of classification knowledge and its applications in robotics. *Machine Learning*, **13**:7–33, 1993.

[470] PAUL C. TAYLOR AND BERNARD W. SILVERMAN. Block diagrams and splitting criteria for classification trees. *Statistics and Computing*, **3**(4):147–161, December 1993.

[471] SEBASTIAN THRUN AND ET AL. The monk's problems: A performance comparison of different learning algorithms. Technical Report CMU-CS-91-197, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1991.

[472] R. TODESHINI AND E. MARENGO. Linear discriminant classification tree: a user-driven multicriteria classification method. *Chemometrics and Intelligent Laboratory Systems*, **16**:25–35, 1992.

[473] J.T. TOU AND R.C. GONZALEZ. *Pattern Recognition Principles.* Addison Wesley, Reading, MA, 1974.

[474] GODFRIED T. TOUSSAINT. Bibliography on estimation of misclassification. *IEEE Transactions on Information Theory*, **20**(4):472–479, July 1974.

[475] CHARALAMBOS TSATSARAKIS AND D. SLEEMAN. Supporting preprocessing and post-processing for machine learning algorithms: A workbench for ID3. *Knowledge Acquisition*, **5**(4):367–383, December 1993.

[476] PEI-LEI TU AND JEN-YAO CHUNG. A new decision-tree classification algorithm for machine learning. In *Proceedings of the IEEE International Conference on Tools with AI*, pages 370–377, Arlington, Virginia, November 1992.

[477] I. B. TURKSEN AND H. ZHAO. An equivalence between inductive learning and pseudo-Boolean logic simplification: a rule generation and reduction scheme. *IEEE Transactions on Systems, Man and Cybernetics*, **23**(3):907–917, May-June 1993.

[478] PETER D. TURNEY. Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, **2**:369–409, March 1995.

[479] PAUL E. UTGOFF. Incremental induction of decision trees. *Machine Learning*, **4**:161–186, 1989.

[480] PAUL E. UTGOFF. Perceptron trees: A case study in hybrid concept representations. *Connection Science*, **1**(4):377–391, 1989.

[481] PAUL E. UTGOFF. An improved algorithm for incremental induction of decision trees. In ML-94 [331], pages 318–325. Editors: William W. Cohen and Haym Hirsh.

[482] PAUL E. UTGOFF AND CARLA E. BRODLEY. An incremental method for finding multivariate splits for decision trees. In *Proceedings of the Seventh International Conference on Machine Learning*, pages 58–65, Los Altos, CA, 1990. Morgan Kaufmann.

[483] PAUL E. UTGOFF AND CARLA E. BRODLEY. Linear machine decision trees. Technical Report 10, University of Massachusetts, Amherst MA, 1991.

[484] J.M. VAN CAMPENHOUT. *On the Problem of Measurement Selection.* PhD thesis, Stanford University, Dept. of Electrical Engineering, 1978.

[485] THIERRY VAN DE MERCKT. NFDT: A system that learns flexible concepts based on decision trees for numerical attributes. In *Proceedings of the Ninth International Workshop on Machine Learning*, pages 322–331, 1992.

[486] THIERRY VAN DE MERCKT. Decision trees in numerical attribute spaces. In IJCAI-93 [221], pages 1016–1021. Editor: Ruzena Bajcsy.

[487] P.J.M. VAN LAARHOVEN AND AARTS. E.H.L. *Simulated Annealing: Theory and Applications.* Reidel, Dordrecht, 1987.

[488] P.K. VARSHNEY, C.R.P. HARTMANN, AND J.M. DE FARIA JR. Applications of information theory to sequential fault diagnosis. *IEEE Transactions on Computers*, **C-31**(2):164–170, 1982.

[489] WALTER VAN DE VELDE. Incremental induction of topologically minimal trees. In Bruce W. Porter and Ray J. Mooney, editors, *Proceedings of the Seventh International Conference on Machine Learning*, pages 66–74, Austin, Texas, 1990.

[490] C. S. WALLACE AND D. M. BOULTON. An information measure for classification. *Computer Journal*, **11**:185–194, 1968.

[491] C. S. WALLACE AND J. D. PATRICK. Coding decision trees. *Machine Learning*, **11**(1):7–22, April 1993.

[492] Qing Ren Wang. *Decision Tree Approach to Pattern Recognition Problems on a Large Character Set.* PhD thesis, Concordia University, Canada, 1984.

[493] Qing Ren Wang and C. Y. Suen. Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6**:406–417, 1984.

[494] Qing Ren Wang and Ching Y. Suen. Large tree classifier with heuristic search and global training. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-9**(1):91–102, January 1987.

[495] Gustav Nicholas Wassel. *Training a Linear Classifier to Optimize the Error Probability.* PhD thesis, University of California, Irvine, 1972.

[496] Gustav Nicholas Wassel and Jack Sklansky. Training a one-dimensional classifier to minimize the probability of error. *IEEE Transactions on Systems, Man and Cybernetics*, **SMC-2**:533–541, September 1972.

[497] Larry Watanabe and Larry Rendell. Learning structural decision trees from examples. In IJCAI-91 [220], pages 770–776. Editors: John Mylopoulos and Ray Reiter.

[498] S. Watanabe. Pattern recognition as a quest for minimum entropy. *Pattern Recognition*, **13**:381–387, 1981.

[499] Nicholas Weir, S. Djorgovski, and Usama M. Fayyad. Initial galaxy counts from digitized POSS-II. *The Astronomical Journal*, **110**(1):1, 1995.

[500] Nicholas Weir, Usama M. Fayyad, and S. Djorgovski. Automated star/galaxy classification for digitized POSS-II. *The Astronomical Journal*, **109**(6):2401, 1995.

[501] S. Weiss and I. Kapouleas. An empirical comparison of pattern recognition, neural nets, and machine learning classification methods. In IJCAI-89 [219], pages 781–787. Editor: N. S. Sridharan.

[502] Shalom M. Weiss and Nitin Indurkhya. Rule-based regression. In IJCAI-93 [221], pages 1072–1078. Editor: Ruzena Bajcsy.

[503] Allan P. White and Wei Zhang Liu. Technical note: Bias in information-based measures in decision tree induction. *Machine Learning*, **15**(3):321–329, June 1994.

[504] P.A.D. Wilks and M.J. English. Accurate segmentation of respiration waveforms from infants enabling identification and classification of irregular breathing patterns. *Medical Engineering and Physics*, **16**(1):19–23, January 1994.

[505] Rogier A. Windhorst, Barbara E. Franklin, and Lyman W. Neuschaefer. Removing cosmic ray hits from multi-orbit HST wide field camera images. *Proceedings of the Astronomical Society of Pacific*, **106**, July 1994.

[506] J. Wirth and J. Catlett. Experiments on the costs and benefits of windowing in ID3. In *Fifth International Conference on Machine Learning*, pages 87–99, Ann Arbor, Michigan, 1988. Morgan Kaufmann.

[507] D. Wolpert. On overfitting avoidance as bias. Technical Report SFI TR 92-03-5001, The Santa Fe Institute, 1992.

[508] K. S. Woods, C. C. Doss, K. W. Vowyer, J. L. Solka, C. E. Prieve, and W. P. Jr. Kegelmeyer. Comparative evaluation of pattern recognition techniques for detection of microcalcifications in mammography. *International Journal of Pattern Recognition and Artificial Intelligence*, **7**(6):1417–1436, December 1993.

[509] Chialin Wu, David Landgrebe, and Philip Swain. The decision tree approach to classification. Technical Report TR-EE-75-17, Laboratory for Applications of Remote Sensing, School of Engineering, Purdue University, West Lafayette, IN, May 1975.

[510] Mihalis Yannakakis. The analysis of local search problems and their heuristics. In *Lecture Notes in Computer Science*, volume 415, pages 298–311. Springer-Verlag, 1990.

[511] K. C. You and King-Sun Fu. An approach to the design of a linear binary tree classifier. In *Proceedings of the Third Symposium on Machine Processing of Remotely Sensed Data*, West Lafayette, IN, 1976. Purdue University.

[512] Y. Yuan and M. J. Shaw. Induction of fuzzy decision trees. *Fuzzy Sets and Systems*, **69**(2):125, 1995.

[513] C.T. Zahn. Graph theoretical methods for detecting and describing gestalt clusters. *IEEE Transactions on Computers*, **C–20**(1), January 1971.

[514] Wang Zhengou and Lin Yan. A new inductive learning algorithm: Separability-Based Inductive learning algorithm. *Acta Automatica Sinica*, **5**(3):267–270, 1993. Translated into *Chinese Journal of Automation.*

[515] Xiao Jia Zhou and Tharam S. Dillon. A statistical-heuristic feature selection criterion for decision tree induction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **PAMI-13**(8):834–841, August 1991.

[516] Seth Zimmerman. An optimal search procedure. *The American Mathematical Monthly*, **66**(8):690–693, March 1959.

Vita

Kolluru Venkata Sreerama Murthy was born in Bapatla, India on 2nd April 1967. He obtained a baccalaureate honours degree in computer science and engineering from Motilal Nehru Regional Engineering College, Allahabad in 1988. He then got a Masters degree in computer science and engineering from Indian Institute of Technology, Madras in 1990. After working briefly for Westinghouse Electric Corporation (Process Control Division, Pittsburgh, PA, USA) and National Center for Software Technology (Bombay, India), he joined the Johns Hopkins University in September 1991, in pursuit of a doctoral degree in computer science. He married Sudha in May, 1995.