ENHANCEMENTS TO THE DATA MINING PROCESS

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

By
George H. John
March 1997

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

Nils J. Nilsson
Professor, Computer Science
Principal Advisor

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

Jerome H. Friedman
Professor, Statistics

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

Pat Langley
Consulting Professor, Symbolic Systems

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

Usama M. Fayyad
Senior Researcher, Microsoft Corp.

Approved for the University Committee on Graduate Studies:

Abstract

Data mining is the emerging science and industry of applying modern statistical and computational technologies to the problem of finding useful patterns hidden within large databases. This thesis describes the data mining process and presents advances and novel methods for the six steps in the data mining process: extracting data from a database or data warehouse, cleaning the data, data engineering, algorithm engineering, data mining, and analyzing the results.

We show how the standard data extraction process can be improved by building a direct interface between a data-mining algorithm and a relational database management system. Next, in data cleaning, we show how automatically iterating through the data mining process can identify records that can be profitably ignored during data mining. For data engineering, we develop an automated way to iterate through the data mining process to choose the subset of attributes that yields the best estimated results. In algorithm engineering, a similar process is used to automatically set the parameters of a mining algorithm.

For the data mining algorithms, we study enhancements to classification tree induction methods and Bayesian methods. Our new flexible Bayes data-mining algorithm is fast, understandable, and more accurate than the standard Bayesian classifier in most situations. In classification tree induction we study various univariate splitting criteria and multivariate partitions.

The analysis of results is necessarily domain-dependent. In an example applying data mining to stock selection, we discuss a key requirement in real-world applications: using appropriate domain-dependent methods to evaluate the proposed solution.

To My Parents and Teachers

Acknowledgments

The past several years of my life have been directed largely by random utterances by people I didn't even know that well at the time. My life would be vastly different if I hadn't heard any one of these:

"Do you like Brahms?" – Clyde H. Pittman, Jr.

Clyde is an entrepreneur in Dallas, TX. We happened to be working out in the same weight room and he wanted to know if he could change the music to something a little more civilized. This question and my answer got us talking, and before the workout was over he was trying to recruit me to come work at his startup telecommunications company. Weeks later I had rejected my planned summer job doing research in a laboratory at a local medical school, and I was instead working for Clyde doing database programming and developing computer algorithms for optimizing a telecommunications backbone. Clyde gave me a lot of opportunities and allowed me to discover for myself that computer science could be just as interesting as natural science. He also taught me that it pays *much* better!

"Take another class or something!" – Vida Tigrani

Vida is now my wife, but at the time I was a freshman, she was a girlfriend, and I'd been hanging around her enough to become a nuisance. To keep myself out of her hair, I ended up taking a class on tutoring and then becoming a tutor at Stanford, which shortly afterwards led to a job as a teaching assistant in the Computer Science department.

"How would you feel about instructing a class this summer?" – Stuart Reges

I was a sophomore working for Stuart as a teaching assistant at the time, and actually instructing a class was beyond my wildest dreams. I ended up instructing four classes

at Stanford, and enjoyed the experience quite a bit.

"You know, why pay tuition?" – Nick Parlante

During that summer of teaching, Nick was a grad student with teaching experience and had been assigned to the class as sort of a teaching advisor. During one conversation with me and the other instructor, Nick happened to utter these five fateful words. Graduate students who have teaching or research jobs receive tuition remission as compensation. Two quarters later I was a master's student and never paid tuition again. If Nick hadn't mentioned this to me, I could not have afforded to spend a fourth year at Stanford, and would never have had the chance to do research with Nils Nilsson.

"Do you need me to write you a letter of recommendation?" – Nils Nilsson

During my fourth year at Stanford, I began doing some research with Nils. He was something of an idol of mine, having been chair of the department when I was an undergraduate, and famous for his research in Artificial Intelligence. Technical books would have chapters about his work, and popular books would feature interviews with him delivering his opinions on the state of the art and the future. It was difficult to reconcile the image of the superstar researcher with the nice guy who genuinely cared about my well-being. After working with him for a few months I was still too awed and shy to ask him to write me a letter of recommendation for the Ph.D. programs to which I was applying. Luckily, he offered, and my Ph.D. career at Stanford began several months later, with Nils as my advisor.

I can imagine my parents somewhere off in the distance reading this and doubting whether *all* of the good things that have happened to me during my life have been random unplanned events. My stepfather, Jerry Walker, somehow seemed to think it was only natural to bring a huge, forty-pound teletype machine to his fiancee's house so her eight-year-old son could learn computer programming by logging into the local university's computer through his company's account. My mother, Patsy Walker, did a great job as teacher and manager of my educational career, as well as the other roles that comprise motherhood.

It really does take a village to raise a Ph.D. student, and I would like to thank many other people who have made it possible and enjoyable. In my life as a student

(and it has been a long one) I have had more than my fair share of amazing teachers. I would like to give special thanks to my teachers at Cistercian Preparatory School in Irving, TX who could not have been more stimulating or entertaining.

At Stanford, Ronny Kohavi and I spent a lot of time together doing research, reading papers, writing papers, and going to conferences. I really enjoyed the work we did together and benefited greatly from his energy and drive. Mehran Sahami and I have spent eight years at Stanford as undergrads and graduate students. We've worked together quite a bit, both taking and teaching classes together. Mehran's friendliness, humor, and brains have been much appreciated at various times over the eight years, particularly when he served as best man at my wedding, and when we worked together on the CS143 compilers project. Looking back, it's hard to say which one I appreciate more.

I've shared my Stanford office with Scott Benson for four years. We've helped each other out a lot—Scott cheerfully suffered through first drafts of many papers of mine, and always kept my LaTeX book safe and sound on his desk. Speaking of writing (he would have killed me if I hadn't made some effort at a transition), Pat Langley has been a great colleague and advisor, often supplying ideas and motivation when they were much-needed, and pushing me to improve my papers and rewrite them one or two (or three) more times.

Prof. Yoav Shoham introduced me to research when he let me, a lowly undergraduate, work on his agent-oriented programming project with Mark Torrance. Mark and I started the quarter with an agent that could do nothing but say "Yo, dudes" now and then, and we finished the quarter with animated (due to Mark) helicopters flying around a screen, making commitments to deliver supplies or pick up troops. Other remarkable teachers at Stanford were David Stork, Wray Buntine and Peter Cheeseman, who deserve credit for broadening my horizons. David taught me a lot about neural networks and pattern recognition in his class, and Wray and Peter have helped me very much by introducing me to relevant work and people in statistics. Wray has also read through a number of drafts of my papers and provided valuable feedback.

David and Peter both told me about Jerry Friedman in Stanford's statistics department, so I showed up at his office one day to talk to him and he's been continually helping me and other computer science Ph.D. students with our research ever since. Since many of the issues we've addressed in our research are primarily statistical, Jerry's advice and pointers to related statistical research have been invaluable.

Usama Fayyad gave me the first clear indication that my work had real merit when he offered me a summer job in his group at JPL. Usama played a significant role in my and many other people's careers by helping to popularize data mining. Unfortunately, Usama and JPL were hundreds of miles away, but luckily a like-minded soul was just around the corner from Stanford. For three summers in a row, I have worked with Evangelos Simoudis, first at Lockheed, then at IBM the last two years. Because of him, I've learned a lot about the business world, and how the technologies that I've been working on in my thesis can be applied in the real world.

The National Science Foundation has my appreciation for its graduate research fellowship program, which supported my work at Stanford for three years.

I'll close with a quote from Nils, spoken at the end of a long day of a conference on artificial intelligence, offering some perspective on the long-term goals of our research:

"The point of all this is to be able to spend more time at the beach!"

Contents

List of Tables

List of Figures

Chapter 1

What is Data Mining?

Data mining is a new name for the old process of finding patterns in data. Typically, a data mining analyst is presented with a problem to solve and an existing database that is relevant to the problem, and the analyst needs to find patterns in the data that can be used to solve the problem. For example, a company may be troubled by high attrition among its customers. By analyzing data on customers who leave and customers who stay with the company, a data mining algorithm might uncover patterns that can be used to predict which customers are likely to leave in the future, so that these customers can be enticed to stay. The patterns should be presented in such a way as to be interpretable by the company, so that in the future they can avoid practices that cause customers to leave. In this example and in scores of other applications, the key ideas are to identify a problem to be solved, analyze a relevant database to discover pertinent knowledge, and use that knowledge to solve the problem.

The name "data mining" has been a godsend. Before it became popular, so many different academic fields were working on the same technologies that it was extremely cumbersome to be inclusive when identifying the discipline. Researchers in areas such as statistics, machine learning, databases, neural networks, pattern recognition, econometrics, and many others were all working on the same kinds of problems. But they were not fully capitalizing on each other's work. Without a name under which to unite, the research suffered from fragmentation. Additionally, the research community (with the exception of some people in neural networks) was generally doing a poor job of getting the rest of the world educated and excited about their technologies. With no single name to excite potential users of the technology outside academia, their efforts in applying their research to real problems never inspired widespread interest.

Data mining unites all of these disciplines together under the premise that there exists much valuable knowledge in databases, but that due to the tremendous and growing volumes of data involved, advanced computer algorithms running on fast hardware are a more economical way to discover this knowledge than tedious manual searches and queries by human analysts. The need for data mining has been fueled in the business community by the popularity of *data warehousing*, which refers to the collection of large databases for analysis and decision support, as opposed to transactional databases which serve accounting and inventory managers. As the amount of data collected has grown, so too has the need for more tools to efficiently analyze the data.

According to some estimates, the amount of data in the world is doubling every twenty months (Frawley, Pietetsky-Shapiro & Matheus 1992). Knowledge may be power, but all this data is just unwieldy. The commercial need for tools to handle the data glut has risen sharply in the last year, creating a large community of users who need tools to mine these database for patterns that can be used to improve a business process, analyze a scientific experiment, help doctors better understand the effects of a treatment, and so forth.

In this chapter, we define the terms *data mining* and *pattern*, and we describe the data mining process, which gives the structure for the rest of the dissertation. We then give some examples of different kinds of patterns that different data mining algorithms work to discover, and we conclude with a brief summary of the contributions of the thesis.

1.1 Definitions

The term *data mining* has somewhat different uses in academia and in the commercial marketplace. In academia, researchers typically refer to the entire process involving data mining as *knowledge discovery in databases*, or *KDD* (Fayyad, Piatetsky-Shapiro & Smyth 1995). They reserve the term *data mining* for the single step in the process that involves finding patterns in the data. However, in the marketplace a single buzzword is better than two, so the use of *data mining* typically refers to the entire process (Ross 1996, Bers 1996, Rosen 1996, Allen 1996, Greenfeld 1996, Connor 1996, Richman 1996, Edelstein 1996, Gessaroli 1995). We feel that this disparity between academia and industry is more likely to be resolved by academia adopting the industry's usage of the term than the reverse, thus we use only the term *data mining* in this dissertation. In addition to being more connotative of the process, it parallels the term *data warehousing*, which refers to the process of building large databases for data analysis.

Definition 1.1 **Data mining** *is the process of discovering advantageous patterns in data.*

We will restrict the form of the data to be a relational database table where each row, record, or observation describes an entity, and each column, field, or variable describes one attribute of that entity.

Others generally speak about discovering *knowledge* in databases rather than patterns, but we prefer not to use the term *knowledge*, as it comes with significant baggage from the philosophical and the artificial intelligence communities. Since statistics is the oldest and most well-developed field that relates to data mining, we will adopt statistical terminology for our definition of a pattern:

Definition 1.2 *A* **pattern** *is a parsimonious statement about a probability distribution.*

Definition 1.1 mentions three important characteristics, identified by the three words *advantageous*, *patterns*, and *data*. Our restriction on the form of the data is largely a mathematical consideration—many other paradigms of data storage could be used in a real system so long as they can be mapped to the relational table model. Many systems that currently mine text, webpages and other ill-structured databases do so by mapping the data into relational tables, using a variety of clever techniques

(Sahami, Hearst & Saund 1996). Definition 1.2 restricts a pattern to be a statement about a probability distribution, but again this should not prohibit applications to poorly-structured data. Note that the language in which the statement is made is unspecified, so a graph of a probability density also is a pattern, as is the statement "the average of attribute three is 4.2." Finally, whether a pattern is advantageous is in the eye of the user of the system. Researchers in data mining have cleverly failed to completely automate away the human in the process, so that they and their students can all have jobs for the foreseeable future. An important role of the humans in the data mining process is to turn the resulting patterns into a plan of action, and to test this plan to be sure that it gives some advantage.

We will also use the term *model* to refer to a pattern or set of patterns. Just as a model airplane captures the salient characteristics of and relationships between the parts of a real airplane, so the patterns describe the salient characteristics of the entities represented in a database. A data mining model typically expresses the joint probability distribution over all variables as some functional form or sum of such forms, or expresses the conditional probability of one or a set of variables given the rest. In both cases we will speak about *learning*, *building*, or *inducing* a model from a database, by which we mean the process of discovering the pattern or model, and we will speak about *applying* the model to a new database, to estimate values for attributes whose values are missing.

Given these definitions, we can develop the example of finding patterns in the customer retention database mentioned earlier. Each attribute would be one piece of information we might know about a customer, such as how long they have been a customer (their "tenure"), and what kind of services they use. Each record would contain data on a single customer. Table 1.1 shows an example of a fragment from this hypothetical database.

One discovered pattern might be "customers who have been with the company less than a year have a 50% probability of attrition." Patterns might also be much more complex, taking into account several variables and specifying the conditional probability of attrition for a large and possibly exhaustive set of possible customer profiles. A plan of action might be to apply the model to the current customers and

Table 1.1: Example fragment of a customer retention database.

Name	Address	ZIP	Tenure	Services	...	Retained?
Jones	51 Elm	94305	3.25	A		Y
Davis	14 Thayer	02138	0.75	ACD		N
⋮						

determine which are most likely to leave, then design a campaign to retain these customers. We can estimate the payoff of such a strategy by assuming some effectiveness rate of the campaign and then determining the estimated payoff from a retention campaign based on the lapse model built by data mining. If the estimated payoff from a campaign were sufficiently high, a real campaign would be implemented, and its results would be measured.

1.2 The Data Mining Process

Data mining is an iterative process involving several steps, beginning with the understanding and definition of a problem and ending with the analysis of results and a strategy for using the results to gain advantage. There are usually two parties involved: the data mining analyst and the domain expert. Usually the problem to solve is well understood by the expert, and the data mining process begins with the expert attempting to explain the problem to the analyst and the analyst explaining the data mining process to the expert. Once they have together defined the problem to solve, in the remaining steps the analyst must collect relevant data if it does not already exist, clean the data, engineer the data to be maximally useful with the problem at hand, engineer a mining algorithm, run the mining algorithm, and explain the results to the expert (Figure 1.1). The process is iterative and has many cycles—the diagram becomes confusing if all possible arrows are inserted, but it can happen that the results of any step can indicate that some previous step ought to be redone in a different way.

The most authoritative descriptions of the data mining process are Fayyad et al. (1995), Fayyad, Piatetsky-Shapiro & Smyth (1996), and Simoudis (1996). Our

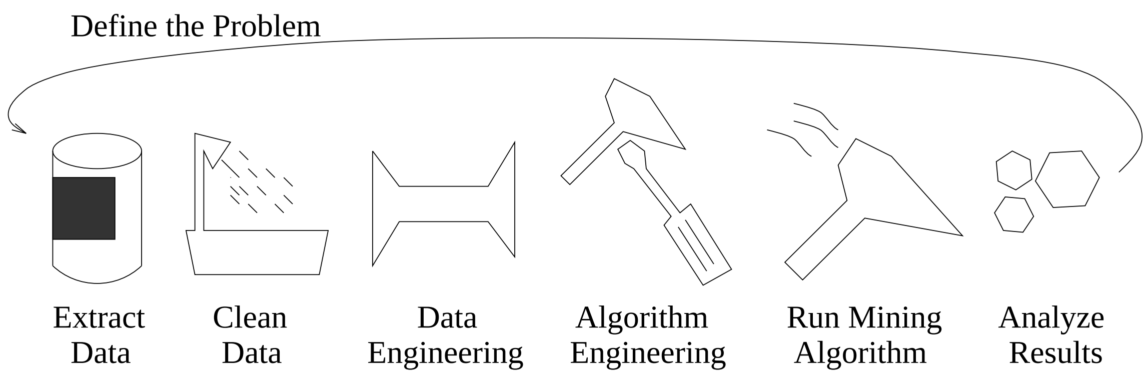

Figure 1.1: The data mining process.

description of the process differs in some ways, but the differences are more in organization and presentation than in content.

As we identify the steps in the process, we will discuss the actions that a data mining analyst performs, with several examples based on the customer retention problem. The data mining process is the subject of this dissertation, and the following chapters explore most steps in the process in further detail. Chapter 2 discusses extracting data from a database. Data cleaning is the subject of Chapter 3, and data engineering of Chapter 4. Chapter 5 explores a similar method for algorithm engineering. Chapters 6 through 8 describe new data mining algorithms, and Chapter 9 gives an example of the analysis of results in a stock selection application.

1.2.1 Understand and Define the Problem

It seems odd to even mention the step of understanding and defining the problem, but as a component of the whole process it can take a significant amount of time. Data mining differs from other analytical processes in the sense that is deceptively easy to apply a data mining algorithm to a database and get some result, but without a clear understanding of the problem the result may be worthless. It is important to include this first step as a reminder of the importance of understanding the problem clearly.

By *define the problem*, we mean that the data mining analyst should work with the expert to make the problem specific enough to be solvable and for results to be measurable. As stated by the expert, the problem may be something that seems clear to him or her but could actually be solved in a number of different ways. Some solutions may be appropriate, but others might be completely useless. To continue working with the customer retention example, the problem may have been posed by the expert as "reduce customer attrition." There are many kinds of patterns we could look for to solve this problem. As a very simple example of the ambiguity, consider that we could build a model that predicts, given current information about a customer, whether they will leave the company within a week. Even if we could build a perfect model to predict this behavior, it may be useless if it actually takes 1.5 weeks for the company to process the resulting list of likely-lapsers and mail them an incentive offer to stay with the company. Patterns often involve variables collected at different times, and so the precise temporal relationship is one important factor to consider. As we work through the example, we will assume a one month lead time was preferable.

1.2.2 Data Collection and Extraction

Once a problem has been defined, relevant data must be collected. In most cases, the relevant data is extracted from an existing operational database, or a data warehouse that was originally created to serve a variety of analytical needs. Usually, data mining algorithms cannot be run directly against a database with multiple tables. Thus, the data must be extracted from a relational database and stored in some format that can be accessed by the mining algorithms. In these cases, the manual extraction step is really just an artifact of the inability of the mining algorithm to run directly against a database. In Chapter 2, we show how SQL, the standard database query language, can be used by a data mining algorithm to directly and automatically get the answers to those questions about the data that it needs to ask. The data mining process can be sped up significantly by relying on the relational database management system's query engine, since when mining large databases (in the several hundred gigabyte to terabyte range), the majority of the mining algorithm's time is spent accessing the database.

In some cases, the optimal strategy is to collect an entirely new database, but this can be cost-prohibitive. Statistical experiment design (Mendenhall & Sincich 1988) speaks to the problem of how to collect data to test certain kinds of hypotheses, but scaling these methods up to the relatively complex patterns found in data mining is an open research question.

In the customer retention example, the company probably does not have a single database that looks like Table 1.1. For more concreteness, let us assume that the company in question is a provider of cellular telephone service, and we are the data mining analysts trying to help them reduce customer attrition. In the industry, the problem of customers repeatedly switching between service providers is known as "churning." Some service providers are already using sophisticated data mining methods to manage this problem (HNC 1994), but for illustrative purposes we will assume we are beginning a project with a company that has not previously used data mining methods to solve this problem. Rather than having a single database table like Table 1.1, the company probably has scores of tables with relevant information scattered among them.

Each division of the company collects the data that it needs to perform its job efficiently, but the result is that there is no single resource that we can use to get all the information about a particular customer. We have to build a data warehouse ourselves. Most data warehouses are themselves composed of many database tables, but since this is just an illustrative example, our data warehouse will be a single table. Additionally, most data warehouses are constructed to serve a wide variety of possible future analytical purposes. In our case we will only consider the customer retention problem previously posed. Thus, what we study is really closer in spirit to a *data mart*.

In the sample database of Table 1.1, for each customer we have their name, address, zip code, their tenure, the services to which they subscribe, and whether or not they stayed with the company. The name, address, and zip code are in a database table somewhere in the accounting department. Geography is usually important, because often people with similar lifestages (marital status, age) and lifestyles (income level, spending patterns) tend to live near each other. Third-party databases can map address and/or zip code to x-y coordinates, and these can be used in the model. Third-party databases can also map zip codes to demographic indicators of lifestage and lifestyle.

The tenure of a customer can be derived from the accounting database. We might also want to look at whether a customer typically pays his bills on time, particularly in the last several months. The accounting database will also contain itemized statements, from which we can tell which services the customer uses, how often they overrun their prepurchased time allotment, and whether they have been assessed late fees or other special charges. We might use the service database to see how often the customer has placed a call to report a problem, and we might use the network database to summarize the kind of equipment that is used to serve the customer. From the variety of data sources available, we would work with the domain expert, which in this case might be a customer relationship manager, to identify and define all potentially relevant attributes in data obtainable from the company's databases.

The preceding description of the data collection process is only illustrative. We have not precisely defined many of the attributes as we normally would in a real data mining project. For example, we decided to consider "whether a customer pays his bills on time, particularly in the last several months." We assume that paying a bill late may be an indicator of either the customer's financial distress, dissatisfaction with the service, or both. There are many ways to include this information in the database we are building. One option is to go back some fixed number of months (say 12) and create 12 attributes, each attribute's value being the number of days early or late the customer paid that month's bill. Or we may summarize the information and just report the number of months in which a bill came late. We assume that the more recent payment behavior will be most indicative of whether the customer will cancel their service within one month, so we might want to include a separate summary just for the preceding four months. These are all issues that are discussed and resolved between the data mining analyst and the expert.

When we are done, if we were to take this database table and travel backwards in time one month, it should be the case that the database contains current information on a subscriber, except that the "stay or leave" attribute would predict perfectly whether the subscriber would lapse at some point during the next month. If the universe is kind to us, then the behavior of customers one month ago and the behavior of customers today will be similar, as a function of the variables we can measure. So if we use data mining methods to build a model from the database we had one month ago, plus whether or not the customer lapsed during the month, we can apply it to the current data on our subscribers and predict which are most likely to lapse in the coming month.

There are really two parts to the data collection and extraction step. The first part we have described above is an intellectual exercise on the part of the data mining analyst and the expert to define the attributes of a customer that are likely to influence future attrition, and is a natural extension of the first step in the data mining process, understanding the problem. The second part involves actually writing code to process the existing databases and retrieve the information prescribed. This part is extremely tedious and dry, and to convey this I can hardly do better than to quote a piece of email from a colleague who was just starting a data mining consulting project for a company:

> The data analysis project is in the early stages, where someone gives us data, it's full of garbage, we ask for something better, and the process repeats. Just wasted most of today trying to reconstruct a data file that had commas inside comma-delimited fields. When I fixed that, the columns didn't match anyway, so I finally surrendered. Have you ever had someone give you data that you could actually just load and start analyzing?
>
> –*Randy Kerber*

When I asked Randy two months later if I could quote him in my thesis, he cheerfully responded, "No problem. In fact, I just got a new version of that data yesterday, and guess what? [The data file still had] company names, with commas, in a comma-delimited file!"

The end result of the collection/extraction and preparation phase is a database that is generally thought of as somewhat sacred, due to its origins as a direct descendant of operational databases or corporate data warehouse, and the (ideally) well-documented process that was used to create it from its ancestor databases. Usually the whole process is rather long and unpleasant. Analysts generally attempt to get every bit of relevant data they can, so that all further work can begin with this "mining base," and the extraction process will not have to be revisited.

1.2.3 Data Cleaning and Exploration

Once the relevant data has been collected, it is important to spend some time exploring the database, for two reasons. First, the data analyst must be intimate with the data—not just knowing the attribute names and what they are intended to mean, but the actual contents of the database. Second, there are many sources of error when collecting data from multiple databases into a single analytical database, and a good analyst must perform several sanity checks to validate the extracted data. In almost every problem we have worked on, there has been something wrong with the database collected for analysis. The three main problems have been mislabeling of fields in the database, data values with *special semantics*, and what we call *temporal infidelity* or *time travel*—cases where an attribute in the database contains information that could not be known at the time the model is intended to be applied, because the information only becomes known in the future.

For example, in the financial database studied in Chapter 9, one field was labeled "Return," but upon inspection we found that the values in that field were uncorrelated with the change in price of the stocks. After talking to several people, we found out that the fields had been mislabeled and it was actually the *next* field that contained the returns. Had we just run a data mining algorithm without first doing some sanity checking, we would have gotten completely useless results but would not have realized it. Thus, one kind of sanity checking is looking at the names of the fields, thinking about how they ought to be related, and checking to see that the expected relationship does hold.

Sometimes the data mining analyst will find that a relationship between attributes does hold in almost all cases, but there are a few records that do not support the expected pattern. In many cases this results from *special semantics* attached to certain data values. For example, a numerical value of zero or 99 is something to be feared, as they often are used to indicate that the actual value of the attribute is unknown. In some cases entire records have special semantics. In a database of consumer grocery purchasing behavior, there was a special customer ID that meant "total for the week." We discovered this while checking the extreme values in the data, and wondered about the eating habits of the family that was spending several million dollars on pickles every week. These records were removed from the database.

There are some more subtle pitfalls to avoid, such as the common error of temporal infidelity. A database of newspaper subscribers was to be analyzed to build a model predicting whether a subscriber would renew or lapse, given the various information the newspaper knew about the subscribers. To familiarize ourselves with the data, we looked at all of the conditional probability distributions of the class ("renew" or "lapse") given each other variable. A few of the other attributes were modestly informative, but one stood out: when the value of the "survey response" attribute was zero, over 90% of the time the customer had lapsed, and virtually none of the lapsed customers had a non-zero value for "survey." The company had given out this survey *after* many of the lapsed customers had already left the company, thus they had no opportunity to fill out the survey and thus the value was zero for "unknown." Thus the database suffered from temporal infidelity—the attribute to be predicted had in some cases been measured *before* one of the input attributes.

In another case of temporal infidelity, a database of frozen orange juice purchasers contained an attribute "Loyal or not?", which indicated whether a customer was likely to buy the same brand almost all of the time, or whether the customer bought several brands often. The idea was to use such a model so that a company could target their marketing towards customers who primarily purchased other brands but were not loyal to those brands. Surprisingly, the resulting model was right about 98% of the time. Now, human beings' orange juice purchasing behavior cannot possibly be so regular that any model could be this good. In fact, the person who had created the database and done some preliminary analysis had accidentally included attributes not only about the customer's demographics but also about their orange juice purchases, so that it became possible to "predict" the "Loyal or not?" attribute perfectly. The resulting model was useless because the input attributes included information that the company could not possibly know at the time they would make decisions about how to target their marketing. The general rule here is to do some preliminary data mining runs and check for unbelievably good results, and ensure that there is no temporal infidelity in the database.

1.2.4 Data Engineering

The previous steps have been concerned with creating and cleaning the *mining base*, which is a static database containing all of the information we would ever want to use in our data mining runs. There are three problems at this stage. First, the mining base might contain many attributes that could profitably be ignored. Selecting which subset of attributes to use is an important problem addressed in Chapter 4. Second, the mining base might contain many more records than can be analyzed during the available time, in which case we must *sample* the mining base. John & Langley (1996) discuss the desirable properties of a sampling method. Third, some of the information in the mining base might be profitably expressed in a different way for a particular analysis. As one explores different solutions to these problems in the course of a data mining project, the data engineering step is repeated many times to converge on the best *customized mining base* for our purposes, where customization addresses all three problems mentioned above.

In contrast to the mining base, which is usually created only once during the data extraction step, many customized mining bases are created during the data engineering step, to explore the use of different attributes, different sample sizes, and different precise definitions of the problem to be solved. For example, we could try to predict a lapse one, two, three, or four months in the future. We could try to predict a lapse using only the services consumed as predictors, or we could use only geographic location as a predictor, or we could use everything we have as a predictor. We could even augment the customized mining base with new attributes that are functions of the original mining base attributes; for example, if we have data on bill payment timeliness for the past twelve months, we might augment this with a new attribute that is the sum of the payment timeliness attributes in the last four months.

A variety of transformations may be applied to the data attributes. For example, if the data mining algorithm cannot handle missing values in the database (such algorithms are a dying breed), the missing values in an attribute may be imputed using the modal value or other more sophisticated methods using density estimation (Ghahramani & Jordan 1993). Other transformations ensure that all variables have roughly the same scale. One example is the z transform, which transforms the values of an attribute across all the records so that they have a mean of zero and standard deviation of one.

Other kinds of transformations are used to give better information to the data mining algorithm. For example, the services a customer uses may be represented as in Table 1.1 in the data warehouse and the mining base; that is, as a set of characters identifying which services they use. However, absent any help from the data mining analyst, there is no way for the algorithms to know that "A" and "AB" refer to two sets, both of which include "A". A solution would be to create one new attribute for each service, whose values would be binary, "yes" or "no".

We must also consider which records to use. In the cell phone attrition example, we might restrict ourselves to customers who had an active account for at least four months, to remove those customers who just joined the company under a teaser rate and left. Then, for customers whose accounts are currently active, we will put "Y" in the "Retained?" column and fill in the other columns with information from one month prior to the current date. For customers whose accounts are no longer active, we would restrict ourselves to those who just lapsed within the last month. For these customers, we will put "N" in the "Retained?" column and again fill in the other columns with information that was current one month prior to the current date.

We might also want to ignore records if we felt they were incorrect, or if they represented customers that were somehow odd and should not be used to infer general patterns. Chapter 3 describes one automated way to detect potentially incorrect values in the database, by making a pass through the data mining process and using the discovered model itself to identify suspicious records.

Considerable human intelligence is injected into the process during this step in the data mining process. Langley & Simon (1995) contend that much of the success of machine learning (and, by inheritance, data mining) is due to the problem formulation and representation engineering steps, which correspond to our problem definition and data engineering steps.

1.2.5 Algorithm Engineering

Often an "off-the-shelf" mining algorithm can be run on the customized mining base, but there remains the problem of selecting which algorithm to use and deciding specifically how to apply it. For example, we could use a classification tree algorithm on the customer retention problem. (Classification trees and other common data mining patterns are discussed in Section 1.3.) In this case the algorithm and the problem seem well matched: the problem is to predict the value of a single attribute (whose values are nominal, or categorical) and the inputs are a set of categorical and numeric attributes. Most classification tree algorithms are built to work in precisely this setting.

There are many such algorithms from which to chose, and this choice is likely to have a significant effect on the quality of the mined patterns. Even once a single classification tree algorithm is chosen, there are still several parameters to set. For example, the patterns discovered by the C4.5 classification tree algorithm (Quinlan 1993) can depend heavily on the parameter setting, and it is difficult to have any kind of intuition about what parameter setting will give the best results. Standard practice is to either ignore the fact that there are parameters that can be tuned, and just leave them set at their default values, or to run a few experiments with different parameter settings and see which perform the best. Both approaches explore only a miniscule portion of the whole parameter space. In Chapter 5 we describe an automated method for tuning the parameters of a mining algorithm that is similar to the attribute selection algorithm of Chapter 4. Both require some objective function that can estimate the quality of the discovered patterns, and use this objective function to perform an optimization in their parameter space.

In some other cases, the available data mining algorithms may not be directly applicable to the problem of interest, and so some work is required to adapt the algorithm to the problem at hand. The mismatch between the problem and algorithm is generally caused by having an inadequate set of data mining tools, or by attempting to apply data mining to new types of problems for which tailored algorithms have not yet been invented. As an example of the former, consider solving a classification problem with five classes using a data mining algorithm that can only handle two. A common solution is to build all five possible two-way classification models and then manually unify them into a single five-way classification model. Applying data mining to new types of poorly-structured data, such as web pages or text, is an example of the latter problem of mismatch between the problem and algorithm. As more tools are added to the data miner's toolbox, this problem will become less frequent, but as long as innovative uses for data mining are explored, the first phase of applications will likely involve extensive algorithm engineering.

1.2.6 Running the Data Mining Algorithm

The next step, running the data mining algorithm, is the point where the analyst and expert can just sit back and watch. Thousands of researchers and developers are busily developing robust algorithms that work well on a wide variety of problems, and this is the step in the process that leverages their work. Some refer to this single step in the process as *data mining*, while referring to the whole process as *knowledge discovery in databases* (Fayyad et al. 1995). Our contributions to the set of algorithms, all addressing classification problems, are described in Chapters 6 through 8. Such algorithms produce *predictive* patterns, which are generally evaluated by seeing how well they do their job—predicting the value of one or more attributes given the rest.

To do this evaluation properly, we must hold out some percentage of the records from the customized mining base, and we call this set the *validation set* or *holdout set*. The remainder of the records constitute the *training set* and are mined by the algorithm to produce a set of patterns. Modern data mining algorithms are designed to avoid discovering "patterns" that may exist in the training set but are in fact just accidental and do not hold in the validation set. Still, the performance of a model on the data from which it was built is usually better than its performance on an independent sample of data from the same distribution, so it is important to use an independent sample to evaluate the quality of the discovered patterns.

Keeping a fraction of the records in a holdout set is a sufficient way to get an unbiased estimate of the performance of a single model. But if we are building a set of models by trying different algorithms or different custom mining bases, and if we use the holdout set performance to choose the best among the set of models, then we have the problem of accurately estimating the performance of the best model. Since the holdout set was used to choose the best model, it no longer gives an unbiased estimate of the performance of the best model. What we really need is yet another independent sample of data, called the *test set*, that will only be used *once* to evaluate the final recommended model[1]. However, even the results on the test set should always be treated as an estimate of the performance of the model. The only true test is to decide how to actually use the model in practice, use it, and evaluate the actual results.

1.2.7 Preliminary Evaluation of Results

The methods used to evaluate predictive patterns depend on the problem being solved. In the retention example, we would like to rank customers based on their probability of lapsing within the next month, so that the company can somehow make an effort to retain them. Lift charts are a common way to compare how well a model ranks entities according to their propensity to take (or not take) some action. They are frequently used in customer retention, targeted marketing, and fraud detection. In all of these problems, the expert generally needs to contact some select subset of people (those likely to lapse, those likely to respond to an offer, or those likely to commit fraud). Contacting a person involves some cost, so the goal of the data mining analyst is to build a model that will rank the candidates in the proper order. Lift charts provide a way to evaluate this ranking, and are thus very common.

[1] Unfortunately, in some papers in the neural net community, the names *test* and *validation* have been confused.

To evaluate the retention model using a lift chart, we would apply it to the holdout data. Each record would receive a *score*, some number such that the lower the number, the greater the probability of attrition, and the higher the score, the greater the probability of retention. Since the holdout data was never used while building the model, performance on the holdout data should give a good estimate of the model's performance. However, there are a number of potential problems (*e.g.*, the temporal infidelity problem mentioned earlier) that can cause overly optimistic results on the holdout set, so it is important to remember that results on the holdout or test sets are just estimates.

Imagine that the company has decided on a standard letter to mail to customers with high attrition risk. The company might not know how many letters they can afford to mail, and would thus like a graph that shows, for varying number of letters sent out, the actual number of customers receiving the mailing who would have lapsed in the next month.

Figure 1.2 shows a gains chart, or lift chart, for a hypothetical retention model. In this example there are a total of 1000 customers and 50 will lapse during the next month. We assume that a scoring model has been applied to the customers and that they have been sorted in ascending order by score, so that the first 100 records would contain the 100 records with the lowest score and thus the highest estimated probability of attrition. If we were to mail an enticement to stay with the company to 100 customers, absent any other information, these would be the 100 to contact. On the graph, the point (x, y) indicates that among the x lowest-scoring records, y were actually customers who were going to lapse.

The ideal model scores the data so that $y = x$ for the first 50 records, and $y = 50$ thereafter. In a random model, since on average one in twenty customers will lapse, $y = x/20$. A data mining model's gains curve usually looks somewhat like the curve shown in Figure 1.2—not perfect, but much better than random guessing. A marketing manager would likely expect such output from a data mining system, and would use the output both to choose from among a set of models as well as decide how many mailings to send out.

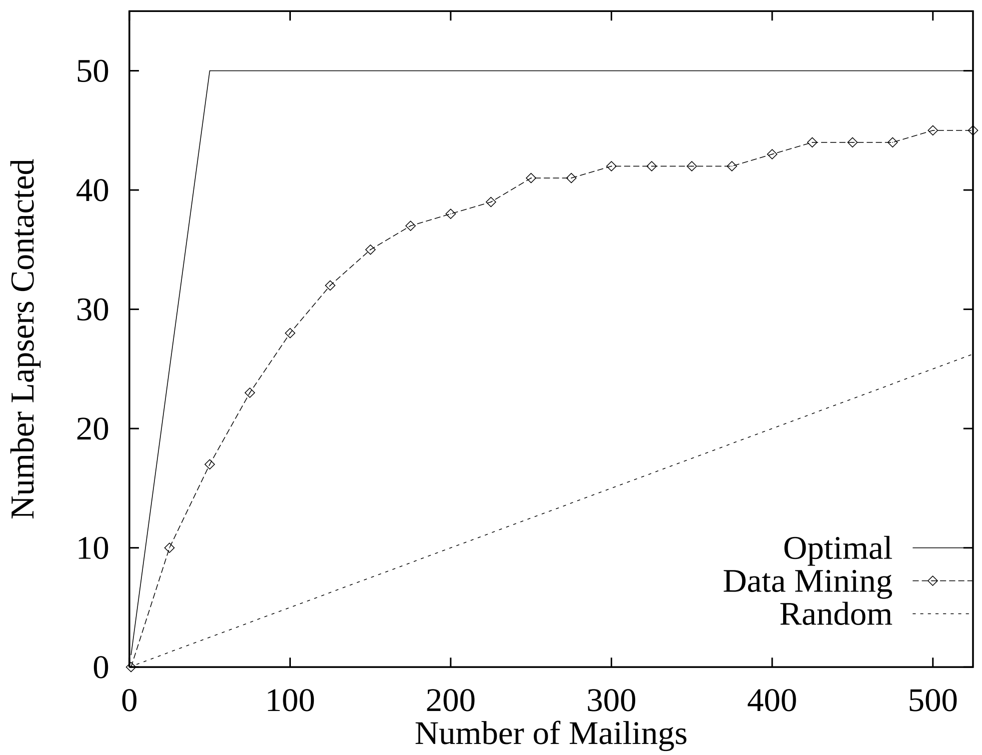

Figure 1.2: Example gains chart

The KS statistic (Kolmogorov-Smirnov, cf. Friedman 1977) is a way of summarizing the relationship between the gains curve for a new model and an existing process. Let $f_m(x)$ denote the y value of a gains curve for a new model, and let $f_e(x)$ denote the y value for the existing process, then $gain(x) = f_m(x) - f_e(x)$, and $KS_m = \max_x gain(x)$, *i.e.*, the difference between the two curves where the difference is greatest. This can be an unrealistic measure, since this difference could occur at some point in the graph which represents contacting more customers than the company could afford. Nevertheless, it is very convenient to summarize the graph with one number, so the KS statistic is quite popular. In Figure 1.2, the KS statistic between the data mining model and the random process is 28.5 and occurs at the point $x = 250$.

As we have mentioned, the proper evaluation of the patterns discovered by data mining algorithms depends on the problem being solved. Although lift charts are widely useful, they are of course not ideal in all situations. Chapter 9 describes the process of evaluating results in a stock selection application. Methods for analyzing stock portfolios are well-developed in the finance community, and it was necessary to use standard methods from finance to explain results to people in the financial community. For example, just reporting the accuracy of a model is nearly useless. It was important to simulate actual trading using the model and report measures of the resulting portfolios' risk and return.

As well as numerically evaluating the performance of the model, other less objective measures are very important. It is almost always necessary for the model to be interpretable by the domain expert. The expert must be comfortable with the model, and although performance results on the holdout set and gains charts do inspire some confidence, without some insight into what the model is doing, she is unlikely to recommend that the model should be used to control some process (*e.g.*, the cell phone retention mailings) in her organization. Usually, when the expert can interpret the model, one of three things happens: the expert is happy with the model but feels she already knew about the patterns it contains, the expert is happy with the model and is genuinely surprised by some of the patterns, or the expert is unhappy with the model. In the first case, where no previously unknown patterns were discovered, the model can still be useful for its quantitative combination of its constituent patterns into one model which may give better results. In the latter two cases, further analysis is needed, and the data mining problem may need to be refined.

1.2.8 Refining the Data and Problem

A frequent response to the results by the domain expert is "that's not quite what I wanted." This is the major source of iteration in the data mining process. For the first iteration, it can often be productive to make one quick and dirty pass through the entire process, possibly skipping or severely abridging the preparation and exploration phases. Showing such results to the expert is an excellent vehicle for communication—it is often quite hard to pin down details of the problem when the data miner has not played with the data or seen what kinds of patterns are to be found, and the expert has not seen what the results of data mining will look like.

Making a quick pass through the process for the purpose of more effective communication with the expert is a good strategy, but care must be taken when presenting results so that the expert understands the spirit in which they are to be taken. For example, the preparation and exploration steps can be the most time consuming and might be abridged during a preliminary pass through the data. The result is that we have results that really must be verified before being presented to the expert as actual patterns discovered in the data, and instead should just be presented as examples of the kind of patterns that will be discovered. As long as this is clear to both the expert and the data mining analyst, this can be an excellent way to be more productive and communicate better with the expert.

Sometimes the patterns discovered during a diligent pass through the data mining process can expose serious problems in the original definition of the data mining problem. For example, a European bank wanted IBM's data mining group to help it increase the response rate for a particular mailing that it was sending out to some of its customers, offering a personal line of credit at a fairly high interest rate. Brought in to "increase the response rate," the IBM team built a model that would score the customers by their propensity to respond to the mailing. Luckily, the model was interpretable, and it was clear that the group most likely to accept the offer was almost entirely composed of individuals with *negative* checking account balances! (In Europe, it is more common for banks to honor a check when insufficient funds exist in the customer's account—the customer's balance simply becomes negative, and they pay a high interest rate on the negative balance.) Thus, if the bank were to blindly increase their response rate to the loan solicitation, they would do so at their peril, exposing themselves to very high credit risk. Those people most likely to accept the loans are already in dire financial straits and are unlikely to be able to repay the loans. The data mining models served as a reminder that, clearly, the objective is not just to increase response rate but to increase profits for the bank, and this must be accomplished by offering the bank's products to customers most likely to respond to the offers, who will also be able to repay the loans.

In other cases, the expert may not be happy with the results, but there is nothing wrong with the definition of the problem to be solved. If results are not good enough, the expert may suggest new attributes to add during data engineering or more attributes to extract from other databases. Strange results can indicate problems in the data; for example, if some attributes have the opposite effect than the expert would expect, there might have been a problem during the data extraction step.

After a few refinement steps, if the performance of the model is sufficiently good and if the expert is comfortable with the discovered model, we are ready to put the results to use.

1.2.9 Using the Results

So far we have only talked about discovering patterns in data and the necessary steps that precede and follow, to prepare the data for analysis and to ensure that the discovered patterns are useful and valid. Once we have made a final pass through the data mining process, we are ready to implement some strategy based on the discovered patterns. Since the definition of a pattern is quite broad, there are many types of patterns and many ways of taking actions based upon them. In the next section we will review the most common types of data mining algorithms and patterns, and there we will briefly describe how the different types of patterns can be used. For now we will only discuss the customer retention example.

In the customer retention example, the attrition model might be applied to half of the company's current customers and the company might continue with their existing retention strategy on the rest of their customers. This way, the real results from the model and their existing strategy can be directly compared. In some cases, the model not only gives some score to each customer that predicts their probability of attrition, but the model also partitions the set of all customers into groups that have similar characteristics and similar probabilities of attrition.

An attrition model on the cellular phone customers might indicate that some of those customers likely to cancel their service fall into one of three groups: those who seldom use their phones and thus pay an extremely high amount per minute of use, those who have experienced excessive problems with sound quality or dropped calls, and those who have had large phone bills due to usage of many more minutes than their prepaid allotment. The campaign to retain these diverse groups should take their characteristics into account. The high-usage group might be mailed a letter offering them a chance to change their service plan so that they have more prepaid minutes for a lower overall per-minute rate than they are now paying. The poor-service group might be mailed a coupon for air time or accessories with an apology and a promise to take care of the problems.

Note that the domain expert is heavily involved in this decision—whether to offer the coupon at all and how large it should be, how to prioritize network upgrades and whether promises of better service in the near future can be made, and so forth. Other business issues in this example include profitability considerations alluded to above in model of responses to the line of credit solicitation. Even if we had a perfect model of attrition, the company may still not *want* to retain all of its customers. Some may have subscribed while the company was offering extremely low rates, and the company may now be providing service to them at a loss, or for various other reasons they might be undesirable as customers. The point is that just having a model can be a long way from being ready to take actions based on the patterns in the model. In the next section we will give examples of the main kinds of data mining patterns and the kinds of actions they suggest.

1.3 Types of Data Mining Patterns

Broadly speaking, there are two kinds of patterns discovered by data mining: predictive and informative.[2] Predictive patterns are built to solve a specific problem of predicting one or more attributes in a database from the rest, like the customer retention problem described above. Predictive patterns need not always be concerned with foretelling the future—the important characteristic is that they make an educated guess about the value of an unknown attribute given the values of other known

[2]In the machine learning community the distinction is drawn by separating the mining algorithms into two types referred to as *supervised* and *unsupervised*, which correspond to algorithms that find predictive and informative patterns, respectively.

attributes. By contrast, the point of informative patterns is not to solve a specific problem, but rather to present interesting patterns that a domain expert might not already know. Informative patterns are harder to evaluate than predictive patterns, because their real value lies in whether they suggest any actions to the domain expert, and how effective those actions are.

All data mining algorithms incorporate some measure of goodness or interestingness of a pattern, and they use this measure as they search for patterns to decide which to keep, which to discard, and which kinds of patterns to explore further. Predictive patterns can be evaluated in the obvious way—by judging how good they are at doing their job. Since they predict the value of an attribute, and that attribute exists in the training database, the common method for evaluating predictive patterns is to compare their predictions with the actual values of the output attributes in the training set. By calculating how often the patterns are wrong, and by how much, a data mining algorithm can evaluate the patterns. During the run of an algorithm, the informative patterns it explores cannot be evaluated by how well they do their job, since their job is to give insight to a human expert. Instead, mathematical criteria are used that will hopefully catch most of the potentially interesting patterns while discarding the uninteresting patterns. Below we describe the two most common types of informative patterns and the interestingness measures used to evaluate them, followed by the two common types of predictive patterns.

1.3.1 Association Rules

Association rules are informative patterns of the form $X \to Y$, where X and Y are statements about the values of attributes of a record. The canonical example is the analysis of market baskets at a grocery store. The store's database of purchases might have one record per shopping basket, and each record contains thousands of attributes that indicate whether or not certain products were purchased. Association rule discovery algorithms search the space of all possible rules $X \to Y$, where X and Y are sets of items. The most famous example of an association rule was discovered in a database of grocery purchases: "diapers $\to$ beer." This pattern apparently is caused by young fathers being sent off to the store to buy diapers, and deciding to reward themselves for their trouble.

The most common measures of interestingness in current association rule discovery algorithms (Agrawal, Imielinski & Swami 1993) are the support and confidence measures, although others are possible. Each association rule has an associated *support* and *confidence*, where support is the probability that a record satisfies both X and Y, and confidence is the probability that a record satisfies Y, given that it satisfies X. Association rules are *patterns* in the sense of Definition 1.2, because each rule with associated confidence and support values is a direct statement about the probabilities of the attributes involved in the rule.

Note that in market basket analysis, there is no way to know which customer is making a purchase. The store only knows which items were purchased all at once by the same shopper. Many retail chains are now marketing loyalty programs to their shoppers, who receive an identification card and, in return for using the card, receive some small discount on their purchases. With non-anonymous shoppers, a store can learn more from their database by mining for *sequential patterns* (Srikant & Agrawal 1996), which are association rules where X represents a set of items purchased during one trip to the store and Y represents items purchased on the same shopper's subsequent visit to the same store.

Since association rules are informative patterns, the process of analyzing and using the results is difficult. In the diapers and beer example, the store decided to move diapers and beer closer together and to put potato chips and other snack items in between. Another example would be that any company offering a number of products to its customers could use association rules for *cross-selling*, that is, selling new products to its existing customers. A large mutual fund company might discover several patterns of the form "85% of the people who own funds A and B also own fund C." An obvious idea is to market C to those 15% who do not already own it.

Association rules are the newest of the common types of data mining patterns, so they certainly have many potential applications that have not yet been explored. Although the discovered patterns are very simple, the value in their discovery is that there are staggeringly many possible association rules that apply to a given database, and filtering the interesting or informative patterns from the rest is quite hard to do efficiently.

1.3.2 Clustering

Another kind of informative pattern is a *clustering* model, also called a *segmentation* model. The goal of clustering is to take a set of entities represented as records in a database and to partition them into a number of groups or clusters, such that the entities within a cluster are similar. Clustering models take on different forms, but usually at some level the patterns exist mathematically as a function that assigns each record in the training set to some cluster number. Statistics on the properties of each cluster can help the domain expert understand the clustering model.

Demographic segmentation was highlighted during the 1996 U.S. Presidential Election campaign, as the pollsters identified "Soccer Moms" as the important swing voters the candidates needed to win. In this case, each record in the database probably contained information on a person eligible to vote in the 1996 presidential election, including demographic variables such as age, sex, occupation, and so forth. By segmenting the database, some homogeneous groups of voters were discovered, including one group that received the catchy label that garnered so much attention in the press. Clustering has also been used in scientific discovery. The AutoClass program developed by Cheeseman, Kelly, Self, Stutz, Taylor & Freeman (1988) was run on a NASA database of infrared spectra of stellar objects and discovered several new types of stars that had not been previously identified by astronomers.

Market segmentation is in widespread use by retail and consumer product companies, who need to learn more about who their customers are so that they can tailor their marketing or product development to their customers. Figure 1.3 shows a hypothetical example of a clustering model discovered from a database of sports car buyers. For each buyer we know their sex, age, and income, and we can read from the graph that there are three clusters of buyers: young buyers, upper-middle income male buyers of all ages up to 55, and midlife-crisis male buyers in their mid forties.

Clustering patterns also fit Definition 1.2. The statement a clustering pattern

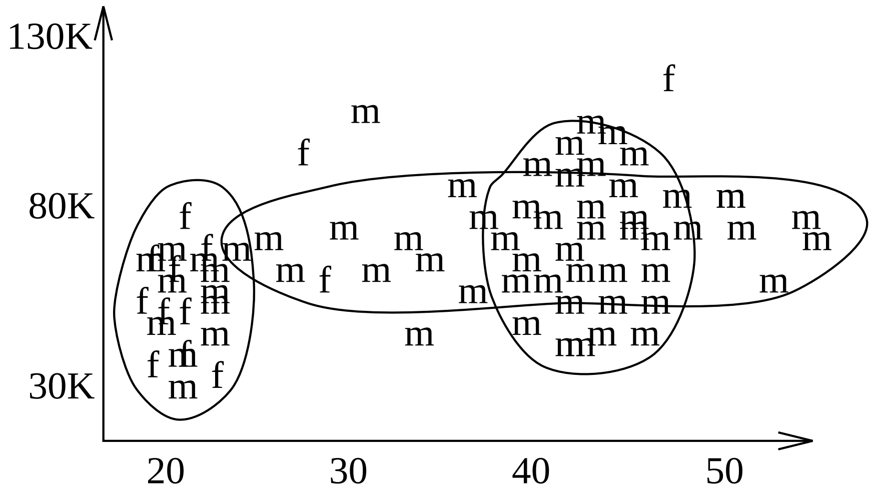

Figure 1.3: Hypothetical clustering of car buyers by age (horizontal axis), income (vertical axis), and sex.

makes about the probability distribution is that the distribution of the entire population can be decomposed into the sum of the distributions of the clusters. Formally, if $p_i(\mathbf{x})$ is the distribution for cluster i, $p(\mathbf{x})$ is the distribution for the entire population, and *cluster* is the new attribute indicating the cluster to which a record belongs, then the statement the clustering pattern makes about the probability distribution of $\mathbf{x}$ is:

$$p(\mathbf{x}) = \sum_i p(\mathbf{x}|cluster = i)p(cluster = i) \ .$$

The process of deciding how to use clustering patterns is carried out by the domain expert and data mining analyst, who work together to understand each cluster. Assigning a memorable name for each cluster can help to remind the users of their properties. The Claritas corporation sells its own proprietary demographic clustering models with demographic segment names ranging from “shotguns and pickup trucks” to “MD’s and PhD’s.” In the car-buying example, the car manufacturer may use the clusters to decide how to advertise the car. This involves both creating marketing materials targeted to the three groups and using the appropriate channels to target the desired demographics. Again, since clustering is an informative pattern, the value of a clustering model lies only in the insight it provides to the domain expert and

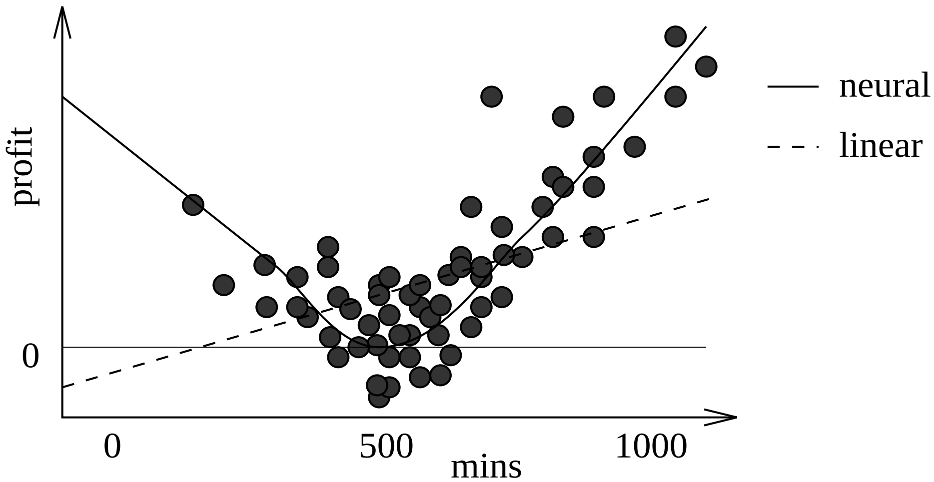

Figure 1.4: Regression of profit on number of minutes used.

the actions the expert takes based on this insight. In the next two sections we will describe predictive patterns, which are easier to evaluate.

1.3.3 Regression

Regression (a term inherited from statistics) refers to discovering predictive patterns where the attribute being predicted is real-valued. Linear regression, where the predicted attribute is modeled as a simple linear function of the input attributes, is widely studied in statistics (Weisberg 1985). After linear regression, neural networks are probably the most commonly-used regression technology in data mining.

A regression model might be used by our cellular phone company to help with customer retention. In addition to knowing which customers are likely to leave the company, it would be helpful to know the value of retaining a customer—some estimate of the profits the company is likely to derive from that customer in the future. If the value of retaining a customer is actually negative then there is no point in making an effort to retain the customer, and in fact the company would benefit by taking measures to encourage the customer to leave.

Consider the problem of predicting profit from a customer using only their number of minutes per month on the phone. Assume we are building the model only on the set of customers who subscribed to a plan that included 500 prepaid minutes per month.

The graph of profit might look something like Figure 1.4. If the customers use less than their prepaid 500 minutes, the phone company profits, and if they use more, the phone company again profits. Their losses only come when the customers use almost exactly the number of minutes they have prepaid. A linear regression would produce a model something like the dashed line in the figure. Since the real pattern in the data is not linear, the model makes many large errors. A neural network might discover a pattern like the solid line in the figure, which fits the data well.

Both linear regression and neural networks are patterns in the sense of Definition 1.2, because their prediction at any point can be viewed as their estimate of $E(y|\mathbf{x})$, the expected value of y, the attribute being predicted, given $\mathbf{x}$, the vector of input attributes.

Predictive patterns have value not only in how well they predict the desired unknown quantity, but also in their ability to convey the interesting patterns to the domain expert, so they are informative in a sense as well as predictive. This is often the basis for choice of which data mining algorithm to use. Even when the predictive ability of one method is worse than another's, the first method may be preferable if it is more easily interpreted by the domain expert.

1.3.4 Classification

A classification pattern is similar to a regression pattern, except that it predicts the value of a nominal or categorical attribute instead of a real-valued attribute. The predicted attribute is called the *class*. Many of the chapters in this dissertation are either explicitly about classification, or they use classification as an example of how the techniques they describe can be applied.

Two common types of classification patterns used in data mining are classification trees (also called decision trees) and the simple Bayesian classifier (also called the naive Bayesian classifier). A classification tree is a series of questions, where each question asked depends on the answers to the previous questions, and the questions terminate in a prediction of the class. Trees can also be used for regression (Breiman, Friedman, Olshen & Stone 1984), where each leaf predicts a constant real value rather

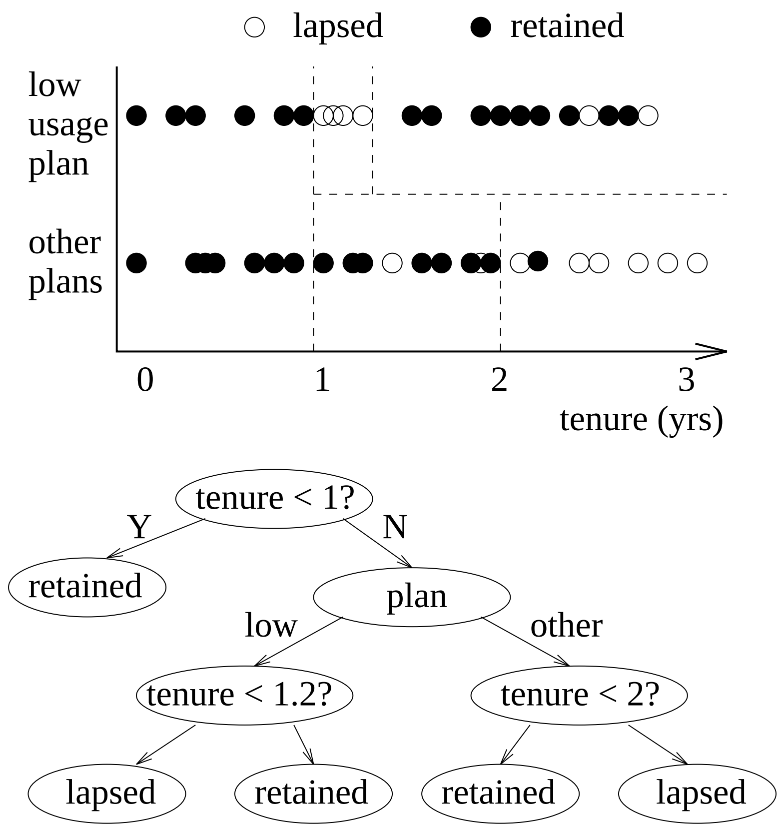

Figure 1.5: Example classification tree pattern.

than a class, but they are most commonly used for classification. Standard trees are *univariate*, meaning each question tests only the value of a single input attribute. Thus, a question might be "Is the value of attribute x_4 greater than 5.6?" or "Is the value of attribute x_7 in the set {yellow, red, green} or the set {purple, orange, white}?" The questions asked and the possible answers and subsequent questions form a tree, where the first question is the root and each node has k children, where k is the number of possible answers to the question it asks, or zero children if it is a leaf, in which case it makes a prediction of the class.

Returning to the problem of a company retaining its cellular phone subscribers, imagine that the class concerns whether or not a customer stayed with the company, and the input attributes are the tenure of the customer and whether or not their service type is the "low usage" plan (with lowest cost and lowest number of prepaid minutes). Figure 1.5 shows a hypothetical database of subscribers and the classification tree that might be discovered, where the boundaries between the leaves of the tree are represented as dashed lines in the plot of the data. The main advantage of the tree model is that it can be directly interpreted. Each leaf of the tree contains a number of records that are similar at least in terms of the attributes tested in the tree. When dealing with retention, one common way to use a classification tree is to investigate each leaf representing lapsed customers and determine how best to convince those customers to stay. Different leaves may represent customers leaving for different reasons, so the enticements to stay are best customized for each type of customer.

Classification models are also patterns in the sense of Definition 1.2. If y is the class the model predicts when shown input $\mathbf{x}$, then the model is making the statement that $p(Y = y|\mathbf{X} = \mathbf{x}) > p(Y = y'|\mathbf{X} = \mathbf{x})$, for all other classes y'. In the dissertation we also discuss the simple Bayesian classifier, which estimates $p(Y|\mathbf{X})$ directly. We will describe the simple Bayes and classification tree algorithms and models more thoroughly in Chapter 2.

With some types of patterns, simple Bayes for example, rather than just predicting a class value, the model gives a *score* that indicates how likely it is that a new record is a member of each class. Records can then be ranked by the score, as in the customer retention example above where likely lapsers were prioritized by their scores, and a gains chart was used to evaluate the model (Figure 1.2).

1.4 Summary of Contributions

The main contributions of this thesis, in order of presentation, are:

- Chapter 2, the strong SQL interface between a data mining algorithm and database
- Chapter 3, the ESTEEM method for data cleaning
- Chapter 4, the wrapper method for data engineering

- Chapter 5, the wrapper method for algorithm engineering
- Chapter 6, the flexible Bayes algorithm for classification, and theoretical results of the pointwise strong consistency of simple Bayes and flexible Bayes
- Chapter 7, a large-scale comparison of splitting criteria for classification trees
- Chapter 8, the soft entropy criterion for linear discriminant trees
- Chapter 9, detailed simulation results of an application of data mining to stock selection

Chapters 2 through 5 constitute a segment that shares the common theme of automating steps of the data mining process (data extraction, data cleaning, data engineering, algorithm engineering) by considering the context in which the step is taken. For example, the set of optimal attributes to use in a mining problem depends on the data mining algorithm to be used later. Thus decisions about which attribute to use should somehow take into account the mining algorithm being used. Similarly, the data that needs to be extracted from a database depends on the algorithm to be used. For example, the simple Bayesian classifier uses only bivariate statistics of the database, so it only needs simple summary statistics. These ideas essentially reflect a realization that the data mining process is indeed iterative, and they provide precise methods that specify how to automatically iterate through parts of the process.

Chapters 6 through 8 share a common theme of breaking down data mining algorithms into their constituent parts, and experimenting with replacing a part with one or more different methods. For example, Chapter 6 looks at the simple Bayesian classifier, specifically the part that requires estimation of the conditional density of a continuous variable given the class using a Gaussian density function, and considers replacing this with a more flexible density estimation method.

The analysis of results is necessarily domain dependent. Chapter 9 gives an example of the kind of analysis necessary to do a thorough evaluation of data mining results in a stock selection application.

Collaborations

Several of these chapters grew out of joint work with colleagues at Stanford and elsewhere. Chapter 2 began its life when I heard Michael Kearns of AT&T Bell Labs give an invited talk at Stanford in 1994 on his proposed statistical query model for computational learning theory, which proposed learning from statistical queries of a database rather than directly from data. The chapter had help from Brian Lent, a Ph.D. student in the databases group at Stanford, who supplied references to recent work in multidimensional databases and datacubes.

Chapter 4 began when Ronny Kohavi, Karl Pfleger and I read Utans & Moody (1991), which discussed experiments where attribute subset selection resulted in larger accuracy improvements than some other regularization methods. Together, we published a paper on the topic (John, Kohavi & Pfleger 1994). My contributions to that work included the idea of using cross-validation, initial attempts at a probabilistic definition of relevance, all of the code for the RELIEF and RELIEVED algorithms, most of the code for the forward and backward stepwise selection algorithms, and the discovery of some of the related work in statistics.

Chapter 5 began as John (1994*a*), which I did independently, and matured into Kohavi & John (1995). My contributions to that paper included using paired t hypothesis tests to evaluate the results, the design, coding, and experiments for the first generation system, and sharing the design of the second generation system and the writing of the second paper. The two papers and sets of experiments they describe are similar except for the choice of optimization method (exhaustive in the first paper, and best-first search in the second), and the greater breadth of experiments in the second.

Chapter 6 resulted from joint work with Pat Langley (John & Langley 1995). Pat had suggested one-dimensional density estimation as an area for improvement of the simple Bayes algorithm in 1992 when I was a teaching assistant for the machine learning class he was co-teaching with Nils Nilsson. This idea remained lodged in my brain until I learned about density estimation methods from Rob Tibshirani in a statistics class he was teaching at Stanford. My contribution to the paper was the idea of using kernel density estimation, the theoretical results, coding, running the experiments, and sharing the algorithm and experiment design and writing.

Chapter 9 is based on joint work with Randy Kerber and Peter Miller, who developed the rule induction algorithm and interface, respectively, for the data mining system used in the experiments (John, Miller & Kerber 1996*a*, John & Miller 1996, John, Miller & Kerber 1996*b*). My contribution was designing and running the experiments, writing the papers, interacting with many experts in finance to learn what parameters to use in the simulated trading and what kinds of portfolio analysis would be expected, and performing the simulated trading and analysis of results.

Chapter 2

Data Extraction: Interfacing with a Database

Although the data mining community has been focusing its efforts on larger and larger problems, little attention has been paid to the data extraction problem and the interface between the database and data mining algorithm. Relational database management systems (RDBMS's) currently manage the majority of databases that are the subject of analysis by data mining algorithms. Rather than the current process of requiring a user to manually extract data from a database, save it to a flat file, and then have a mining algorithm process that file, one could instead simply redirect the investigations of the mining algorithm so that it directly queried the RDBMS rather than manipulating database records itself. In this chapter, we propose the SQL Interface Protocol (SIP), which is one such framework for interaction between a mining algorithm and a database. The data continues to reside entirely within the DBMS, but the query interface to the database gives the data mining algorithm all the information it needs to discover patterns. This model of interaction brings several advantages, among them relieving the data mining analyst of the task of manually extracting data from a database, and possibly speeding up the data mining step by letting the mining algorithm leverage parallelized DBMS's running on fast hardware.

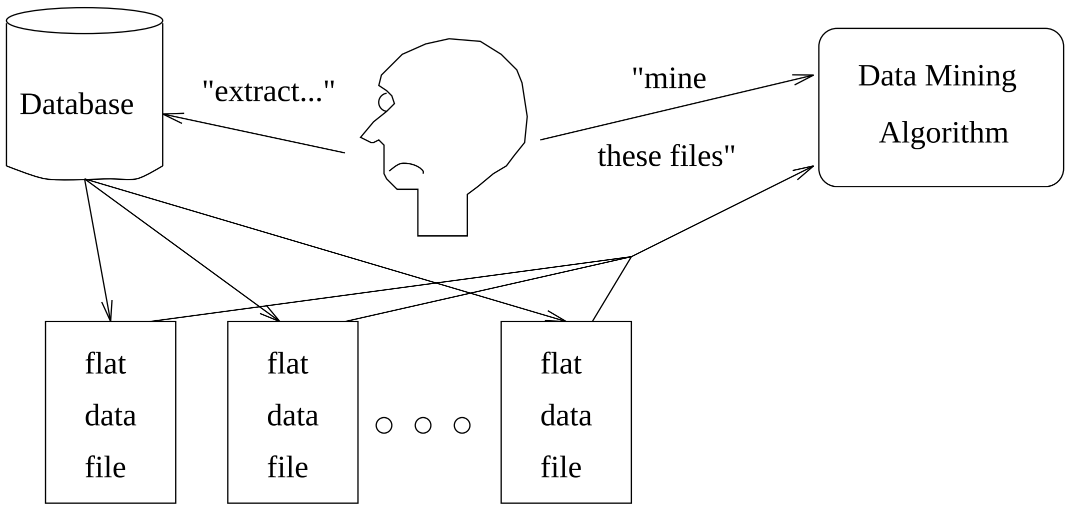

Figure 2.1: Existing process for mining large databases.

2.1 Introduction

Data mining algorithms are commonly expressed as programs that operate directly on a data file. Early algorithms from machine learning and statistics often loaded the entire file into main memory as a first step. As we move into industrial applications involving the mining of large data warehouses, this model becomes insufficient. Today, as disk space becomes cheaper and as more institutions attempt to leverage their data assets, "small" data warehouses are on the order of ten to twenty gigabytes in size. Not only is it infeasible to load such databases into main memory to mine the data, it also becomes quite unwieldy to manually extract data into flat files for analysis.

Recall that the data mining process begins with precisely stating a problem to be solved, and then deciding which data are appropriate to use in building a solution. The next step is to obtain the relevant data and process them into a form that a data mining algorithm can understand. Though often a simple problem technically, in practice this can be an arduous process (Figure 2.1). At the end one typically has produced one or more flat files that may be presented to a mining algorithm. There are several shortcomings of this approach. First, duplicating data is generally undesirable: it requires more storage space and it creates the problem of maintaining consistency with the source database. Second, since the data mining process is iterative, as

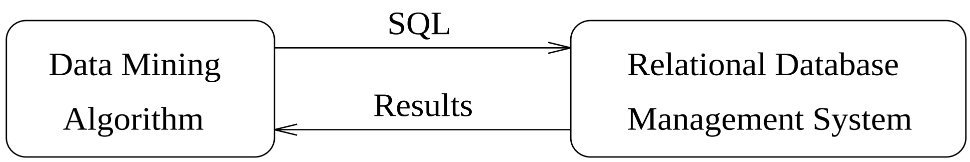

Figure 2.2: Proposed method for mining from SQL queries.

one redefines the data to be examined the whole extraction process may need to be repeated. Third, many file systems have a file size limit on the order of 2GB, so users must manually partition larger database extracts and maintain a set of files. Finally, it is not even clear that a mining algorithm would be able to process the resulting extract files if they are large, since many implementations of mining algorithms often load the file contents into main memory as a first step.

Not only is it cumbersome to give a mining algorithm access to the entire contents of a database, but it is also unnecessary. Many mining algorithms are based on relatively simple summary statistics of the database. The full information contained in the database is superfluous because the mining algorithm only uses the database to extract these statistics. If we could instead work within a framework where the mining algorithm simply gets the statistics it needs from an RDBMS, both problems—the impossibility of loading the file into memory and the daunting task of extracting flat files from the database—would be solved.

In the data extraction phase of the data mining process, recall that there are two steps. The first deals with identifying relevant information in the database and the second deals with processing the database into some form amenable to analysis by the data mining algorithms at hand. This second part can be replaced by our proposed SQL Interface Protocol (SIP) whenever the information a mining algorithm needs to know can be specified in terms of SQL queries (Figure 2.2). Rather than "drinking from the data firehose," the mining algorithm can direct the DBMS to process the volumes of raw data to discover the necessary statistics, and thus the data mining algorithm needs only to take a "sip" of this pure, concentrated information.

Many mining algorithms can be expressed mathematically such that all model-building steps are based only on functions of statistics of the data. These algorithms may be directly implemented within SIP by simply issuing SQL queries to get the required database statistics. Not all algorithms will be able to work within this framework; for example, neural networks access each full data record and thus will not work well in the SIP framework. The key in developing a version of a mining algorithm compatible with SIP will be determining exactly what information the algorithm needs to know about the data.

2.2 Information Requirements and SIP

The first step in defining the interface between mining algorithms and a DBMS is to decide what information the algorithms need to know. There is a large class of algorithms whose entire interaction with a database is simply a series of queries of conditional probabilities. For example, the simple Bayesian classifier uses only bivariate statistics involving the class and each attribute. All of the time spent running the algorithm is spent scanning the extracted database and calculating these statistics. SQL is rich enough to provide exactly the kind of summary statistics needed. In this section we define the SQL interface protocol for interaction between a data mining algorithm and DBMS, and show the kind of information SQL `GROUP-BY` queries can provide.

As an example, let us return to a customer retention problem, but for a change of scenery imagine that we are analyzing newspaper subscribers instead of cellular phone service subscribers as in Chapter 1. During the course of building a classification model, a mining algorithm might need to know the probability that a subscriber will cancel, given that they are in an older age group and have had recent negative interaction with the customer service department. (In the following, let us name these database attributes CANCEL, AGE, and SERVICE, where each can take on two values.) Based on this probability and results of previous queries, the mining algorithm will issue many new statistical queries. As well as querying the probability of a single event as above, the algorithm might want to know the entire distribution—what is the probability of observing each of the eight possible combinations of values of CANCEL, AGE, and SERVICE.

```
> SELECT CANCEL, AGE, SERVICE, COUNT(*)
  FROM SUBSCRIBERS
  WHERE DELIVERYTIME=MORNING AND PAYSCHEDULE=SPECIAL
  GROUP BY CANCEL, AGE, SERVICE

CANCEL  AGE    SERVICE  COUNT
------  ---    -------  -----
yes     old    positive   430
yes     old    negative  2214
yes     young  positive   322
yes     young  negative   631
no      old    positive 31227
no      old    negative 23419
no      young  positive 11993
no      young  negative 15526
```

Figure 2.3: Example `GROUP-BY` query in SQL, with `COUNT` aggregation.

SQL contains one form of query that gives exactly the desired results (Ullman 1988). A `GROUP BY` query in SQL can return a convenient table of counts showing how often any particular combination of the selected set of variable values occurred in the database. From these counts one can estimate the joint distribution over any set of discrete variables. This count table can be constructed from the entire database, or from a select subset of the records. Figure 2.3 shows an example `GROUP-BY` query and result where we restrict our attention to morning-delivery subscribers on the special payment schedule (which might be the rates offered by a special promotion). The result is a table of counts broken down by whether or not the subscriber canceled, their age group, and their recent customer service interaction.

Given a set of attribute names ϕ and a set of attribute-value assignments ψ, SQL `GROUP-BY` queries can easily answer the count query $N(\phi|\psi)$, which is a table of the number of times each possible assignment of values to the variables in ϕ was observed, considering only records where ψ holds. In the example above $\phi = \{$CANCEL, AGE, SERVICE$\}$ and $\psi = \{$DELIVERYTIME = MORNING, PAYSCHEDULE = SPECIAL$\}$. ϕ and or ψ might be null. When ϕ is null, $N(|\psi)$ returns a single count of all records where ψ holds. When ψ is null, $N(\phi)$ is a count of each possible value assignment to variables in ϕ, counting over the entire table. When both are null, $N()$, or simply N, is the total number of tuples in the relation. We assume that all queries are with respect to a single relation. If necessary, we can construct a view to make a single relation out of many tables.

The count query leaves to the mining algorithm the job of estimating the probability distribution from the counts, which is proper—it gives the algorithm the option of choosing an estimation method, perhaps maximum likelihood or Bayesian estimation. Counts are also important for statistical hypothesis tests that some algorithms use during mining. Almost all hypothesis test statistics are functions of the number of observations, so only getting a probability estimate from the database would be insufficient.

The following definitions distinguish between a "pure" and "impure" SQL query. The pure queries (Definition 2.1) return only statistics, not actual data values from the database. They answer exactly the $N(\phi|\psi)$ queries above. Impure queries return actual fragments of the database, but the intent is for the fragments to be small and the number of impure queries to be low.

Definition 2.1 *A* **pure SQL query** *is a model of interaction between a mining algorithm and database management system via SQL queries, such that the only queries issued by the mining algorithm are of the form*

`SELECT` $\phi, f(*)$
`FROM` D
`WHERE` ψ
`GROUP BY` ϕ

where f is an aggregation operator (count, sum, avg, stddev, min, max), ϕ is a list of attribute names, ψ is a list of attribute names with associated constraints, and D is the name of the relation.

Some data mining algorithms can *almost* be implemented using only pure SQL queries, but occasionally require more direct access to the database. For such algorithms we use impure queries.

Definition 2.2 *A k-th* **Degree Impure SQL Query** *is a model of interaction that allows all queries allowed by the strong SQL query model, plus queries of the form*
`SELECT` ϕ
`FROM` D
`WHERE` ψ
where ϕ includes exactly k attribute names.

Note that for an n-dimensional relation, the nth degree impure query is equivalent to asking for a dump of the entire relation, and thus any mining algorithm that normally uses flat files could as well be implemented in the nth degree impure SQL query model. This is of course uninteresting as it does not solve the important problems we identified in applying mining algorithms to large databases. Generally we intend for mining algorithms requiring impure queries to issue few impure queries, with k never more than two or three. Examples of cases where impure queries are required are given later.

Definitions 2.1 and 2.2 describe the two types of queries we will allow in the SQL interface protocol. In the following section we discuss common classification tree and Bayesian mining algorithms, and show that each may be implemented using SIP.

2.3 SIP-Compliant Algorithms

For concreteness, in this section we will discuss the C4.5 (Quinlan 1993) classification tree algorithm and the simple Bayesian classifier, showing how the algorithms can be modified to work with SIP rather than accessing a database directly.

2.3.1 The C4.5 Classification Tree Algorithm

Most classification tree mining algorithms are of the top-down, recursive partitioning variety. They begin with the entire training set of data records, decide on one way to split the data into two or more disjoint sets of records by testing the value of an attribute, and then repeatedly split these sets into subsets until a stopping criterion is met (either the set to be split is too small, or a statistical measure of the reliability

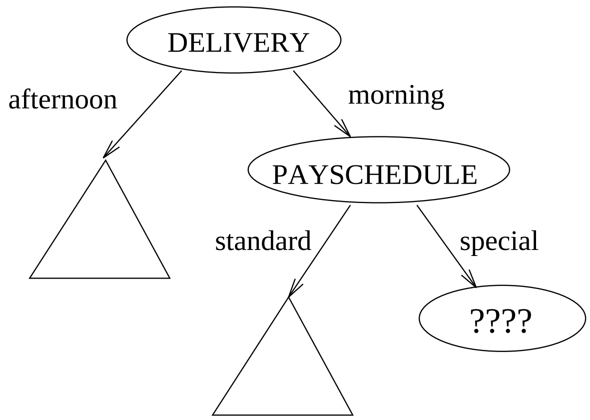

Figure 2.4: The general scenario in the classification tree building process: which attribute to split on next?

of estimates from the data set is too low). When drawn as a graph, the subset-of relation between the various sets of data forms a tree structure. Typically the functions used to split the data are tests on the value of a single attribute in the data. Each node is labeled with the name of the attribute, and the arcs to its children are labeled with a range of values of the variable. Classification trees are built with the goal of predicting the value of the class attribute, given the values of the rest of the attributes in a relation. Leaves are labeled with the most frequent class observed in their associated data set. As discussed earlier (Section 1.3.4), a new record is classified by sorting it down the tree until it filters down into a leaf, and the class of that leaf is assigned to the record.

A variety of algorithms have been developed within this framework. There are two main open issues when designing a classification tree mining algorithm: what *splitting criterion* is used to identify a good split during the recursive partitioning, and when to stop the partitioning. Tree-building algorithms generally use a fixed vocabulary of candidate splits, and simply evaluate each candidate as a potential split for each node, picking the best split to use in the tree. Nearly all *splitting criteria* are functions of counts involving the class and the candidate. (In Chapter 7, we give concise definitions for seven common splitting criteria in one table, showing that all are functions of counts involving the class and a single attribute being tested

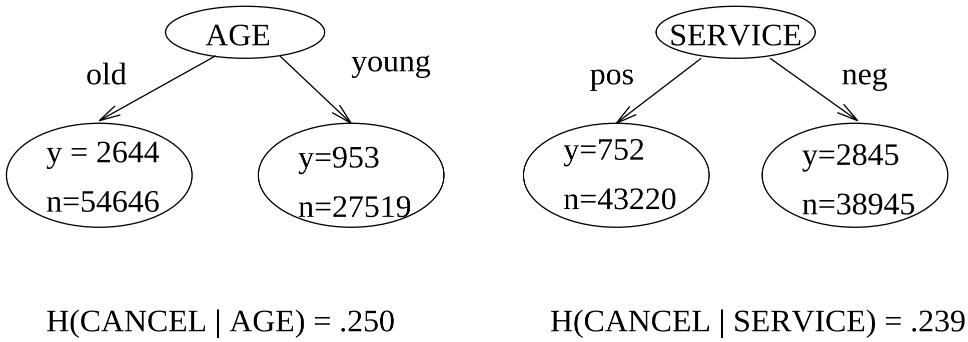

Figure 2.5: Two candidate splits and associated counts, with entropy values shown.

as a candidate split.)

Taking C4.5 as an example of a classification tree algorithm, we will define its splitting criterion precisely and define the appropriate pure SQL query that will generate the counts necessary to calculate the value of the criterion. C4.5 uses either the *information gain* or the *gain-ratio* splitting criteria. Since the first is simpler and gives the same results as the latter on average, we will investigate information gain. C4.5's default vocabulary of splits is the set of all partitions based on single categorical (character) attributes, where the number of partitions created by the split is equal to the number of unique values of the attribute. Figures 2.4 and 2.5 show a snapshot of the tree-building process in C4.5 using the customer retention example from before. Assume that the splits in the two complete nodes in Figure 2.4 were already determined by C4.5, and it is now focusing on the bottom right node, marked "????." The remaining attributes are AGE and SERVICE, so the two candidate splits are the binary split by AGE and the binary split by SERVICE, as shown in Figure 2.5. The data residing at the node in question consists of all records for which DELIVERYTIME = MORNING and PAYSCHEDULE = SPECIAL. For both candidate splits, the pure SQL query in Figure 2.3 provides sufficient information to count the number of CANCEL=YES and CANCEL=NO records in each subset of data induced by the split.

C4.5's splitting criterion then springs into action: it tries to maximize information gain, which is equivalent to minimizing the *conditional entropy* of the class given the attribute, $H(C|X)$, within the subset of data being considered, where C is the random variable denoting the class of a record and X denotes an attribute. Further, let c represent a particular class label, and let x represent a particular observed attribute value vector.

$$H(C|X) = -\sum_c p(C = c) \sum_x p(C = c|X = x) \log_2 p(C = c|X = x)$$

All of the probabilities involved in the calculation of $H(C|X)$ can be estimated from counts, and the counts are exactly those present in Figure 2.3. For example, the "y=2644" data in Figure 2.5 is obtained by summing the counts from the output of the pure SQL query for CANCEL=Y, AGE=OLD, SERVICE=POSITIVE and CANCEL=Y, AGE=OLD, SERVICE=NEGATIVE. Given the counts, we can estimate for example that p(CANCEL=YES | AGE=OLD) = 2644 / (54646 + 2644) = .046. Calculating the conditional entropy, we find it is smallest for the split on SERVICE, which would thus be chosen as the appropriate split for the node in question.

In general, to calculate the conditional entropy of the class given a single attribute, we would use a pure SQL query where ϕ is the attribute in question and the class, and ψ is the set of constraints along the path to the root, specifying which subset of the data has filtered down to the node in question. The example query of Figure 2.3 does not fit this specification since its ϕ includes two attributes (AGE and SERVICE) along with the class. This returns more information that we would need to calculate the conditional entropy for either attribute by itself, but if we know we need to calculate both it may be more efficient that issuing two separate queries.

The other important choice an algorithm for mining classification trees must make is when to stop growing the tree, or how to "prune" it back if it becomes too large during the growth phase. To give one example, the C4.5 algorithm grows large trees initially, stopping only when a node is *pure* (all records are from the same class), or when the data set is very small (fewer than 2 records, by default). C4.5's pruning phase then replaces a subtree with a leaf whenever a hypothesis test fails to show that the subtree has higher accuracy than the leaf. We will skip the details, but this hypothesis test is based upon counts: for each leaf, it counts the number of training records in its associated data set, broken down by their class.

These counts can all be obtained through SIP. Given a leaf, the constraints that must be satisfied for a record to filter into that leaf are obtained by tracing parent pointers to the root of the tree, giving a set of constraints on attribute values for the ψ in the pure query. ϕ is simply the class variable, and SIP would use the `COUNT` aggregation in the SQL query.

One issue we have not treated is splitting on a continuous variable. The standard approach is to sort the data by the value of the candidate and try all possible splits between different class values. For this we would need a second degree impure SQL query, to extract the numeric attribute and class attribute for each tuple. While this would be necessary to exactly mimic the functioning of C4.5, approaches to classification tree mining using pre-discretization (converting a real value into a discrete value using binning) have shown promise, and would alleviate the problem (Dougherty, Kohavi & Sahami 1995).

2.3.2 Simple Bayes

The simple Bayesian classifier directly characterizes the conditional distribution of the class given the attributes (John & Langley 1995). It makes strict ("simple") assumptions that allow it to build a model by estimating only a small number of parameters. Although one would think the model too simple to be of any use, a surprising characteristic of the algorithm is that it works quite well on many problems.

When mining for predictive patterns such as the simple Bayesian classifier, we are interested in the conditional distribution of the class given the observed attributes. Bayes' rule tells us that the conditional distribution of the class given the attributes,

$$P(C|\mathbf{X}) = \frac{P(C)}{P(\mathbf{X})} P(\mathbf{X}|C) \ , \tag{2.1}$$

and by assuming conditional independence we have

$$P(C|\mathbf{X}) = \frac{P(C)}{P(\mathbf{X})} \prod_i P(X_i|C) \ . \tag{2.2}$$

Equation 2.2 is the basis for the simple Bayesian classifier, which produces a model of exactly this form.

During mining, Simple Bayes gathers statistics (counts) required to model $P(X|C)$ for each input attribute X and statistics required to model $P(C)$.

$$\begin{aligned} P(C=c) &= N(C=c)/N \\ P(X=x|C=c) &= N(X=x, C=c)/N(C=c) \text{ for each nominal attribute} \\ P(X=x|C=c) &= g(x; \mu_c, \sigma_c^2) \text{ for each numeric attribute} \end{aligned} \tag{2.3}$$

The μ_c and σ_c parameters are the sample mean and variance for samples from class c, and g is the Gaussian density function with mean μ_c and variance σ_c^2. (Section 6.2 discusses details we omit here regarding Equation 2.3.) Since $\sum_c P(C=c|\mathbf{X}) = 1$, the term $P(\mathbf{X})$ in the denominator of Equation 2.2 is not explicitly calculated, but rather the estimates for each c of $P(C=c|\mathbf{X})$ are calculated and then normalized so that they sum to one, so the above equations describing how Simple Bayes estimates $P(C)$ and $P(X|C)$ completely specify how the model is built.

To clarify the probability estimation process, consider a small data set in which there are two classes, $+$ and $-$, a nominal attribute X_1 that takes values a and b, and a continuous attribute X_2. Given the five training cases $\{(+, a, 1), (+, b, 1.2),$ $(+, a, 3.0)$, $(-, b, 4.4)$, $(-, b, 4.5)\}$, simple Bayes obtains the probability estimates

$$\begin{aligned} \hat{P}(C=+) &= 3/5 \\ \hat{P}(X_1=a|C=+) &= 2/3 \\ \hat{P}(X_1=b|C=+) &= 1/3 \\ \hat{P}(X_2=x|C=+) &= g(x; 1.73, 1.21) \end{aligned}$$

for the positive class and analogous estimates for the negative class. By plugging these estimates into Equation 2.2, one obtains a fitted model that can be used to provide an estimate of $P(C|\mathbf{X})$.

All of the queries used in estimating $P(X|C)$ are pure SQL queries, where the aggregation function is either `COUNT` (for categorical attributes), or `AVG` and `STDDEV` (for numeric attributes). In all cases, ϕ includes C and a single X_i, and ψ is empty.

When depicted graphically, the model has the form shown in Figure 2.6, in which

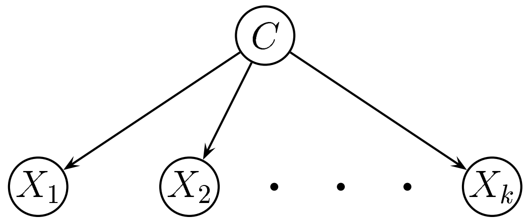

Figure 2.6: A simple Bayes model depicted as a Bayesian network. The input attributes $(X_1, X_2, \dots X_k)$ are conditionally independent given the class attribute (C).

all arcs are directed from the class attribute to the observable, predictive attributes. Many algorithms for building more general Bayesian networks, which relax the conditional independence assumption, are also based on counts and may be expressed in the strong SQL query model. For example, Friedman & Goldszmidt (1996) present a fast algorithm for learning Bayes nets with restricted topologies (a spanning tree connects the nodes) that makes choices about which arcs to add using functions of strong SQL query model counts. Their algorithm could also use strong SQL query model counts to estimate the parameters of the network after the topology is determined.

2.4 Related Work

In SIP, the relationship between mining algorithm and database management system is essentially that of a customer and a vendor. We have shown above that some important mining algorithms are capable of being customers in this model. We need the vendors, the DBMS's, to provide fast and accurate responses. If we were using one of today's off-the-shelf DBMS's, each SQL query issued by a mining algorithm might involve a pass through the database. However, current areas of database research are working towards database management systems that can efficiently produce and maintain complex models of data which could be used to quickly answer a variety of queries that mining algorithms would issue.

In the past two years, a number of researchers in databases have focused on efficiently computing aggregates, an important operation in on-line analytical processing (OLAP) databases used for decision support. Graefe (1993) describes a sort-based and hash-based method for efficiently computing the answer to a single `GROUP-BY` query, and would thus be useful in speeding up a single pure SQL query. Regarding multiple similar queries, Gray, Chaudhuri, Bosworth, Layman, Reichart, Venkatrao, Pellow & Pirahesh (1996) introduced the `CUBE-BY` operator. In our terminology, the `CUBE-BY` operator takes a set of attributes ϕ and returns the answers to all pure SQL queries (Definition 2.1) on the power set of ϕ. Sarawagi, Agrawal & Gupta (1996) present the PipeSort algorithm for efficiently computing the data cube, which runs in about 20% of the time required by the naive algorithm on five sample databases. Their algorithm can also be used to efficiently compute related sets of `GROUP-BY` aggregate queries which are subsets of a data cube.

Even more promising is the trend towards off-line precomputation of queries. Precomputing results to queries is an important technique for speeding up decision support applications, where query results are needed within a few seconds or minutes at most. For example, the data warehouse for Mervyn's department store has 2400 precomputed aggregations, to optimize performance. "Picking the right set of queries to materialize [precompute] is a nontrivial task, since by materializing a query we may be able to answer other queries quickly" (Harinarayan, Rajaraman & Ullman 1996). By precomputing the answers to queries asked by data mining algorithms, OLAP databases can exhibit faster response to queries issued by human experts looking for interesting patterns in the database, as well as data mining algorithms. Gupta, Harinarayan & Quass (1995) describe new query-optimization methods to automatically use precomputed aggregates when they are available, and Gupta, Harinarayan, Rajaraman & Ullman (forthcoming) describes indexing structures for the precomputed results. Looking at the other aspect of the relationship between data mining and databases, the queries asked by mining algorithms such as C4.5 might suggest the kinds of queries that would be of interest to a human expert looking at the same database (Jeffrey Ullman, personal communication), and could thus guide the precomputation process.

Selectivity is defined as "the ratio of the number of records that satisfy the query to the total number of records in a file" (Whang, Kim & Wiederhold 1994). Selectivity is used by DBMS's in query optimization, but can be useful in providing fast responses to pure SQL queries using the `COUNT` aggregation. Whang et al. (1994) explore the multi-level grid file method for precomputing and incrementally updating multidimensional selectivity estimates, so that selectivity can be estimated in time logarithmic in the number of attributes. They rely on some simplifying assumptions, essentially estimating the joint probability distribution of the data as a multidimensional histogram (*i.e.*, piecewise constant), and so the results may be incorrect. Slower but more exact methods for precomputing and updating statistics are studied in *statistical databases* (Shoshani 1982, Michalewicz 1992).

Most of the top DBMS vendors have parallelized their systems. As data storage and retrieval needs are growing at an exponential rate, while the cost per MIP of processing power or gigabyte of disk space has decreased dramatically. With increased parallelism, query response times are greatly enhanced. Since, as we propose, there exists a clean and modular interface between the mining components and the relational DBMS, any benefits realized by the DBMS because of increased parallelism will immediately benefit the mining process as well.

Within the machine learning community, the work that inspired the SQL interface protocol was Kearns (1993), which proposes the *statistical query model* for learning. Any mining algorithm that fits into the statistical query model will also fit into the SQL query model. However, the former model was developed as a theoretical device and assumes a statistical "oracle" answers queries, which ignores important issues in estimating probabilities. Our first insight, that almost all classification tree splitting criteria are based upon the same type of probability estimates, arose during work on a class of "soft" splitting criteria (John, 1995, and Chapter 8). In that setting, the similarity between the criteria is used in conjunction with the chain rule to more easily differentiate the splitting criteria with respect to the input variables.

2.5 Summary

An increasingly important issue in data mining is applicability to large databases. Since the databases community is already researching efficient algorithms for building, maintaining, and retrieving information from large databases, data mining algorithms should clearly leverage their efforts to the extent that they can improve the data analysis process. One approach is to rely upon a database management system to provide the statistics that a mining algorithm needs to build a model. For a variety of mining algorithms, this is relatively straightforward. Such algorithms can then be used in the SQL interface protocol for data mining. Current research in databases may provide extensive performance enhancements by providing technology by which a DBMS might create and maintain many of the statistics required by mining algorithms, using less time and disk space. As pointed out by Silberschatz, Stonebreaker & Ullman (1995), as databases are more frequently used for analysis and decision support, and as data mining algorithms are more often used for data analysis, the interface between the two is an important area that merits further research and development.

Chapter 3

Data Cleaning

Data cleaning is a very important step in the data mining process. Errors in large databases can be extremely common, so an important property of the data mining process is *robustness* with respect to errors in the database. Most sophisticated methods in data mining address noisy data to some extent, but not fully, and they can be improved by addressing the problem more directly. In this chapter we present ESTEEM (Elimination of Suspicious Training Examples with Error on the Model), a method of data cleaning that uses a data mining algorithm to find patterns in a database, then "improves the self-esteem" of the data mining algorithm by removing records from the training database that do not fit the discovered patterns, and retraining. We use ESTEEM to clean data for the C4.5 classification tree induction algorithm and show that on some databases, the resulting algorithm builds much smaller trees with slightly better accuracy than the standard algorithm.

3.1 Introduction

As data mining becomes more common in practice, users will apply data mining methods and algorithms to more and larger databases. An unfortunate property of many large databases is the high frequency of errors. Although theory and practice of database management has advanced rapidly over the past two decades (Ullman 1988), in the end the data stored in a database is no better than the typing skill of the data-entry clerk or the decisions made by experts that are recorded in the database. In order to be successful, data mining systems must be *robust* with respect to errors in databases.

Although nearly all machine learning and statistical methods address robustness over errors to some extent in the guise of overfitting avoidance or regularization, this puts the focus on only part of the system. Most methods for avoiding overfitting involve measuring the accuracy of the learned model on the training database, and measuring the complexity of the learned model. The model is then repeatedly made simpler until some desired balance of accuracy and complexity is reached. In this paradigm, the training database is fixed and the model is adjusted to optimize some criterion.

Another paradigm, which we explore here, allows the model *and* the training data to be adjusted. We present an iterative optimization method that alternately builds a model on a training database, then removes that set of records whose removal from the training set will most increase the model's estimate of its own performance (its "self-esteem") on the training data. This process of deleting records that disagree with the model is called ESTEEM, for Elimination of Suspicious Training Examples with Error on the Model.

While standard data mining algorithms penalize a model for being too complex, our method also penalizes the data itself for being too hard to learn, by removing records from the training database. As we shall see, in a case study using the C4.5 algorithm for building classification tree models, the ESTEEM method results in much more understandable patterns (in this case, trees) with slightly better predictive ability.

Section 3.2 discusses robust statistical methods. We then review the C4.5 tree building algorithm, explaining the pruning method used to avoid overfitting and relating it to ideas in robust statistics. Section 3.4 describes ESTEEM-C4.5, our ESTEEMed modification of C4.5, and Section 3.5 presents our experimental results. The next section discusses related work in machine learning, neural networks, and statistics; we then present our conclusions on data cleaning.

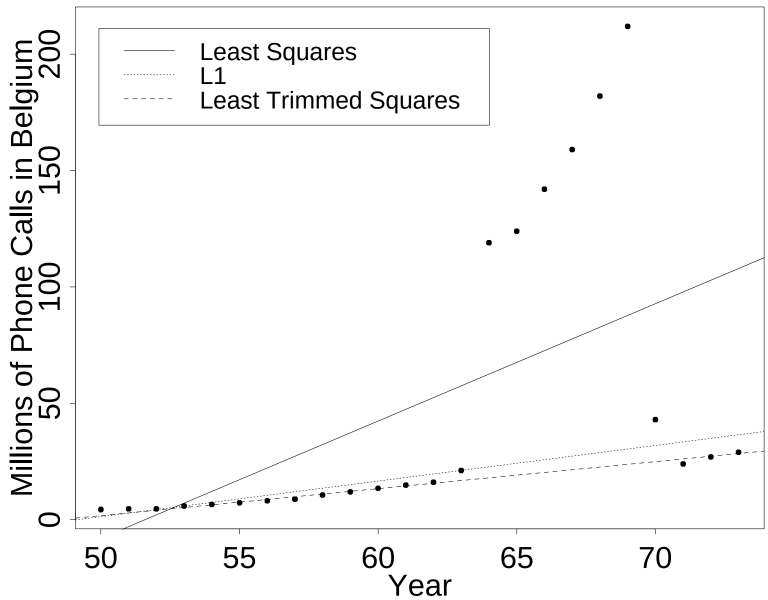

Figure 3.1: Millions of phone calls in Belgium, 1950–1973, from Rousseeuw & Leroy (1987), plotted in S-PLUS using code from Venables & Ripley (1994).

3.2 Robust Statistical Methods

All physical sciences and engineering disciplines are grounded on observations and measurements of physical entities and events. Whenever such data are collected into a large database, we almost inevitably have the problem of finding and addressing errors in the database. Since this is such a core issue to all physical sciences and engineering disciplines, one might expect that scientists and statisticians would have considered the problem for some time. In 1777 while studying astronomical data, Daniel Bernoulli wrote "I see no way of drawing a dividing line between those [data points] that are to be utterly rejected and those that are to be wholly retained; it may even happen that the rejected observation is the one that would have supplied the best correction to the others..." (Barnett & Lewis 1993, p. 27). Over two hundred years later, the problem of drawing this dividing line between the good data and the outliers still exists.

In statistics, an *outlier* is defined as a "case that does not follow the same model as the rest of the data" (Weisberg 1985). This is broad, including not only erroneous data but also "surprising" veridical data. Most of the work addressing outliers in statistics assumes either a simple estimation task, such as estimating the mean and covariance matrix of a normal distribution, or a linear regression. Statistical methods addressing the presence of outliers are called *robust*, since the estimators can withstand (are only minimally affected by) outliers in data. We will present a high level overview of work on outliers, and then discuss how they relate to data mining and classification trees.

Figure 3.1 shows a database of phone calls made in Belgium. Except for the six years from 1964–1969, it is a pretty dull database. The six outliers were caused by a mistake in recording the data—during the six-year period, the total number of minutes spent on the phone was recorded rather than the total number of calls. We can see that the effect of the outliers on standard linear regression is disastrous. Linear regression attempts to fit a line to a set of points so as to minimize the sum-squared residuals (errors):

$$\sum_p (y_p - \hat{y}_p)^2 \ ,$$

where p indexes each training point, and $\hat{y}_p$ indicates the value at x_p predicted by the linear model. A simple algebraic procedure finds the line minimizing this criterion. Outliers are especially problematic because of the squared error objective function.

There are two main ways that statisticians have addressed the outlier problem: accommodation and identification/rejection. We first discuss an example of the accommodation method, then the rejection method. Figure 3.1 shows the result of "L1" regression, which is just like least squares regression except that the objective function is changed to $\sum_p |y_p - \hat{y}_p|$, which does not over-emphasize points far from the fitted line. The accommodation methods, due to Huber (1981), involve modifying the objective function (usually sum-squared error) to discount the effect of outliers. Huber's proposal was the M estimate: $\sum_p \rho(y_p - \hat{y}_p)/\sigma$, where ρ is some function and σ is an unimportant constant scale factor. When ρ is the absolute value function we get L1 regression.

Rather than defining an objective function that lessens the impact of outliers while still allowing them to influence the fitted model, we may instead simply try to identify outliers directly and remove them from consideration. Another way to say this is that we may restrict our attention only to those training records that seem "normal." The least trimmed squares method of Rousseeuw & Leroy (1987) uses this idea—rather than minimizing the sum of squared errors, it instead minimizes the trimmed sum of squared errors, which corresponds to minimizing the errors on those points that seem most normal, and ignoring the rest.[1] Figure 3.1 shows that the least trimmed sqaures regression fits the veridical data best.

Another approach to identification is not through the residuals on each training record but rather on a measure of the effect a record has on the learned model. Points with high *leverage*, disproportionately high effect on the fitted model, are identified, and then a statistician investigates these points further to determine whether or not they should be included in the training database.

Commonly, outliers are thought of as shown in Figure 3.1—points that are far away from the rest of the data—but this need not be the case. For example, in a classification task, a point of class A might be an outlier if found in the middle of a dense cluster of points from class B. The subject of this chapter, the identification of outliers in categorical (nominal, unordered) data, has not been addressed in statistics. The ideas that seem to be easily portable from linear regression to classification tree induction are the idea of leverage and the idea of rejecting records that do not seem to fit the model.

3.3 C4.5 and Pruning: Model Selection

After inducing a classification tree from a database one is often left with a complex tree that fits the training database well. Most modern algorithms then *prune* the tree to make it simpler. This usually causes it to perform more poorly on the training data but better on data held out as a test sample. Given a complex classification tree with n nodes, we need some procedure for choosing one of the many subtrees that we may arrive at by pruning. C4.5's procedure for this involves repeatedly asking

[1] In least trimmed squares regression, to calculate the objective function, one calculates the squared error on each training point, then sorts these errors and discards or *trims* the top and bottom 10%, then sums the remaining errors.

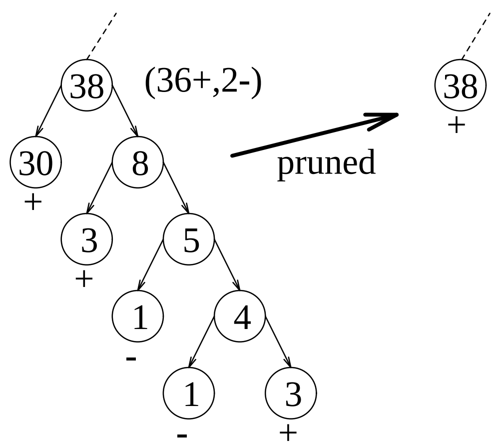

Figure 3.2: Leverage in a classification tree. The figure shows counts of the number of training records used to build each node. The two negative records caused the subtree to be built. When they are removed, the tree size shrinks by 8 nodes.

"should this node be made into a leaf?" at each node. The question is answered by estimating the error of the node given the current tree, then estimating the error if the node were pruned to a leaf. If the resulting error is smaller, the node is pruned. After repeatedly pruning or not pruning nodes, the pruning algorithm finishes and we are left with the final tree.

C4.5's error estimation method implicitly weighs the number of records that would be misclassified if a given current node were pruned against the reduction in tree size resulting from pruning the node. If only a few points will be misclassified, but many nodes can be saved, C4.5 will prune the node. Thus, after pruning, the set of records misclassified by C4.5's tree will approximate the set of points with high leverage—those few points which caused a large tree structure to be created so that they would be classified correctly.

3.4 EsteemEd C4.5: Data Selection

Pruning is a method of local data selection, since records are effectively removed from the partitions of the training set that induce the nodes deep in the tree. Pruning a node (*i.e.*, removing the subtree rooted at the node so that the node becomes a

Table 3.1: The ESTEEM method of outlier removal.

```
ESTEEM(Algorithm,TrainingData)
  repeat {
    Model <- Apply(Algorithm,TrainingData)
    foreach record in TrainingData
       if Model misclassifies Record then
          remove Record from TrainingData
  } until Model correctly classifies all
      Records in TrainingData
```

leaf) has the same effect as removing all records that were not in the majority class in the subset of the database used to build that node's subtree. This raises the question: if the records are *locally* un-informative or harmful, then why should we suspect that they helped the algorithm to discover patterns higher in the tree, that is, more globally?

The ESTEEM method repeatedly runs a classification algorithm and removes those records it misclassifies from the training database, until all records in the reduced training set are correctly classified (Table 3.1). The novelty in the ESTEEM approach is to *rerun* the mining algorithm using the reduced training set, which is the original training set minus the records which the first model classifies incorrectly. While retraining may seem odd, it is in fact just an extension of the assumptions underlying pruning. In C4.5, pruning the tree essentially assumes that these confusing records are locally not useful. Retraining merely takes this assumption a step further by completely removing these records from the training set. This makes explicit the assumption that locally uninformative or harmful records are globally uninformative as well. ESTEEM-C4.5 continues pruning and retraining until no further pruning can be done. ESTEEM is guaranteed to stop since the size of the training set monotonically decreases as a function of the number of iterations through the main loop. Though common in regression in the guise of *robust* (Huber 1977) or *resistant* (Hastie & Tibshirani 1990, Chapter 9) fitting, in the context of classification this idea appears to be novel.

Relating this method to the previous discussion on robust statistics, in the context of classification trees we may identify a set of records with high leverage by examining the difference in number of nodes between a tree built with and without the records (Figure 3.2). This difference is estimated by starting with a pure tree built using all the points and then pruning. The training records that it now classifies incorrectly are the high leverage points that were removed. However, the removal of the points by pruning was only approximate—the obvious step is to remove the points from the training set and retrain.

3.5 Experimental Evaluation of Esteem

Because of the arguments above, we hypothesized that Esteem-C4.5 would be a worthy competitor to C4.5, always resulting in smaller trees and increasing accuracy on domains with many outliers. Also, since the definition of a robust statistic requires small changes in the statistic for small changes in the data, the standard deviation of the accuracy of Esteem-C4.5 should be smaller than C4.5. Below we first describe the experimental method designed to test these hypotheses, along with the results.

3.5.1 Experimental Method

To test the hypotheses, we compared Esteem-C4.5 and C4.5 on 22 databases selected from the UCI and Statlog database repositories (Murphy & Aha 1994, Michie, Spiegelhalter & Taylor 1994). The databases are all stored and processed as a single relation (table). To compare Esteem-C4.5 and C4.5 on a given database, we used ten-fold cross-validation: we partition each database into ten parts of equal size, then repeatedly train on nine of the parts and test on the remaining one, running each algorithm a total of ten times. We expected Esteem-C4.5 to build trees that are smaller than C4.5's trees, with about the same accuracy but with more stability (*i.e.*, Esteem-C4.5's trees should vary less as we randomly pick different subsets of records for training).

We also ran a set of lesion experiments, where we crippled (or "lesioned") the

Esteem-C4.5 algorithm in the following way: instead of rejecting the records that were misclassified during one pass of the algorithm, LE-C4.5 instead rejects the same number of records but it randomly chooses which records to remove. Each run was repeated ten times (throwing out a different random set of records) and the results were averaged. This experiment was important because, after the first set of experiments, there remained the possibility that the good results were not due to the success of the Esteem method, but rather that the moderately reduced training set size resulted in slightly smaller models that were about as accurate. We hypothesized that LE-C4.5 will build models with worse accuracy than C4.5 and Esteem-C4.5, and with smaller size than C4.5.

For purposes of testing the hypotheses, the important statistics to gather for each run of each algorithm on each database are the mean of the accuracies from the ten runs, the mean of the tree sizes, and the standard deviation of the accuracy among the ten cross-validation runs.

3.5.2 Experimental Results

Table 3.2 presents the accuracy results from the experiments described above. For each database and algorithm, we report the average of the ten cross-validation accuracies and their standard deviation. (Since the accuracies for LE-C4.5 are already averages of ten runs, their standard deviation cannot be directly compared with Esteem-C4.5 and C4.5.) At the bottom of the table, each column's average is reported.

There are several interesting things to point out in the table. Esteem-C4.5's average accuracy is slightly higher than C4.5, which weakly supports our hypothesis that Esteem-C4.5 would have higher accuracy than C4.5. Both Esteem-C4.5 and C4.5 had higher accuracy than LE-C4.5. Note that since Esteem-C4.5 specializes in the removal of outliers, it might perform badly on complex databases where there is no noise but where some patterns are under-represented. In such settings we might expect Esteem-C4.5 to degrade performance by throwing out perfectly good data (the problem that worried Bernoulli over two hundred years ago), and this was the

Table 3.2: Accuracies for C4.5, E-C4.5, and LE-4.5, and the percentage of training records that were discarded.

	Accuracy Avg			Accuracy StdDev		% Dis-
Database	C4.5	E-C4.5	LE-C4.5	C4.5	E-C4.5	card
australian	85.0	84.5	84.3	4.1	2.8	8.2
breast-cancer	74.5	74.5	71.8	5.2	5.2	24.0
breast	94.8	95.3	95.0	2.4	2.3	2.8
chess	99.5	99.5	99.4	0.4	0.4	0.4
cleve	75.6	76.2	74.6	3.9	4.4	10.2
crx	82.8	83.3	84.0	3.4	2.9	8.8
diabetes	72.2	71.3	71.7	5.1	3.4	18.0
german	72.7	72.9	71.9	4.4	4.0	18.8
glass	69.6	67.7	68.4	8.0	8.4	11.6
glass2	77.2	77.9	76.2	8.0	6.8	6.8
heart	74.1	76.7	75.5	7.2	6.3	9.4
hepatitis	76.7	76.7	76.3	6.5	6.6	12.0
horse-colic	85.9	85.3	84.4	4.9	4.8	11.7
hypothyroid	99.2	99.2	99.2	0.5	0.5	0.7
iris	94.0	94.7	94.8	5.8	6.1	2.4
labor-neg	82.7	87.7	85.6	15.8	16.1	10.8
lymphography	75.5	75.5	74.4	12.8	12.8	10.8
segment	96.2	95.8	96.2	1.1	1.1	1.6
sick-euthyroid	97.6	97.7	97.6	0.9	0.9	1.6
tic-tac-toe	86.2	83.3	84.8	4.6	3.9	11.6
vote	94.7	94.7	94.7	2.2	2.2	3.1
vote1	88.7	88.5	88.2	5.9	5.7	7.4
AVERAGE	84.33	84.50	84.06	5.14	4.89	8.8

case. Using a paired t test, we found that the accuracies of Esteem-C4.5 and C4.5 were statistically indistinguishable (at the 95% level) on all databases but three. Esteem-C4.5 performed better than C4.5 on the Wisconsin breast cancer database ("breast"), but performed worse on Tic-Tac-Toe and the image segmentation problem ("segment"). The Tic-Tac-Toe database is noise-free but complex— the average tree size is a very high 133—so it fits our "hard-but-error-free" profile perfectly.

It is difficult to discern any pattern in Table 3.2 that describes how the percentage of discarded records is related to change in accuracy between C4.5 and Esteem-C4.5.

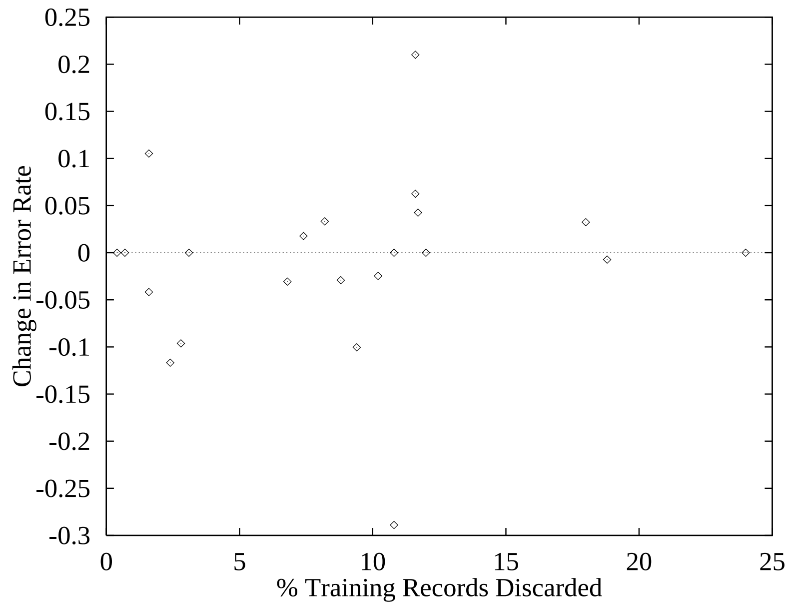

Figure 3.3: Percent of records discarded vs. percent change in error rate.

Figure 3.3 shows a plot where the horizontal axis indicates the percentage of training records discarded by ESTEEM-C4.5 and the vertical axis indicates the change in error rate between C4.5 and ESTEEM-C4.5. For example, the labor-negotiation database is responsible for the point (10.8,-.289), because 10.8% of the records were dropped by ESTEEM-C4.5 and (12.3 - 17.3)/17.3 = -28.9%. Except for this domain, on the databases where 10% or more of the training records were removed, we usually see an increase in error rate. This suggests that when more than about 10% of the records are removed, the algorithm's definition of an outlier is not a good match for the database.

Regarding the robustness of ESTEEM-C4.5, note that the average standard deviation is indeed smaller than C4.5 by .25. Looking at each database, the standard deviation of ESTEEM-C4.5 was smaller than C4.5 on ten, the same on seven, and higher on five. This weakly supports the last hypothesis and indicates that, by removing outliers, the ESTEEM-C4.5 algorithm is able to learn more stable representations of the hidden patterns in the data than C4.5.

Another interesting pattern in the data is the average tree size, shown in Table 3.3. ESTEEM-C4.5 manages to produce trees on average over 20% smaller than C4.5's

Table 3.3: Tree sizes for C4.5, E-C4.5 and LE-C4.5.

Database	C4.5	E-C4.5	LE-C4.5
australian	59.8	48.1	55.5
breast-cancer	10.7	11.0	16.6
breast	18.4	17.8	18.6
chess	55.0	55.0	55.0
cleve	39.8	35.5	41.6
crx	57.7	42.8	53.5
diabetes	127	71.0	107
german	152	84.8	134
glass	50.0	40.2	45.1
glass2	26.8	27.2	27.3
heart	40.8	28.8	38.4
hepatitis	16.0	10.0	13.9
horse-colic	17.4	15.6	17.3
hypothyroid	12.2	12.2	12.2
iris	9.6	9.60	9.4
labor-neg	7.3	7.40	6.9
lymphography	26.7	23.2	24.7
segment	82.6	76.4	81.7
sick-euthyroid	25.6	25.4	26.0
tic-tac-toe	133	112	117
vote	14.5	14.5	14.1
vote1	28.6	26.8	24.0
AVERAGE	46.03	36.19	42.81

trees, and up to 44% smaller, while maintaining the same level of accuracy. In many cases ESTEEM-C4.5 can dramatically reduce tree size—note the "diabetes" (Pima Indian diabetes) and "german" (accept/reject loan applications) databases, where tree size was almost cut in half. The "german" database probably resembles most closely the types of tasks common in data mining: given attributes such as salary, savings, type of loan and other personal information, predict whether the loan should be granted or denied. Since one of the goals of data mining is to present understandable patterns to a data analyst, ESTEEM-C4.5 should be a worthwhile additional tool.

Trevor Hastie (personal communication) has suggested that perhaps the success of Esteem-C4.5 is due to the fact that C4.5 does not prune enough, so that the advantage from Esteem-C4.5 results from the fact that after rebuilding the tree, the pruning algorithm gets another shot at reducing the size further. To test this hypothesis we ran two other full experiments like the one reported above with different settings for C4.5's pruning parameter, and found qualitatively the same results as in our original experiment. Thus the superior performance of Esteem-C4.5 does not seem to be due solely to underpruning on C4.5's part.

3.6 Related Work

A similar automated approach to handling outliers is described by Brodley & Friedl (1996), who build an ensemble of classifiers and only reject a record if all of the models misclassify it. If we also have a model built by encoding knowledge acquired from a human expert, we can use this model to identify potential errors (Drastal, Meunier & Raatz 1989). Weigend (1996) uses a more sophisticated approach, wherein during the model fitting the free parameters iterate between the model and the training data. Weigend's "clearning" (cleaning while learning) method takes a much more fine-grained approach, allowing the algorithm not only to decide whether to keep or remove a training record, but also allowing the algorithm to make slight adjustments in a record's attribute values.

Regarding robust methods and outlier rejection, Huber (1977) states "I am inclined to ... prefer technical expertise to any 'statistical' criterion for straight outlier rejection." We are guilty of this sin in our work, but Guyon, Boser & Vapnik (1993) have proposed an interesting method for making use of human expertise in outlier removal to "clean" a database. Similarly, rather than rejecting suspected outliers outright, Esteem could as well interview an expert, presenting the suspicious database records and asking for an opinion as to whether the record should be removed or down-weighted. Barnett & Lewis (1993) give an interesting and entertaining survey of work in outlier removal in statistics. Venables & Ripley (1994, ch. 8) present a concise review of robust statistics, accompanied by S-PLUS code.

Another way of looking at robust methods (and the identification approach in particular) is as a data selection method. The selection of data for training is an ubiquitous problem in data mining. Many methods are *passive*, accepting data from a training sample in some random order or all at once as a set. Other methods actively accept or reject training records from a temporal sequence presented to them (Aha 1991). The difference between outlier removal and active learning (as commonly presented) is somewhat akin to the difference between forward and backward attribute subset selection (Chapter 4) or forward and backward (construction/pruning) search methods over neural net or classification tree architectures. In general, unless time is a very limited resource, the best results are achieved by starting "big" and then shrinking (Breiman et al. 1984). This should apply to data selection as well, and thus we expect backward data selection (*i.e.*, outlier removal) will give better results than forward selection. Cohn (1994) discusses yet more "active" data selection methods that might perform even better, but they require the availability of a domain expert.

Since Esteem-C4.5 just runs C4.5 many times on similar databases, the approach could benefit from an incremental tree-updating scheme such as that proposed in Utgoff (1994). Rather than removing the outliers from the database and retraining on the entire remainder, Esteem-ITI could instead simply incrementally remove the outliers from the tree itself. By avoiding the full retraining step, a significant amount of time could be saved.

One can also use Esteem in combination with other data mining methods. For example, association rules can be used to spot outliers in databases. If an association rule has 99.5% confidence, then one ought to be highly suspicious of the .5% of records that satisfy the antecedent of the rule but not the consequent. Such cases often indicate either errors in the database or actual low-probability events that the domain experts were not aware of. The Esteem method may be applied directly to association rules, but in this case the value is not in producing better rules; rather, the value lies solely in identifying potentially erroneous records in the data.

3.7 Conclusion

In this chapter we have presented a method for handling outliers in data mining based on ideas drawn from robust statistics. Rather than just being concerned with the complexity and quality of the patterns discovered by a data mining algorithm in a database, the Esteem method also actively screens the training database in an attempt to ignore erroneous records. Such robust methods seem well-suited to learning patterns in databases that are likely to contain many errors—by ignoring the errors, the discovered patterns can be more accurate and far simpler than patterns produced by comparable algorithms that are forced to pay attention to all records. Our experiments on 22 databases support this claim, showing that on databases with characteristics typical of data mining tasks, Esteem-C4.5 can learn trees with slightly higher accuracy and up to 44% fewer nodes than C4.5's trees.

We have only investigated the use of Esteem with C4.5 in this chapter, but the general idea can apply to any data mining task with a relational table as input data, as long as the mining algorithm uses some objective function to measure its performance on each record of the database. In this case the performance measure was classification accuracy, but one could as well imagine using squared error in the regression setting or likelihood in a clustering application. Future work should also investigate the use of Esteem to reduce the influence of suspicious records, rather than completely removing them.

Chapter 4

Data Engineering: Attribute Subset Selection

In this chapter, we address the problem of finding a subset of attributes that lets a data mining algorithm discover more useful patterns. Typical data mining problems involve large databases with many attributes per record. Usually, the best approach to such databases is to use the domain expert to liberally identify a large number of potentially relevant attributes, and then run the data mining algorithms and hope that they can do a good job of selecting the most important attributes. However, data mining algorithms can be confused by irrelevant attributes, and sometimes even the inclusion of attributes that are highly relevant can diminish the quality of the discovered patterns. We examine definitions of two degrees of relevance, which improve our understanding of the behavior of attribute subset selection algorithms.We maintain that the attributes selected should depend not only on the attributes and the target concept, but also on the data mining algorithm. As in Chapter 3, we describe a method that automatically iterates through part of the data mining process. The "wrapper" method presented in this chapter is applicable to any mining algorithm so long as there is an objective function that lets the method compare the quality of the patterns discovered during several mining runs. As a case study, we apply our method to the ID3 and C4.5 classification tree algorithms showing that they can be improved by the addition of automatic attribute selection.

4.1 Introduction

The data engineering step in the data mining process includes a wide range of activities, including the manufacture of new attributes that the data mining analyst and expert feel will be more informative than the raw attributes in the database. Here we focus on the attribute subset selection step, which involves the choice of which subset of attributes to use during the rest of the data mining process. Using only a subset of the available attributes can give three advantages: first, it speeds up the data mining step by reducing the total amount of data that needs to be considered; second, reducing the number of available attributes usually reduces the number of attributes that appear in the discovered patterns, making them easier for the domain expert to understand; and third, most data mining algorithms can be confused by the presence of irrelevant or even relevant but harmful attributes, and by removing them we expect to find higher quality patterns.

Table 4.1 shows some examples of the ill effects of suboptimal attribute subset selection. We use the credit approval database and the Pima Indian diabetes database from the UCI repository. The table shows the results from running C4.5 on a training set containing 75% of the records and a test set containing the remainder. The table also shows the degraded accuracies when an irrelevant attribute was added. The irrelevant attribute was binary and its values were uniformly randomly distributed. On both databases, C4.5 chose the irrelevant attribute at some point as the best split at some node in the tree. Even though the attribute is irrelevant, at some node in the tree, because of fragmentation the estimates of the conditional distribution of the class given the irrelevant attribute seemed to indicate that it would be the best split at that node. As the table shows, this significantly hurt the accuracy of the resulting tree on an independent test set. But perhaps surprisingly, a relevant attribute can also be harmful. We constructed a relevant attribute so that 65% of the time it would have the same value as the class, and the rest of the time its value would be the opposite of the class. Even though it was informative, it hurt the accuracy of C4.5's trees in the two domains because C4.5 selected it for a split high in the tree, which caused fragmentation.

Table 4.1: The ill effects of suboptimal attribute selection.

Attributes Used	Accuracy Credit Approval	Accuracy Diabetes Diagnosis
all attributes	87.9	74.0
all + relevant	85.5	73.5
all + irrelevant	82.7	68.4

The problem of *attribute subset selection* involves finding a "good" set of attributes under some objective function that assigns some numeric measure of quality to the patterns discovered by the data mining algorithm. In the classification setting that we focus on in this chapter, common objective functions are prediction accuracy or misclassification cost, pattern complexity, and minimal use of input attributes (*e.g.*, when attributes are tests that have an associated cost). We chose to investigate the possibility of improving prediction accuracy or reducing the number of attributes used without significantly decreasing prediction accuracy. This specific problem has been thoroughly investigated in the statistics literature, but under assumptions that do not apply to most data mining algorithms.

We begin in Section 4.2 by formalizing the attribute subset selection problem. Section 4.3 gives several definitions of attribute relevance, and describes their relationships and properties. Sections 4.4 and 4.5 look at two approaches to attribute subset selection: the filter method and the wrapper method. We claim that the wrapper method is more appropriate than the filter method, which has received more attention in the literature. Section 4.6 presents our experimental results, Section 4.7 describes related work and promising lines of future research, and Section 4.9 presents somer conclusions about attribute subset selection.

4.2 The Attribute Subset Selection Problem

The objective of a data mining algorithm $\mathcal{A}$ is to take a training set T and discover a set of patterns P such that P optimizes some objective function $F(P)$ that assigns some real-valued measure of goodness to P. The output of $\mathcal{A}$ is determined by which attributes are present in the training set. We can parameterize the attributes used as a Boolean vector θ, where $\theta_i = 0$ means that attribute i is not used, and 1 indicates that it is used. Thus we can think of the training set T as a function that takes the full training set and θ and returns a training set that includes only the attributes prescribed by θ. We define the optimal attribute subset θ^* in Definition 4.1.

Definition 4.1 (Optimal Attribute Subset)

$$\theta^* = \underset{\theta}{argmax} F\left(\mathcal{A}(T(\theta))\right) \quad . \tag{4.1}$$

In general, finding θ^* is impossible for two reasons. First, most objective functions F cannot be calculated precisely, and can only be estimated from the database. For example, if $\mathcal{A}$ is an algorithm that builds classification models, we might want to find P with minimal misclassification cost. If P is a classification model then it is itself a function mapping an input vector $\mathbf{x}$ to an output Y. Given some probability distribution P_X over $\mathbf{x}$, P's misclassification cost is defined as $\int_{\mathbf{x}} C(P(\mathbf{x}), f(\mathbf{x}))P_X(\mathbf{x})d\mathbf{x}$. C gives the cost of predicting $P(\mathbf{x})$ when the true class is $f(\mathbf{x})$; if we only care about error rate, then C is 1 when its arguments are different and 0 when they are the same. We cannot directly calculate the misclassification cost of P since the distribution P_X is unknown. The only information we have about the distribution of $\mathbf{x}$ is the training set T. Thus, we must somehow estimate the value of the objective function on the patterns using only the training set.

Even if we did know the distribution P_X, we have the practical problem that if there are m attributes then there are 2^m possible values of θ, and this number is typically too large to search exhaustively. Thus, in addition to using some method to approximate the quality of the discovered patterns, we must also find some way to search only a portion of the space of all subsets.

Since both obstacles prevent us from finding θ^*, we must instead be content to find some $\widehat{\theta}$ approximating θ^* that also results in low expected loss. In Section 4.5 we discuss an error estimation method for estimating misclassification rate and heuristic search methods for exploring only a subset of Θ.

4.3 Relevance

One way to avoid directly finding a set of attributes satisfying Definition 4.1 is to conjecture that the right attributes to use are those that are *relevant*. In this section we review definitions of relevance that have been suggested in the literature, and we present our own definitions of strong and weak relevance. We use examples to show that the existing definitions of relevance do not adequately capture the notion of relevance. Our own definitions of strong and weak relevance agree with our intuition and correctly identify relevant attributes that are not identified by existing definitions.

Almuallim & Dietterich (1991) define relevance under the assumption that all attributes are Boolean and that there is no noise. In their definition, an attribute is relevant to a concept if and only if it appears in every Boolean formula that represents the concept. We generalize their definition, extending it in Definition 4.2 to cover any supervised data mining task, by referring to expressions of conditional probabilities instead of Boolean formulae. Since Almuallim & Dietterich (1991) presented an algorithm they call FOCUS, we refer to Definition 4.2 as the PFOCUS definition of relevance.

Definition 4.2 (PFOCUS)
An attribute X_i is relevant if and only if every expression for $P(Y|\mathbf{X})$ contains X_i.

Another approach to defining relevance is given by Gennari, Langley & Fisher (1989), who define relevant attributes as those whose "values vary systematically with category membership." We formalize this as stating that X_i is relevant if and only if there exists some x_i and y such that $p(X_i = x_i \mid Y = y) \neq p(X = x_i)$. We prefer to work with conditional probabilities where the output value is the subject and the input attributes are the context, and Lemma 4.1 allows us to do this.

Lemma 4.1
$p(X_i = x_i \mid Y = y) \neq p(X = x_i)$ *iff* $p(Y = y \mid X_i = x_i) \neq p(Y = y)$.

Proof: Assume $p(X_i = x_i \mid Y = y) \neq p(X = x_i)$. If $p(Y = y) = p(Y = y \mid X_i = x_i)$ were true, we have

$$\begin{aligned} p(Y = y) &= p(Y = y \mid X_i = x_i) \\ p(Y = y) &= p(Y = y)p(X_i = x_i \mid Y = y)/p(X_i = x_i) \\ p(X_i = x_i) &= p(X_i = x_i \mid Y = y) \ , \end{aligned}$$

which violates our assumption. Thus, it must be the case that $p(X_i = x_i \mid Y = y) = p(X = x_i)$. This proves the "if" part of the "if and only if," and a similar argument proves the "only if" part. ∎

Definition 4.3 (Cobweb)
An attribute X_i is relevant iff there exists some x_i and y such that

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

Under Definition 4.3, X_i is relevant if knowing its value can change the estimates for Y or, in other words, if Y is conditionally dependent on X_i. Note that this definition fails to capture the relevance of attributes that only affect the output when certain other attributes are known.

Definition 4.4 addresses this problem by indicating that an attribute X_i is relevant if the probability of the output given all attributes can change when we eliminate knowledge about the value of X_i.

Definition 4.4 (Strong relevance)
Let S_i be the set of all attributes except X_i, i.e., $S_i = \{X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_m\}$. Denote by s_i a value-assignment to all attributes in S_i. X_i is strongly relevant iff there exists some x_i, y, and s_i for which $p(X_i = x_i, S_i = s_i) > 0$ such that

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

Strong relevance (Definition 4.4) implies that the attribute is indispensable in the sense that one cannot express the true conditional distribution of Y given $\mathbf{X}$ if X_i is missing from $\mathbf{X}$. In other words, X_i must be present in any expression for $P(Y|X)$, which generalizes the PFOCUS definition of relevance to probability distributions. We will prove that strong relevance generalizes PFOCUS relevance below in Theorem4.1.

Lemma 4.2

Given three sets of random variables, S, R, and I, if $p(S|R,I) = p(S|R)$, then for any I' subset of I, $p(S|R,I') = p(S|R)$.

Proof: Let $I'' = I - I'$.

$$\begin{aligned} p(S|R,I') &= \sum_{I''} p(I''|R)p(S|R,I',I'') \\ &= \sum_{I''} p(I''|R)p(S|R,I) \\ &= \sum_{I''} p(I''|R)p(S|R) \\ &= p(S|R)\sum_{I''} p(I''|R) \\ &= p(S|R) \end{aligned}$$

▮

Theorem 4.1

X_i is strongly relevant iff X_i must appear in all expressions of $p(Y|\mathbf{X})$. Formally, define S_i as in Definition 4.4. Then, X_i is strongly relevant iff for any $S' \subset S$, $p(Y|X_i,S') \neq p(Y|S')$.

Proof: If X_i is strongly relevant, then by definition $p(Y|X_i,S) \neq p(Y|S)$ and thus by Lemma 4.2 there exists no S' such that $p(Y|\mathbf{X}) = p(Y|S')$, because this would imply that $p(Y|X_i,S) = p(Y|S)$, which contradicts the definition of strong relevance. Thus, if X_i is strongly relevant, there is no expression of the form $p(Y|S') = p(Y|X_i,S)$, thus X_i must appear in all expressions of $P(Y|X)$. Conversely, if X_i appears in all expressions of $P(Y|X)$ then it must be the case that $p(Y|X_i, S) \neq p(Y|S')$, otherwise $p(Y|S')$ would be an expression for $p(Y|X)$. ∎

One shortcoming of strong relevance is that it can be too conservative about which attributes it labels as relevant. For example, if there happen to be two redundant attributes that give the same information, then either one may be dropped without harm and thus neither is strongly relevant, even if the information they contain is highly relevant. A similar problem arises when using indicator variables to encode nominal attributes. For example, the indicator variable encoding of a four-valued nominal $\{a, b, c, d\}$ would be $\{0001, 0010, 0100, 1000\}$. Under such an encoding, any single indicator variable is redundant and can be determined by the rest. Thus none of the variables is strongly relevant. Since strong relevance can be too strict, we developed the notion of *weak* relevance, described by Definition 4.5.

Definition 4.5 (Weak relevance)
An attribute X_i is weakly relevant iff it is not strongly relevant, and there exists a subset of attributes S'_i of S_i for which there exists some x_i, y, and s'_i with $p(X_i = x_i, S'_i = s'_i) > 0$ such that

$$p(Y = y \mid X_i = x_i, S'_i = s'_i) \neq p(Y = y \mid S'_i = s'_i) \ .$$

Weak relevance implies that the attribute can sometimes contribute to prediction accuracy. We call attributes *relevant* if they are either strongly or weakly relevant, and *irrelevant* otherwise. Irrelevant attributes can never contribute to prediction accuracy, by definition.

Table 4.2 shows the possible kinds of relevant attributes, under the definitions described in this section, and gives examples of each kind of attribute. Note that the Cobweb definition of relevance does not correctly identify all relevant attributes. PFOCUS is not represented explicitly in the table since it is generalized by strong relevance, but note that strong relevance by itself also fails to identify all relevant attributes.

The definitions of relevance are useful in understanding the concept, but are not put forth as bases for attribute subset selection algorithms. Since they are all based

Table 4.2: The possible kinds of relevant attributes.

Cobweb	Weak	Strong	Explanation & Example
N	Y	Y	Impossible: by definition of weakly relevant.
Y	Y	Y	Impossible: see above.
Y	N	N	Impossible: all Cobweb-relevant attributes are either weakly relevant or strongly relevant.
N	N	N	An irrelevant attribute. *e.g.*, if customer ID numbers are assigned randomly, then the ID numbers are unrelated to any other attributes of the customer, and would not be relevant by any definition.
Y	N	Y	A typical strongly relevant attribute. *e.g.*, regardless of the other attributes that may be known about a customer, for some customers tenure will always effect their probability of attrition, thus the tenure attribute is strongly relevant.
N	N	Y	An atypical strongly relevant attribute. *e.g.*, say Siskel and Ebert vote on whether they like a movie, and their votes are statistically independent. If their votes are input attributes and the output is whether or not they disagree, then each vote is strongly relevant, since we must know both to say whether the vote was mixed, but knowing only one vote is worthless, thus the votes are not Cobweb-relevant.
Y	Y	N	A weakly relevant attribute where S' can be empty. *e.g.*, say Siskel and Ebert's friend can often correctly predict when they will disagree. If we know Siskel and Ebert's votes then we know whether they disagreed, so it is useless to ask their friend whether he thinks they will disagree. But, if we do not know their votes or only know one vote, asking their friend would give us a better estimate of whether they disagreed than random guessing.
N	Y	N	A weakly relevant attribute where S' is nonempty. *e.g.*, say their friend can only predict whether they will agree if he knows Ebert's vote, and when no votes are known he just guesses randomly. Then if we know no votes, the friend's prediction is irrelevant, but if we know one vote, the prediction is relevant.

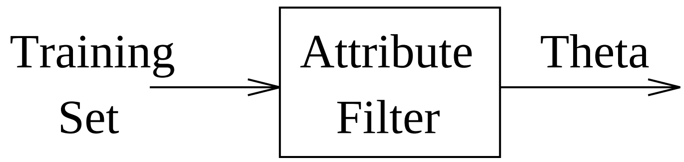

Figure 4.1: The attribute filter method, in which the attributes are filtered without regard to the mining algorithm to be used.

on the true underlying distribution of the data, they cannot be directly applied to a real database. Instead we would have to estimate the probabilities from the data, and define tolerance thresholds for the equality tests. Even if we did this, testing attributes for relevance could be extremely time-consuming. Recall that an attribute X_1 is strongly relevant if there exists a value assignment y for Y, x_1 for X_1, and s for S (where S is the set of all attributes except X_1) such that $p(Y = y|X_1 = x_1, S = s) \neq p(Y = y|S = s)$. To use this definition to show for example that X_1 is not strongly relevant would require searching through *all* possible assignments s, x_1, and y, and this would take time $O(k^m)$ where m is the number of attributes and k is the number of values per attribute. As a method for selecting an attribute subset, the use of definitions of relevance is a filter approach, since it ignores the data mining algorithm that will be used. Some filter methods with better running time requirements are described in the next section.

4.4 Filter Methods

As the name would imply, filter methods for attribute subset selection act as a filter in the data mining process, to remove possibly irrelevant attributes before the data mining algorithm is run. Figure 4.1 shows a diagram of how filter methods fit into the process: they are independent of the data mining process that came before or will come after them and simply analyze the data to determine which attributes might be relevant. We review three incarnations of the filter method: FOCUS, Relief, and tree-filtering.

The FOCUS algorithm (Almuallim & Dietterich 1991), originally defined for noise-free Boolean domains, exhaustively examines all subsets of attributes, selecting the minimal subset of attributes that is sufficient to determine the class. Its authors referred to this as the MIN-FEATURES bias. As discussed in Section 4.3, FOCUS finds strongly relevant features, but there are cases where weakly relevant features should also be included. Also, since the algorithm uses an exhaustive search and the space it is searching is exponential in the number of attributes, its running time becomes prohibitively large as the number of attributes grows.

Cardie (1993) and Kibler & Aha (1987) use classification tree algorithms to filter attributes. Cardie uses C4.5 while Kibler & Aha used ID3 to select "only the relevant" attributes in a database. Any attributes not used by the tree were deemed irrelevant, and the rest were relevant. As we showed in the introduction to this chapter, even the C4.5 algorithm can be confused by irrelevant or relevant but harmful attributes, so this filter will not be perfect. Obviously, using algorithm $\mathcal{A}$ as a filter will have no effect if we also use $\mathcal{A}$ as the data mining algorithm. But there are some simple algorithms whose performance can be seriously degraded by the presence of many irrelevant attributes, and the worst such algorithm is the nearest-neighbor classifier. Cardie and Kibler & Aha all focused on improving the nearest-neighbor algorithm by using only the subset of attributes that appeared in the classification tree built by C4.5 or ID3, respectively. The resulting performance of the nearest-neighbor classifier was higher than with the entire set of attributes.

The Relief algorithm (Kira & Rendell 1992*a*, Kira & Rendell 1992*b*) is more generally applicable than the tree filter methods and less computationally intensive than the FOCUS method. Relief is a randomized algorithm that samples records randomly from the training set and updates the relevance weight of each attribute based on the difference between the selected record and the two nearest records of the same and opposite class (the "near hit" and "near miss"). If a record and its near hit have different values for attribute X_i, the relevance weight of X_i is reduced. The weight of X_i is increased if a record and its near miss have different values for X_i. We used 0 as the threshold for relevance, so that $\widehat{\theta}_i = 1$ iff `weight[i]`> 0. The papers describing the algorithm considered only strongly relevant and irrelevant attributes, and show that most of the time Relief was able to distinguish between the two. Relief does not take advantage of situations where the size of the attribute subset may be reduced without harming prediction accuracy. If two attributes were exact copies, it would assign the same weight to each.

In preliminary experiments we found significant variance in the relevance rankings given by Relief, due to its internal randomness. Since Relief randomly samples records and their neighbors from the training set, the answers it gives are unreliable without a large number of samples, on the order of two to three times the number of points in the training set. We were worried by this variance, and implemented a deterministic version of Relief that uses all records and all near hits and near misses of each record. This gives the results one would expect from Relief if run for an infinite amount of time, but requires only as much time as the standard Relief algorithm with the number of samples equal to the size of the training set. Since we are no longer worried by high variance, we call this deterministic variant RelieveD. This new algorithm worked well on several small artificial problems, so we chose it as a representative of the filter algorithms in our experiments.

The disadvantage of the filter approach is that it totally ignores the effects of the selected attribute subset on the performance of the data mining algorithm. As a trivial example, consider a buggy mining algorithm that returns useless patterns whenever the fifth input attributes is included (regardless of the database being used). Every filter method would be unable to adapt, and half the time would produce an attribute subset that caused the data mining algorithm to produce worthless patterns.

We claim that, to find a useful subset of attributes, the subset selection method must consider its context in the data mining process, specifically the data mining algorithm being used and the objective function used to evaluate the results. This motivated us to consider the the wrapper approach, which does employ such information.

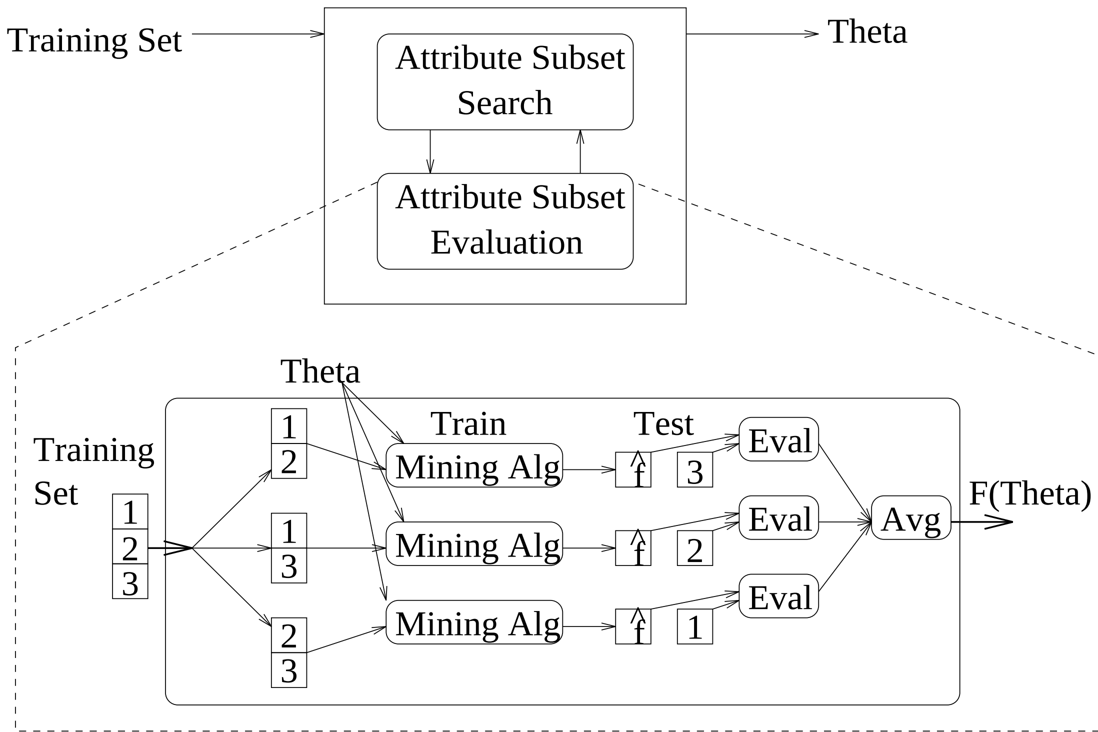

Figure 4.2: The **wrapper** method, shown with expanded view of evaluation procedure using 3-fold cross validation. The mining algorithm is used as a "black box" by the subset selection algorithm.

4.5 The Wrapper Method

In the wrapper method, the attribute subset selection algorithm exists as a wrapper around the data mining algorithm and result evaluation (Figure 4.2). The attribute subset selection algorithm conducts a search for a good subset using the mining algorithm itself as part of the function that evaluates an attribute subset.

Since we are focusing on classification problems in this chapter, our evaluation of the discovered classification model will be an estimate of its accuracy. Given a subset of attributes, to estimate the accuracy of the model discovered by using only the given attributes, we use cross validation. Table 4.2 shows the wrapper method with a detailed view of the subset evaluation mechanism. The essential characteristic of

Table 4.3: The backward elimination algorithm.

```
BACKWARD ELIMINATION(Theta, F, epsilon)
 Theta = <1 1 1 ... 1>
 repeat{
   BestF = -Infinity;  BestAtt = 0;
   For i = 1 to m
     If Theta[i] = 1 then
       Theta' = Theta;
       Theta'[i] = 0;
       If F(Theta') > BestF then
         BestF = F(Theta');
         BestAtt = i;
   If NewValue > F(Theta) - epsilon then
     Theta[BestAtt] = 0;
   else BestAtt = 0;
 } until BestAtt = 0;
 return Theta;
```

the wrapper is that the evaluation mechanism uses the same data mining algorithm and result evaluation method that will be used later in the data mining process. We chose to use cross-validation in our experiments but other methods that also use the same data mining algorithm and result evaluation method, such as the holdout set method, would also work. Note that no knowledge of the induction algorithm is necessary, except the ability to test the resulting patterns on the validation sets.

Finding a good subset of attributes under some measure requires searching the space of attribute subsets. Many common AI search algorithms may be employed for this task, and some have been suggested in the statistics literature under various assumptions about the induction algorithm (see Section 4.7). These assumptions do not hold for most machine learning algorithms, hence heuristic search is used.

One simple greedy algorithm, called *backward elimination*, starts with the full set of attributes and repeatedly greedily removes the single attribute that results in the best patterns (Table 4.3). The algorithm stops when removing any attribute would degrade the estimated performance by more than ϵ, a parameter that must be specified by the user. Setting $\epsilon = 0$ ensures that the final θ returned by the algorithm has the highest estimated F value of any attribute subset considered during the search. Setting $\epsilon > 0$ allows the algorithm to return an attribute subset that may be worse than the best subset considered, but by no more than ϵ times the number of attributes in the best subset minus the number in the returned subset. In other words, $\epsilon > 0$ (for small ϵ) gives slight preference to small attribute subsets.

A similar algorithm, called *forward selection*, starts with the empty set of attributes, and greedily adds attributes. The algorithm is exactly the same as the backward elimination algorithm in Table 4.3, except that θ is initially set to all zeroes, the algorithm repeatedly tries setting θ_i to 1 instead of 0, and ϵ is usually set slightly negative. A small negative value makes the algorithm give slight preference to small attribute subsets.

The algorithms can be extended by considering both addition of an attribute and deletion of an attribute at each step. For example, during backward elimination, also consider adding each one of the deleted attributes and see if it improves performance. Thus at each step the algorithm greedily either adds or deletes an attribute. The only difference between the backward and forward versions is that the backward version starts with all attributes and the forward version starts with no attributes. These algorithms have been described in many statistics books (Draper & Smith 1981, Neter, Wasserman & Kutner 1990) under the names *backward stepwise elimination* and *forward stepwise selection*, respectively

The backward elimination and forward selection heuristic searches increase the overall running time of the data mining algorithm by a multiplicative factor of $O(m^2)$ in the worst case, where m is the number of attributes. This is because the algorithms iterate through the `repeat` loop at most m times, and each step is $O(m)$. Note that the increase in running time is independent of n, the number of records, which is fortunate. In the worst case, the stepwise algorithms increase the running time by a multiplicative factor of $O(m2^m)$, since the total number of states is 2^m and exploring the options at each state takes $O(m)$ time. In practice, the algorithm usually stopped after exploring less than $2m$ states.

4.6 Experimental Evaluation

In order to evaluate the attribute subset selection using the proposed wrapper method, we ran experiments on nine databases using C4.5 and ID3 (without chi-squared pruning).[1] Since it does no pruning, ID3 should be more sensitive to varying attribute subsets and thus might be a better medium for the comparison of filter and wrapper approaches. On the artificial databases, we used the "-s -m1" C4.5 flags, which indicate that subset splits may be used and that splitting should continue until purity (*i.e.*, continue splitting as long as the training records at a node are of mixed class). Inside the wrapper, to estimate the accuracy for attribute subsets we used 25-fold cross validation on the training set. Thus our attribute subsets were evaluated solely on the basis of the training data without using data from the test set. Only after the best attribute subset was chosen by our algorithm did we use the test set to give the results appearing in this section.

To understand how relevant attributes relate to the optimal attribute subset, we used several artificial databases in the experiments in this chapter: CorrAL, Monks3, and Parity5+5. The artificial datasets allow us to control the presence of irrelevant, weakly relevant, and strongly relevant attributes, so that we may see how well the filter and wrapper methods do in a variety of situations.

CorrAL (John et al. 1994). Let attributes $A0, A1, B0, B1, Irr, Corr$ be Boolean. The target function is $Y = (A0 \wedge A1) \vee (B0 \wedge B1)$. Irr is an irrelevant attribute and $Corr$ is correlated with Y such that $Corr = Y$ 75% of the time. Since the problem has a high Correlation between one Attribute and the Label, the database is named "CorrAL." The correlated attribute is weakly relevant.

Monk3 (Thrun et al. 1991). The database has six attributes whose domains contain between two and four values. The Monk3 database has 5% noise in the training set but is otherwise a simple Boolean function with three strongly relevant input attributes and three irrelevant attributes.

---

[1]For the ID3 results we ran the C4.5 algorithm and used the unpruned trees. This is reasonable since the main difference between ID3 and C4.5 is the pruning method.

Table 4.4: Accuracy, size, and number of attributes used for ID3, ID3 with forward stepwise selection, and ID3 with backward stepwise elimination.

Database	Accuracy			Size			# Atts		
	ID3	ID3-F	ID3-B	ID3	ID3-F	ID3-B	ID3	ID3-F	ID3-B
Credit	80.5	79.5	81	117	65	97	15	4	13
Labor	82.4	82.4	82.4	12	7	12	16	3	12
Vote	94.8	95.6	97	25	13	37	16	3	15
Parity5+5	50.8	50	100	109	13	63	10	3	5

Parity 5+5 . The target concept is the parity of five bits, $Y = 1$ if $\sum_{i=1}^{5} x_i$ is odd, otherwise $Y = 0$ when the sum is even. The database contains ten attributes, five weakly relevant and five irrelevant attributes. The training set contained 100 records, and all 1024 possible records were used in the test set.

Tables 4.4 and 4.5 summarize our results. We give details for those databases that had the largest differences either in accuracy or tree size. We also ran experiments on the Iris, Thyroid, and Monk1 databases. The results on these databases were similar to those reported here.

Table 4.4 shows that backward stepwise elimination resulted in the best (highest accuracy) patterns. Only backward elimination was able to select the correct set of five weakly relevant features in the Parity5+5 database. Note that on the Vote database, backward elimination removed one attribute, and the resulting tree was much larger than that built by the ID3 algorithm with no subset selection. This can happen if ID3 had chosen the removed attribute as a split in the tree, but as a result of that split became confused and was unable to identify the correct features as relevant. Backward elimination generally removed only a few attributes before stopping, which probably indicates that it was stopping too soon, since forward selection achieved nearly the same accuracy with very few features.

Forward stepwise selection used far fewer attributes and resulted in smaller trees, but performed worse than the original ID3 algorithm on the Credit and Parity5+5 databases. It may be that, like backward elimination, forward selection was stopping too soon and not using enough attributes. Still, it is likely to be highly useful in helping a domain expert to understand the database better, by highlighting only a

Table 4.5: Accuracy, size, and number of attributes used for C4.5, C4.5 with forward stepwise selection, C4.5 with backward stepwise elimination, and C4.5 with the RelieveD filter. (Monk3 results are averages over ten runs with randomly selected training sets.)

Database		C4.5	C4.5-F	C4.5-B	C4.5-R
Credit	Accuracy	80	81.5	80	82
	Size	44	16	44	41
	# Atts	15	3	14	14
Vote	Accuracy	97	97	97	97
	Size	7	4	7	7
	# Atts	16	1	15	15
CorrAL	Accuracy	81.2	81.2	100	81.2
	Size	11	5	13	5
	# Atts	6	2	5	5
Monk3	Accuracy	98.6	97.5	98.6	97.5
	Size	7.8	12.9	7.8	6
	# Atts	6	3.3	2	2.4

handful of attributes as the most important.

The results in Table 4.5 are harder to interpret, as we suspected might be the case, since C4.5 uses pruning and thus is more likely to remove irrelevant attributes from the tree by itself. Again, forward selection should be a useful tool since it can focus the attention of the domain expert on the most relevant variables. Compared with RelieveD, it selects far fewer attributes on three of the four domains, and gives about the same accuracy. Backward elimination almost always selects the same number of features as RelieveD, but sometimes gives much greater accuracy. For example, on the CorrAL database, RelieveD only discarded the irrelevant feature (how could it have known that the relevant correlated feature would be harmful?), while backward elimination discarded the correlated feature. Figure 4.3 shows the tree built by C4.5 using no feature selection on the CorrAL database. The left subtree is the correct classification tree, which is learned if the "correlated" attribute is removed from the data. Although this is an artificial database, it does show clearly the power of the wrapper method, which can tune the set of features to get the best results out of the data mining algorithm being used.

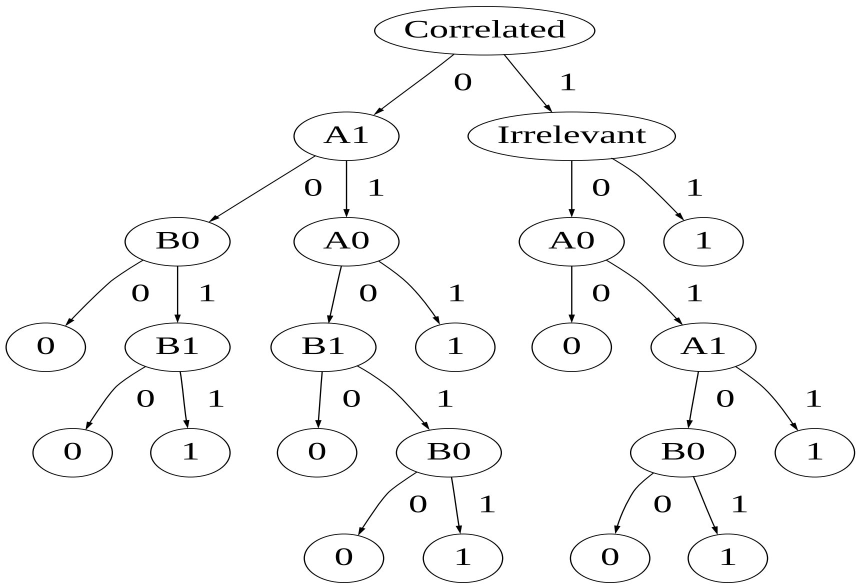

Figure 4.3: An example where ID3, C4.5, and CART pick a harmful weakly relevant attribute (correlated) for the root and an irrelevant attribute (irrelevant).

One possible cause for the lack-luster results on some of the databases is the subset evaluation mechanism in the wrapper. We observed high variance in the 25-fold cross-validation estimates of the error. Since our algorithms depend on cross-validation to choose which attribute to add or remove, a single "optimistic" 25-CV estimate caused premature stopping in many cases. Such an estimate of low error could not be improved and the algorithm stopped. Figure 4.4 shows a histogram of the 10-fold cross validation estimates of accuracy from 100 different cross validation runs on the Pima Indian diabetes database. The only difference between the runs of CV is that the database records are partitioned differently into 10 partitions. This suggests that, when using CV in the wrapper to compare different attribute subsets, we should use the same partitioning for every subset evaluation. We did not do this in this chapter, and as a result the variance due to randomized partitioning might have negatively affected our results in those cases where the difference in accuracy between two subsets was small relative to the difference in accuracy resulting from variance in the random cross validation partitioning. Note that in cases where the difference in

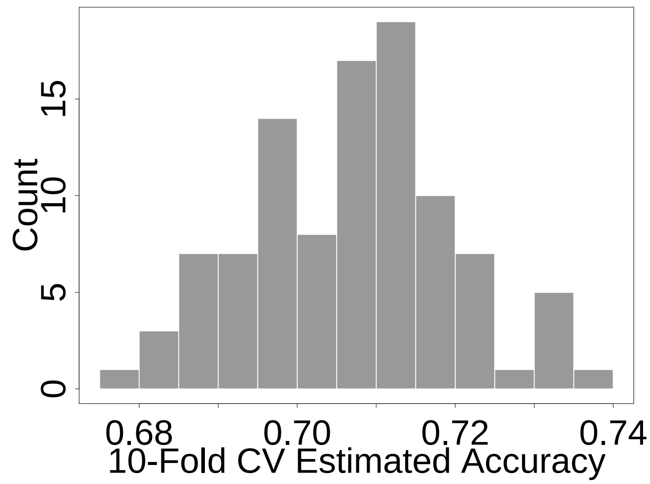

Figure 4.4: Histogram of cross-validation estimates of accuracy in the Pima Diabetes database (100 runs).

accuracy between subsets was large, Parity5+5 and CorrAL, the backward elimination wrapper identified the optimal attribute subset.

4.7 Related and Future Work

Researchers in statistics (Boyce, Farhi & Weischedel 1974, Narendra & Fukunaga 1977, Draper & Smith 1981, Miller 1990, Neter et al. 1990) and pattern recognition (Devijver & Kittler 1982, Ben-Bassat 1982) have investigated the attribute subset selection problem for decades, but most work has concentrated on subset selection using linear regression.

Sequential backward elimination, sometimes called sequential backward selection, was introduced in Marill & Green (1963). Kittler generalized the different variants including forward methods, stepwise methods, and "plus ℓ–take away r." Branch and bound algorithms were introduced by Narendra & Fukunaga (1977). Finally, more recent papers attempt to use AI techniques, such as beam search and bidirectional search (Siedlecki & Sklansky 1988), best-first search (Xu, Yan & Chang 1989), and genetic algorithms (Vafaie & De Jong 1992).

Many measures other than cross validation have been suggested to evaluate an attribute subset, such as adjusted mean squared error, adjusted multiple correlation coefficient, and the C_p statistic (Mallows 1973). Mucciardi & Gose (1971) empirically compared seven different techniques for subset selection on a nine-class electrocardiographic problem.

The search for the best subset can be improved by making assumptions about the evaluation function. The most common assumption is monotonicity—that increasing the number of attributes in the subset can only increase the performance. Under such assumptions, the search space can be pruned by the use of dynamic programming and branch-and-bound techniques. However, the monotonicity assumption is not valid for many induction algorithms used in machine learning (see for example Table 4.1).

The terms *weak* and *strong* relevance are used in Levy (1993) to denote formulas that appear in one minimal derivation or all minimal derivations, respectively. We found the analog for attribute subset selection helpful. Moret (1982) defines *redundant features* and *indispensable features* for the discrete case. The definitions are similar to our notions of weak relevance and strong relevance, but do not coincide on some boundary cases.

In the machine learning literature, the most closely related algorithms are FOCUS and Relief, which we have described. Littlestone (1988) introduced the WINNOW family of algorithms that efficiently learn linear threshold functions with many irrelevant attributes in the mistake bound model and in Valiant's PAC model.

Recently the machine learning community has shown increasing interest in this topic. Moore & Lee (1994*b*) present a set of efficient algorithms to "race" competing subsets until one outperforms all others, thus avoiding the computation involved in fully evaluating each subset. Their method is an example of the wrapper method using a memory-based (instance-based) algorithm as the induction engine, and leave-one-out cross validation (LOOCV) as the subset evaluation function. Searching for attribute subsets is done using backward and forward hill-climbing techniques similar to ours, but they also present a new method—schemata search—that seems to provide a four-fold speedup in some cases.

Langley & Sage (1994*b*) have also recently used leave-one-out cross validation in a nearest-neighbor algorithm, whereas Langley & Sage (1994*a*) show that a wrapper approach to feature selection can improve the accuracy of the simple Bayesian classifier by removing harmful attributes. Simple Bayes assumes that attributes are conditionally independent given the class, and when this is not the case it can be advantageous to discard even highly relevant features.

Kohavi & John (forthcoming) also discusses this problem, and thoroughly investigates the wrapper method for attribute selection, including the use of identical partitions in cross validation as well as some important speedups to the algorithm. Results in this paper show that, on the natural databases studied, on average the wrapper method performs best, using no subset selection is second best, and the RelieveD filter is the worst method.

Caruana & Freitag (1994) also present methods to quickly search over attribute subsets. They test the forward and backward stepwise methods on a calendar apprentice domain, using the wrapper method and a variant of ID3 as the induction engine. They introduce a caching scheme to save evaluations of subsets, which speeds up the search quite a bit, but this idea seems to decision tree induction.

Finally, Skalak (1994) uses the wrapper method for attribute subset selection and for decreasing the number of prototypes stored in instance-based methods. He shows that this can sometimes increase prediction accuracy.

4.8 Future Work

We included the results for forward and backward search methods separately to illustrate the different biases of the two greedy strategies, but one can easily imagine combining the two methods to achieve the best behavior of both. In the simplest approach, we could run both methods separately and select the best of the two results, based on our evaluation method. This should yield the same positive results as forward search in most cases while retaining the reasonable behavior of backward search for problems with high attribute interaction, such as parity5+5.

Note that even the better of the backward and forward results should not be taken as the best performance possible from the wrapper method. More comprehensive search strategies could search a larger portion of the search space and might yield improved performance. Obvious extensions would be to perform bi-directional beam search or to search all the way to the other side (*e.g.*, to the full set of attributes for the forward direction). The attribute relevance rankings produced by RelieveD could also be used to create a set of initial states in the space from which to search.

In the real world, sometimes it may cost time or money to measure the value of a feature (*e.g.*, a doctor may need to order a test, or a marketer may need to send out a survey). Even worse, the cost of an attribute may not be fixed, but may depend on which other attributes are measured. For example, if a doctor is already sending out blood samples for a standard lab analysis, the incremental cost of performing some more unusual blood tests will be less than had the basic analysis not already been ordered. Since the wrapper method always maps a *set* of attributes to an estimate of performance, we may easily couple it with complicated attribute costs as long as they also map a set of attributes to a cost. Given these, we can either optimize for estimated performance, and simply report the cost of the attributes, or we can optimize for some combination of accuracy and attribute cost.

4.9 Conclusion

Attribute subset selection is an important data engineering step that has many ramifications. Our introductory example (Table 4.1) and experimental results show that common classification algorithms such as C4.5 can fail to ignore harmful attributes—those attributes whose inclusion in the training set causes the mining algorithm to discover lower-quality patterns than if they had been removed from the training set.

We defined three categories of attribute relevance in order to clarify our understanding of the attribute selection problem. We advocated the wrapper method as a means of identifying useful attribute subsets and tested two greedy search heuristics—forward stepwise selection and backward stepwise elimination—using cross validation to evaluate performance.

Our results show that the wrapper methods can significantly improve accuracy in some domains, greatly outperforming the basic mining algorithm with no subset selection and also outperforming the RelieveD filter method. In other domains the wrapper methods produce about the same or slightly worse results than the filter method, and we have outlined future work to investigate the causes. The forward selection method frequently discovers subsets of only a handful of attributes that still allow discovery of high quality patterns. Discovering which attributes are most important is extremely useful in many data mining studies, so forward selection should be a useful tool.

Chapter 5

Algorithm Engineering: Parameter Tuning

In this chapter, we address the problem of finding the parameter settings that will result in optimal performance of a given mining algorithm using a particular database as training data. We again use a wrapper method, considering determination of the best parameters as a discrete function optimization problem. The method uses best-first search and cross-validation to wrap around the basic induction algorithm: the best-first search explores the space of parameter values, running the mining algorithm many times on training and holdout sets produced by cross-validation to get an estimate of the expected error of each parameter setting. Thus, the final selected parameter settings are tuned for the specific induction algorithm and database being studied. We report experiments with this method on 33 databases selected from the UCI and StatLog collections using C4.5 as the basic induction algorithm. At a 90% confidence level, our method improves the performance of C4.5 on nine domains, degrades performance on one, and is statistically indistinguishable from C4.5 on the rest. On the sample of databases used for comparison, our method yields an average 13% relative decrease in error rate. We expect that other data mining algorithms may be similarly improved by the addition of automatic parameter tuning.

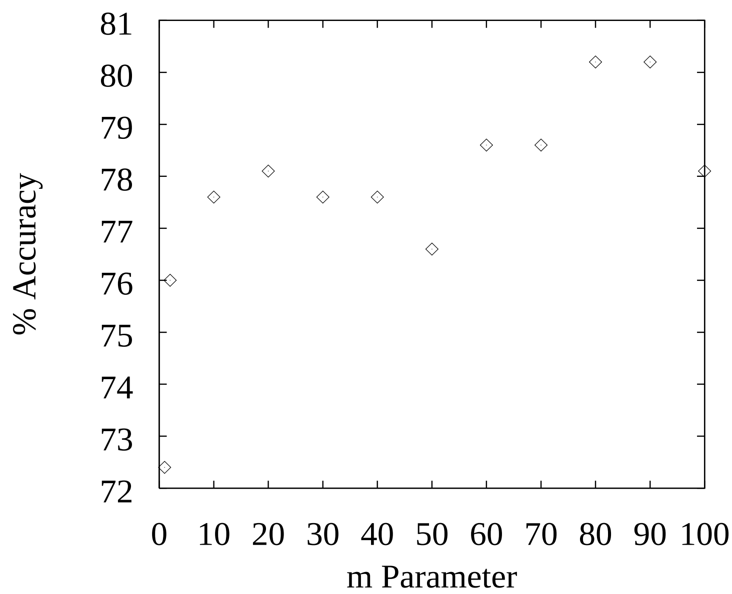

Figure 5.1: Example of the importance of parameter tuning, showing test-set accuracy on the Pima Indian diabetes database as a function of C4.5's MINOBJS parameter.

5.1 Introduction

The algorithm engineering step in the data mining process involves tuning the mining algorithm to improve its performance on the problem at hand. Often this means writing problem-specific programs to convert the output of the data mining algorithm into more useful results, but in this chapter we will focus on domain-independent methods for automatically tuning the parameters of a mining algorithm to discover the best patterns.

Although the problem of selecting a data mining algorithm for a particular task is recognized as an important issue (Brazdil, Gama & Henery 1994, Schaffer 1993), the problem of finding the best parameter values has not been systematically studied. Nearly all mining algorithms have parameters that a user can set, and the values of these parameters can have a large effect on the quality of the discovered patterns. As shown in Figure 5.1, C4.5's test-set accuracy on the Pima Indian diabetes database varies from a low of 72.4% to a high of 80.2% as one of its parameters varies from 1 to 100. Such a sizeable difference indicates that parameter tuning is an important problem.

Selecting an algorithm and finding the best parameters for a particular algorithm are equivalent in the abstract, but the latter involves optimization over a much larger (possibly continuous) space, one that cannot be completely explored within reasonable time limits. Thus, issues of heuristic search arise that are not present in the algorithm selection problem.

Authors of data mining algorithms typically give some rules of thumb regarding the setting of their algorithm's parameters, or they supply default parameter settings that they expect to work well on the problems of interest to the user. Although such heuristics may fare reasonably well on a variety of databases, a better strategy would be to determine the best parameter values for the particular database under analysis. An algorithm for parameter tuning should be able to make use of the information in the training data to guide it in a search for the best parameters. This idea motivated us to use the wrapper method of Chapter 4, which does employ such information. Just as in attribute subset selection, the wrapper method's search for a good parameter setting uses the induction algorithm itself as part of the evaluation function.

In Section 5.2, we formally define the problem we wish to solve: finding the parameters that maximize the quality of the discovered patterns. Section 5.3 discusses cross-validation and best-first search, the two components of this chapter's wrapper method. Section 5.4 gives an example of our approach, describing how automatic parameter tuning can be applied to C4.5. Section 5.5 discusses our experimental methodology and presents results on C4.5. Section 5.6 discusses related work in machine learning and statistics. Finally, Section 5.7 gives our conclusions and suggests directions for further work on this problem.

5.2 The Parameter Selection Problem

As in Section 4.2, the objective of a data mining algorithm $\mathcal{A}$ is to take a training set T and discover a set of patterns P such that P optimizes some objective function $F(P)$ that assigns some real-valued measure of goodness to P. In this chapter, rather than being concerned with the effect of the choice of attribute subset on P, we are concerned with the choice of parameters given to $\mathcal{A}$. Let us again use θ to describe a point in the space we are searching, but now θ will be a vector of parameter settings for algorithm $\mathcal{A}$. Thus we can think of the algorithm $\mathcal{A}$ as a function that takes T and θ as inputs and produces the patterns P as output. We define the best parameter settings θ^* in Definition 5.1.

Definition 5.1 (Optimal Parameter Vector)

$$\theta^* = \overset{argmax}{\theta} F\left(\mathcal{A}(T, \theta)\right) \quad . \tag{5.1}$$

As in the previous chapter, finding θ^* is impossible in general because we cannot directly evaluate most objective functions; we do not know the true underlying probability distributions that describe the data we have seen in the training database and the data we will see in the future. Again, even if we could calculate F directly we cannot afford to explore the entire parameter space Θ in most cases.

In the remainder of the chapter we will focus on classification problems, discussing first a general wrapper method for estimating $F(\mathcal{A}(\mathcal{T}, \theta))$ and heuristically searching Θ, then describing specific instantiations of these general methods to work with the C4.5 classification tree algorithm.

5.3 The Wrapper Method for Parameter Selection

The wrapper method for parameter selection works exactly the same as the wrapper method for attribute selection, shown in Figure 4.2, except that the search and evaluation are both done on vectors of parameters settings instead of attribute subsets. Recall that the wrapper requires two components: a search component and an evaluation component. The search component repeatedly suggests parameter settings. The evaluation component evaluates these settings by running the induction algorithm and getting an estimate of the quality of the discovered patterns. In this section we describe the evaluation and search algorithms used.

5.3.1 Error Estimation by Cross Validation

As in the previous chapter (Section 4.5), we use cross validation to evaluate θ. There are three differences in how we use cross validation in this chapter. The least important difference is that we use 10-fold cross validation instead of 25-fold, since 10-fold only takes 40% of the time and a number of informal experiments suggested that it worked just as well. A more important difference in this chapter is that, when comparing θ_i and θ_j, cross validation's partitioning of training records into ten subsets is the same, so that a difference in their evaluations should be due only to the effects the different parameter settings have on the algorithm, and not due to random differences in the partitioning. The third change deals with the remaining source of variance in cross validation estimates.

One problem with using cross-validation as an error estimate for parameter selection is that it has high variance. Thus, we must use some care in using it to select a parameter setting, so that we do not choose one parameter setting over another without being reasonably confident. If the variance of the estimated value of a parameter setting is too high (the variance of the value returned by cross validation is the variance of the ten samples, divided by ten) then we re-run cross validation with a different random partitioning, which should roughly halve the variance.

5.3.2 Optimization using Best-First Search

The task of finding the best value for each parameter of a given learning algorithm is a function optimization problem. We chose to view the problem as state-space search, and selected the best-first search algorithm, which works by repeatedly expanding the most promising unexpanded state (Nilsson 1980). Table 5.1 describes the best-first search algorithm; note that this is much more than simple hill-climbing. Search problems can be characterized by five distinct components. We identify these components below, describing how each is involved in the general wrapper method for parameter tuning. The next section will return to these five components and specify exactly how we set them when using the parameter-tuning wrapper with C4.5.

The *state space* being searched is obviously Θ, the space of possible parameter

Table 5.1: The best-first search algorithm, modified for function optimization

```
Best-First-Search
   1. OPEN list <- {initial state}
      CLOSED list <- empty
      BEST <- initial state
   2. Let v = the state in OPEN with minimal f(v).  Remove v from
      OPEN, add v to CLOSED.
   3. if f(v) + epsilon < f(BEST),  BEST <- v
   4. Expand v: Apply all operators to v.  Evaluate all new states
      (those not already on the OPEN or CLOSED lists) and add to
      the OPEN list
   5. If no change in f(BEST) during the last k expansions,
      exit and return BEST
   6. Goto 2
```

settings. The *heuristic function* takes a parameter vector θ and evaluates it, giving a real number as output. The heuristic function f guides the search by providing a mechanism for ranking the states that have been discovered during the search, so that exploration can continue from the best state discovered so far. The heuristic function does not need to be the same as the *objective function*, which is the real function we are trying to optimize, but in our case cross-validation (as described in the preceding section) serves as the heuristic and objective function. States and *operators* form a graph with states represented by nodes and operators labeling the directed arcs between nodes. The operators in the parameter tuning wrapper will be specific to the data mining algorithm whose parameters are being tuned.

The search begins with the *initial state*, which must be specified, and ends when the *termination criterion* has been reached. Recall that we are trying to find θ^* (Definition 5.1), but even if we could evaluate the true objective function directly, it may be impossible to search the entire parameter space. Instead of searching exhaustively, the search stops and returns the current best state if some reasonably thorough further search in the space does not uncover a better state: when k consecutive expansions fail to yield a new state that is at least ϵ better than the current best

Table 5.2: Parameters to the C4.5 algorithm.

Name	Possible Values	Default Value	Description
MINOBJS (m)	$1 \ldots \infty$	2	Stopping parameter in tree construction. Halts the recursive partitioning process when no partition of the current node results in children all having weight $\geq m$. (*Weight* is equal to the number of records unless there are missing values.)
CONFIDENCE (c)	$[0, 100]$	25	Confidence level parameter in tree pruning. Small values of c cause much pruning, large values cause little pruning.
GAINRATIO (g)	on,off	off	Splitting criterion: information gain or gain ratio. When g is off, information gain is used as the splitting criterion in tree construction. When on, gain ratio is used.
SUBSETS (s)	on,off	off	Subset splits. When s is on, subset splits are considered during tree construction. When off, all splits on k-valued nominal variables result in k children.

state. All function optimization methods must specify some stopping criterion, and ours is similar to those used in numerical optimization (Press, Teukolsky, Vetterling & Flannery 1992).

5.4 Automatic Parameter Selection for C4.5

Up to this point we have specified the wrapper as much as possible independent of any learning algorithm. To make our ideas more concrete, we describe the application of our method to the C4.5 algorithm for building classification trees. The rest of this section details choices we made that were specific to using the parameter tuning wrapper with C4.5.

For the state space, we chose to automatically set all of the C4.5 tree-building parameters (m, c, g, and s), which appear in Table 5.2. For example, a parameter vector θ might be $(10, 25, \text{off}, \text{on})$, indicating $m = 10, c = 25, g = \text{off}$ (use gain-ratio), and $s = \text{on}$ (use subset tests). For the m and c parameters we only considered integer values $1-5, 10, 15, 20, 25, 30, 40, \ldots, 100$. This decision was based on our observations that fine granularity was not important for large values of the parameters. Thus, there are a total of 1156 states.

For the heuristic function, we used 10-fold cross-validation as the estimate of the expected error for each state. When the estimated standard deviation of err_{CV} is larger than 1%, we run cross-validation again, up to a maximum of three times. We used a 10% trimmed mean, first suggested for use in this context by John (1994*a*); that is, if we run 10-fold CV three times, then out of the 30 resulting estimates we remove the lowest and highest three and take the average of the rest.

There are ten operators that may be applied to any state. One operator toggles the value of the s parameter and another toggles the value of g. For each of the numeric parameters, there are operators for moving up or down circularly either one or five positions in the array of possible values. (That is, from a current state where $m = 3$ we might try $m = 2, m = 4, m = 80, m = 20$.) In our first version we only moved up or down one position, but noticed the algorithm was getting trapped in local minima. We added the option of moving five steps and moving circularly to allow escape from local minima. (We tried no other step sizes.) This gives a total of ten operators.

The initial state used was the default setting for C4.5: $\theta = \langle 2, 25, \text{off}, \text{off} \rangle$. For the termination condition, we used the condition described in Section 5.3.2, setting k to 5 and ϵ to 0.1%. This completely specifies the parameter tuning wrapper as used with C4.5. In the next section, we compare the tuned C4.5 algorithm with the original to see if we obtain any improvement in accuracy.

5.5 Experiments with Tuning C4.5

Our hypothesis is that, given some learning algorithm $\mathcal{A}$ that has parameters, we can use our wrapper method to create an $\mathcal{A}$-AP algorithm (with Automatic Parameter tuning using the wrapper method) that performs better than $\mathcal{A}$ (using its default parameter setting)on real-world domains.[1] We first describe our methodology for testing this hypothesis and then present the results and our analysis.

5.5.1 Experimental Method

To test our hypothesis, we compared the accuracy of the C4.5 and C4.5-AP (C4.5 with automatic parameter tuning) algorithms on 33 databases gathered from the UCI (Murphy & Aha 1994) and StatLog (Michie et al. 1994) collections. The databases represent all of the available StatLog databases except the Shuttle database (which was too large), all of the UCI databases used by Holte (1993), all of the Monks databases (Thrun et al. 1991), and Corral, which is the artificial database defined in Chapter 4.

For some databases, a single test set was specified by the contributors. In such cases we used the holdout method to evaluate C4.5 and C4.5-AP so that results would be comparable to those reported in the literature. For all other databases, we used 10-fold cross-validation to evaluate C4.5 and C4.5-AP. As mentioned above, we used the same partitioning for both algorithms so that the ten resulting performance numbers for C4.5 and C4.5-AP are pairwise comparable. At this point it is best to consider the C4.5-AP algorithm as a black box; although it uses cross-validation itself to set its own parameters, the outer cross-validation estimates we report in this section are completely separate from the inner cross-validation. We are now interested only in comparing the performance of the two algorithms.

[1]However, as noted by Schaffer (1994) and Wolpert (1994), we cannot hope to improve $\mathcal{A}$ over *all* domains.

Table 5.3: Experimental results on natural databases.

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

5.5.2 Experimental Results

Table 5.3 gives 10-fold CV estimates of the accuracies for C4.5 and C4.5-AP on the natural domains. Table 5.4 gives results for those databases with a specified test set. Both tables also give results for the C4.5* "algorithm." C4.5* results were obtained by modifying the C4.5-AP algorithm: instead of using cross-validation as the evaluation method, it "cheats" and uses holdout error on the same test set that

Table 5.4: Experimental results on artificial and large natural databases.

Database	Train/Test Size	Accuracy Default C4.5	Accuracy C4.5-AP	Accuracy C4.5*	$P(t)$
corral	32/129	81.2±3.44	100.0±0.00	100.0±0.00	1.0 ✓
DNA	2000/1186	92.3±0.77	93.2±0.73	93.2±0.73	.80
letter	15000/5000	86.8±0.48	87.0±0.48	87.1±0.48	.62
monk1	124/432	75.7±2.06	100.0±0.00	100.0±0.00	1.0 ✓
monk2	169/432	65.0±2.29	62.5±2.33	82.4±1.83	.22
monk3	122/432	97.2±0.79	97.2±0.79	100.0±0.00	.50
satimage	4435/2000	85.2±0.79	86.1±0.77	85.2±0.79	.79

we later use to evaluate its performance. The C4.5* results might never be achieved in practice. The point is to show the limits of our approach—even if we were to use the world's best error estimation method instead of cross-validation, we could never surpass the C4.5* results. We believe that best-first search conducts a reasonably thorough search of the space, and therefore we conjecture that the results of C4.5* cannot be significantly improved upon by any settings of C4.5 parameters.

For each database in Table 5.3, we used a one-tailed paired t statistic (Casella & Berger 1990) to test the hypothesis that the accuracy of the C4.5-AP algorithm is higher than the C4.5 algorithm (versus the null hypothesis that the algorithms perform equally). When we run 10-fold cross-validation, we get ten accuracy estimates that we average to give the final estimated accuracy. In our experiments we treated each of the ten accuracy estimates as samples from a population, and used these as the estimates to be paired in the t test. The $P(t)$ values in the table should be interpreted as supporting our hypothesis at the $P(t)$ confidence level; for example, when $P(t) \geq .95$ we may state with 95% confidence that C4.5-AP has higher accuracy than C4.5.

From these tests we conclude with 90% confidence that the C4.5-AP algorithm outperforms the C4.5 algorithm on nine of the 33 databases, and is outperformed by C4.5 on a single database. At a 95% confidence level, C4.5-AP outperforms C4.5 on six databases, and is never outperformed by C4.5. In the one case where C4.5-AP loses at a 90% confidence level, the labor-negotiation database, note that the entire database is very small, with only 57 records. Because of the small size, the state evaluations in C4.5-AP had high variance, and the search did not find good parameter values.

With a significance of 0.90 in the t test, if the algorithms were equivalent in their prediction performance, one would expect three or four significant results, about two for each algorithm. Tables 5.3 and 5.4 show ten significant results, nine indicating the performance of C4.5-AP is superior and only one indicating inferior performance. We therefore conclude that C4.5-AP is indeed significantly superior.

C4.5-AP is often significantly outperformed by C4.5*, so it is important to consider the possible reasons. C4.5-AP and C4.5* are exactly the same except that C4.5* "cheats" and uses the test set for evaluation and thus its parameters are tuned to fit that single test set later used for evaluation. In many cases, C4.5* may be overfitting the test set, so that if we were to later generate an independent sample of data from the same distribution, it might not outperform C4.5-AP by as much, and it may even be worse. In other words, the C4.5* accuracy estimates are optimistically biased, so C4.5* is probably not as much better than C4.5-AP as the results would imply.

Upon analyzing the parameters chosen by C4.5*, we found that a wide range of values for the c parameter yielded the same accuracy. The 15%–35% range was most common (ten databases). Significantly, the value 25% (the default value in C4.5) was chosen for all databases except tic-tac-toe (where C4.5-AP halved the error rate of C4.5) and glass2 (where C4.5-AP improved absolute accuracy by over 4%). For the m parameter, 1 was often the best choice or there was a tie among 1 through 5. Although this tends to create a larger tree that then just gets pruned back, C4.5 sometimes prunes by replacing a node's test with the test at one of its children, so perhaps m=1 gives more latitude in the pruning phase. Information gain (turning the g parameter on) was a big winner on several databases: vehicle, segment, hypothyroid, heart, and cleve. Turning on the s parameter helped in tic-tac-toe and monk1.

These observations suggest the possibility that perhaps $m = 1$ is a better default value than $m = 2$, and begs the question as to whether C4.5 with this new default parameter setting would be a more worthy competitor to C4.5-AP. We have run C4.5 with the m parameter set to 1 on the same databases used above and have gotten worse performance on average as compared with $m = 2$ (83.8% accuracy for $m = 1$ vs. 84.4% for $m = 2$); thus, although $m = 1$ yields significant improvements in some cases, it hurts on others and the improvement cannot be attributed to this single parameter.

The time required for C4.5-AP can be large: letting $O(A)$ be the running time of $\mathcal{A}$, the time required to run k-fold cross-validation t times is $O(tkA)$. We must do this for each state investigated by our search algorithm, so if we evaluate n nodes in the search, the total time is $O(ntkA)$. In our experiments, t was limited to three, k was 10, and the number of nodes expanded was about 60. For example, on the DNA database (which took the most time) C4.5 took just under 2 minutes, while C4.5-AP took 6.8 hours.

5.6 Related Work

In statistics, *model selection* refers to the general problem of selecting a learning algorithm $\mathcal{A}$ for a particular task (Linhart & Zucchini 1986). Though the problems are equivalent in the abstract, model selection typically refers to the choice of one from a small set of algorithms, while different issues arise in parameter selection because of the large space to be searched. Nevertheless, because of the parallels between the two tasks, we might expect to observe some of the same phenomena in parameter selection as encountered in model selection. Schaffer (1993) explored the use of cross-validation for selecting a mining algorithm, and concluded that the selected algorithm was sometimes much better and sometimes worse than any given algorithm, but never much worse. Our results support this claim—the use of cross-validation in the C4.5-AP algorithm resulted in improvements on many of the databases tested.

Selecting values for important parameters is a widely-studied problem in statistics. Breiman et al. (1984) describe the CART program, which is the prime example of automatic parameter setting (of a cost-complexity parameter) in classification tree induction. CART uses 10-fold cross-validation to set this parameter. There are still many parameters left to be set by the user, however, and it would be interesting to compare a fully automated CART to the standard CART. Nearly all statistical methods of regression contain a single *smoothing parameter* λ (similar to the cost-complexity parameter in CART and the `m` and `c` parameters in C4.5), which attempts to address the bias-variance dilemma: how to trade off fit to the training data with some measure of "complexity" of the model. Wahba (1990, Chapter 4) discusses cross-validation, generalized cross-validation, maximum-likelihood, and Bayesian methods of choosing λ in the context of smoothing splines. Quinlan (1993) discusses the importance of the C4.5 parameters, and suggests that the user manually perform a search through the space of parameter values using cross-validation to evaluate each parameter. Craven & Shavlik (1993) used this method in their work, but did not report the accuracy improvement over the default parameters nor the specifics of how the search was conducted. John (1994*a*) reports preliminary results on using cross-validation and exhaustive search to set the `m` parameter in C4.5.

Moore, Hill & Johnson (1992) discuss a method for using leave-one-out cross-validation to select many parameters in a nearest-neighbor setting. In the same scenario, Moore & Lee (1994*a*) give efficient algorithms for approximating the leave-one-out estimates, evaluating each model only partially until it becomes clear that either it is or is not the best model. Our heuristic of re-running cross-validation when the standard deviation of the estimate of the mean accuracy is above 1% is a rough rule of thumb along the same lines. Gasser, Kneip & Kohler (1991) also consider variance in err_{CV} in the context of the use of "plug-in" estimators in kernel regression, giving better bandwidth parameter selection than cross-validation.

In the neural network community, issues of model selection and parameter selection also arise. Since neural nets generally take much longer to train than classification trees, it is not feasible to use the wrapper method as reported here. Utans & Moody (1991) give a lucid description of the model and parameter selection problem in neural networks. They also present experiments using cross-validation to select the number of hidden units to use in the network and the number of weights to prune after training, using a clever method to speed up cross-validation. Lang, Waibel & Hinton (1990) use a holdout set to decide when to stop training. Weigend (1994) and Finnoff, Hergert & Zimmerman (1993) present methods for indirectly selecting a model parameter (the *effective* network complexity) using a holdout set.

5.7 Conclusion

We have presented a general method for setting the parameters of a data mining algorithm: the wrapper method. Using best-first search and cross-validation, we have applied the wrapper method to the C4.5 algorithm, yielding a version of C4.5 we call C4.5-AP that selects its own tree-building parameters automatically. We ran experiments on 33 databases from the UCI and StatLog collections and concluded that, at a 95% confidence level, our method improved C4.5 in six domains and had no significant effect in the remainder. At a 90% confidence level, our method improved C4.5 in nine domains, degraded the performance of C4.5 on one very small database, and had no effect on the rest. On the large sample of databases we used to compare C4.5 and C4.5-AP, C4.5-AP's error rate improved C4.5's by a relative factor of 13%.

Our results support our earlier claims that tuning the parameters of a learning algorithm for a specific database is better than just using the default parameters for all databases. We believe that this is an under-appreciated component of the algorithm engineering process. Future data mining applications should incorporate a similar approach to tuning an algorithm's parameters for improved results.

Chapter 6

Flexible Bayesian Classifiers

The simple Bayesian classifier is an important tool in data mining. As discussed in Chapter 2, its reliance on relatively simple summary statistics makes the algorithm quite fast and amenable to combination with the strong SQL query model. Additionally, since the model has clear probabilistic semantics, the models can be presented and interpreted in a straightforward fashion. There are two main assumptions made by the simple Bayesian classifier: the conditional independence assumption, which assumes that the effect of a variable on the class is independent of the values of other variables, and the normality assumption, which assumes that, given the class value, the distribution of each continuous attribute is Gaussian. In this chapter we argue that the good properties of the algorithm derive primarily from the first assumption, but the normality assumption can be removed at little expense either in running time or interpretability. We argue for the use of kernel density estimation, a more flexible method for modeling a continuous probability density, and we present the flexible simple Bayes algorithm that incorporates this scheme. We show empirically that the new method performs better in most cases and identify when it does not, and we prove that it is strongly pointwise consistent, a desirable statistical property.

6.1 Introduction

In data mining, since we often are analyzing extremely large databases, and since at the end of the day we need to explain the discovered patterns to domain experts who are not necessarily experts in data mining, the speed of an algorithm and the interpretability of its results are key properties. The *simple Bayesian classifier* possesses speed and interpretability in spades. The only problem with the algorithm is that it can perform poorly in some domains where its simplifying assumptions, which help it to be fast and interpretable, become a liability. What simple Bayes lacks is the flexibility to discover patterns that depend on the interaction between two or more input attributes and the class, and the ability to model any continuous distribution other than a simple Gaussian.

Overcoming the first limitation while retaining speed and interpretability is difficult, and is the focus of a large part of the Bayesian network research community. Also, several machine learning researchers have built modified versions of the simple Bayesian classifier that could handle more complex patterns, either by using attribute subset selection methods described in Chapter4 (Langley & Sage 1994*a*), or by creating new attributes that allow more complex patterns to be represented (Pazzani 1995).

These lines of research are important, but there remains the second limitation of simple Bayes, the Gaussian assumption. In fact, overcoming this limitation while retaining speed and interpretability is actually quite possible, and forms the basis of the *flexible Bayesian classifier* algorithm described in this chapter. In empirical tests on UCI databases, we find that flexible Bayes achieves significantly higher accuracy than simple Bayes on many domains. Theoretically, we show that one can prove the strong pointwise consistency of flexible Bayes with fewer assumptions than are required for simple Bayes.

In the pages that follow we review SIMPLE BAYES, the simple Bayesian classifier, then describe FLEXIBLE BAYES, an extension that eschews the single Gaussian assumption in favor of kernel density estimation (but which retains the independence assumption). We next discuss some important properties of kernel estimation that

Flexible Bayes inherits. After this, we present some hypotheses about the new method's behavior, followed by experiments with natural and artificial domains designed to test those hypotheses. At the end of the chapter, we review some related work and suggest some directions for future research on this family of classifiers.

6.2 The Simple Bayesian Classifier

As we have mentioned, the simple Bayesian classifier provides a simple approach, with clear semantics, to representing, using, and learning probabilistic knowledge. The method is designed to mine for predictive patterns, in which the performance goal is to accurately predict the class of test records and in which the training records include class information.

The mechanics of Simple Bayes were discussed already in Chapter 2, so we will not repeat them in detail here, but we will review the equations involved. Let C be the random variable denoting the class of a record and let $\mathbf{X}$ be a vector of random variables denoting the observed attribute values. Further, let c represent a particular class label, and let $\mathbf{x}$ represent a particular observed attribute value vector. Given a test case $\mathbf{x}$ to classify, one simply uses Bayes' rule to compute the probability of each class given the vector of observed values for the predictive attributes:

$$\begin{aligned} p(C = c|\mathbf{X} = \mathbf{x}) &= \frac{p(C = c)p(\mathbf{X} = \mathbf{x}|C = c)}{p(\mathbf{X} = \mathbf{x})} && (6.1)\\ &= \frac{p(C = c)\prod_i p(X_i = x_i|C = c)}{p(\mathbf{X} = \mathbf{x})}\ , && (6.2)\\ &\quad \text{by the conditional independence assumption.} \end{aligned}$$

Recall from Chapter 2 that one does not directly estimate the distribution in the denominator of Equation 6.1, as it is just a normalizing factor. Instead one ignores the denominator and then normalizes so that the sum of $p(C = c|\mathbf{X} = \mathbf{x})$ over all classes is one.

For discrete attributes, $p(X = x|C = c)$ is estimated from the training data using maximum likelihood estimation. For numeric attributes, $p(X = x|C = c)$ is typically modeled as a Gaussian distribution, and the mean and standard deviation are estimated using maximum likelihood estimation. For continuous attributes we can write

$$\begin{aligned} p(X=x|C=c) &= g(x;\mu_c,\sigma_c)\ , \quad \text{where} && (6.3)\\ g(x;\mu,\sigma) &= \frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}}\ , && (6.4)\end{aligned}$$

the probability density function for a normal (or Gaussian) distribution. Equation 6.3 is not strictly correct: the probability that a real-valued random variable exactly equals any value is zero. Instead we speak about the variable lying within some interval: $p(x \leq X \leq x+\Delta) = \int_x^{x+\Delta} g(x;\mu,\sigma)dx$. By the definition of a derivative, $\lim_{\Delta\to 0} p(x \leq X \leq x+\Delta)/\Delta = g(x;\mu,\sigma)$. Thus for some very small constant Δ, $p(X=x) \approx g(x;\mu,\sigma)\times\Delta$. The factor Δ then appears in the numerator of Equation 6.1 for each class. They cancel out when we perform the normalization, so we may use Equation 6.3.

In summary, SIMPLE BAYES provides a simple and efficient approach to data mining for classification tasks. However, it typically relies on an assumption that the numeric attributes obey a Gaussian distribution, which may not hold for some domains. This suggests that we should explore less constrained methods for estimating continuous distributions.

6.3 Flexible Simple Bayes

We can now introduce the FLEXIBLE BAYES learning algorithm, which is exactly the same as SIMPLE BAYES in all respects but one: the method used for density estimation on continuous attributes. Although using a single Gaussian is the most common technique for handling continuous variables, it is certainly not the only one. Researchers have also explored a variety of methods for nonparametric density estimation.

We chose to investigate *kernel density estimation*. Recall that SIMPLE BAYES estimates the density of each continuous attribute as $p(X=x|C=c) = g(x,\mu_c,\sigma_c)$.

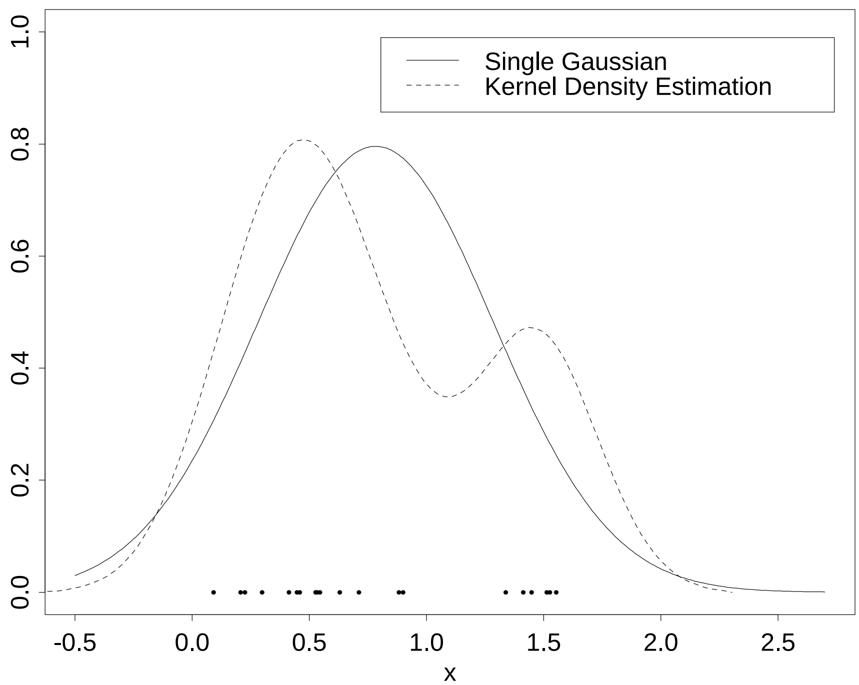

Figure 6.1: The effect of using a single Gaussian versus a kernel method to estimate the density of a continuous variable.

Kernel estimation with Gaussian kernels (one can use other kernel functions as well) looks much the same, except that the estimated density is averaged over a large set of kernels

$$p(X = x|C = c) = \frac{1}{n} \sum_i g(x, \mu_i, \sigma_c) \ , \tag{6.5}$$

where i ranges over the training points of attribute X in class c, and $\mu_i = x_i$. The dashed line in Figure 6.1 shows the kernel density estimation based on the 20 sampled points. Readers familiar with kernel methods should note that our Equation 6.5 is equivalent to the standard kernel density formula $p(X = x|C = c) = (nh)^{-1} \sum_j K(\frac{x-\mu_i}{h})$, where the bandwidth parameter $h = \sigma_c$ and the kernel shape $K = g(x, 0, 1)$.

Whereas in Simple Bayes one could estimate μ_c and σ_c by storing only the sum of the observed x's and the sum of their squares, the sufficient statistics for a normal distribution, Flexible Bayes must store every continuous attribute value it sees

Table 6.1: Algorithmic complexity for SIMPLE BAYES and FLEXIBLE BAYES, given n training records and k attributes.

	SIMPLE BAYES		FLEX BAYES	
Operation	Time	Space	Time	Space
Train on n records	$O(nk)$	$O(k)$	$O(nk)$	$O(nk)$
Test on 1 record	$O(k)$	$O(k)$	$O(nk)$	$O(k)$

during training. The only sufficient statistic for the list of μ_i's is the list of x_i's itself. (Nominal attributes' distributions are still learned by storing a single number per value that represents the sample frequency, as in SIMPLE BAYES.)

Interpretability and speed are still important considerations. Note that even though the algorithm must store many values instead of just the mean and standard deviation, the model is still easily interpreted by an expert through a visualization like Figure 6.1, which graphs the value of $p(x|c)$ over the domain of X. Regarding speed, when processing the training data the algorithm still only needs to make a single pass through the database, and only performs constant-time updating operations with the processing of each record. There is a time penalty when using FLEXIBLE BAYES, but it comes when applying the algorithm to new data. When computing $p(X = x|C = c)$ for a continuous attribute to classify an unseen test record, SIMPLE BAYES only had to evaluate g once, but FLEXIBLE BAYES must perform n evaluations, one per observed value of X in class c. This leads to some increase in the storage and computational complexity, as summarized in Table 6.1.

We have not yet addressed the most important issue in kernel density estimation—the setting of the bandwidth parameter σ_c. As we shall see in the next section, kernel estimation has some nice theoretical properties when σ_c shrinks to zero as the number of records goes to infinity. The statistical literatures reports various rules of thumb for setting the kernel width, but each heuristic makes implicit and explicit assumptions about the density function that will be true of some distributions and not others. In this chapter we set $\sigma_c = 1/\sqrt{n_c}$, where n_c is the number of training records observed with class c. Thus, as FLEXIBLE BAYES observes more training points, its density estimates become increasingly local. (Numeric variables are first scaled linearly so that the middle 80% of their values lie in the interval $[0, 1]$.)

The intuition behind Flexible Bayes is that kernel estimation will let the method perform much better in domains that violate the normality assumption, while usually performing as well as Simple Bayes in domains where it does hold. To understand this claim, we must review the theoretical underpinnings of kernel methods.

6.4 Theoretical Properties of Flexible Bayes

In general, density estimation involves approximating the probability density function of a continuous random variable. The Bayesian classifier encounters this problem whenever it must estimate $p(X|C)$ for some continuous attribute X. This is a general problem in statistics, and a variety of methods are available for solving it (Venables & Ripley 1994, Silverman 1986). In this section we discuss the theoretical properties of kernel density estimation and their implications for the Flexible Bayes algorithm.

Statisticians are principally concerned with the *consistency* of a density estimate (Izenman 1991).

Definition 6.1 (Strong Pointwise Consistency)
If f is a probability density function and $\hat{f}_n$ is an estimate of f based on n examples, then $\hat{f}_n$ is strongly pointwise consistent if $\hat{f} \rightarrow f(x)$ almost surely for all x; i.e., for every ϵ, $p(\lim_{n\rightarrow\infty} |\hat{f}_n(x) - f(x)| < \epsilon) = 1$.

The strongest asymptotic result we could hope for regarding Flexible Bayes would be that, provided the independence assumption holds, its estimate of $p(C|\mathbf{X})$ is strongly pointwise consistent. This would imply that, in the limit, as we get more and more training data, using the Flexible Bayes estimate of $p(C|\mathbf{X})$ for classification will converge to the Bayes optimal error rate. We will prove strong consistency of Flexible Bayes in three steps, first proving that the method provides a strongly consistent estimate of $p(X|C)$ when X is nominal, then proving the same property when X is continuous, then proving that the estimate $p(C|\mathbf{X})$ is strongly consistent.

Theorem 6.1 (Strong Consistency for Nominals)
Let $X_1, \ldots, X_n$ be an independent sample from a multinomial distribution with v values, where the probability of drawing value j is p_j. Let n_j be the number of samples of value j. Then n_j/n is a strongly consistent estimator of p_j.

Proof: This is a direct instantiation of the strong law of large numbers (Casella & Berger 1990). ∎

Theorem 6.2 (Strong Consistency for Reals)
Due to Devroye (1983). The kernel density estimate is strongly consistent when:

- *The kernel function K is a bona fide density estimate—it must be nonnegative for all x, and it must integrate to 1.*
- $h_n \to 0$ *as* $n \to \infty$. *Recall that h in the standard notation is equivalent to our* σ.
- $nh_n \to \infty$ *as* $n \to \infty$.

All of these conditions are satisfied by using Gaussian kernels with $h_n = 1/\sqrt{n}$, so each $p(X|C)$ density estimate in Flexible Bayes is strongly consistent.

Lemma 6.1 (Consistency of Products)
Let the functions $\hat{f}_1, \ldots, \hat{f}_k$ be strongly consistent estimates of density functions $f_1, \ldots, f_k$, respectively. Then $\prod_i \hat{f}_i$ is a strongly consistent estimator of $\prod_i f_i$.

Proof: We prove that $\hat{f}_1\hat{f}_2$ is a strongly consistent estimator of $f_1 f_2$, from which the lemma follows by induction. By the definition of strong consistency, for any ϵ_1 and ϵ_2, $p(\lim_{n\to\infty} |\hat{f}_{1,n} - f_1| < \epsilon_1) = 1$ for all x, and similarly for $\hat{f}_2$. For $\hat{f}_1\hat{f}_2$ to be strongly pointwise consistent, $p(\lim_{n\to\infty} |\hat{f}_{1,n}\hat{f}_{2,n} - f_1 f_2| < \epsilon_3) = 1$ must hold for all x, and this will be true whenever $\epsilon_1\hat{f}_2 + \epsilon_2\hat{f}_1 + \epsilon_1\epsilon_2 < \epsilon_3$ (by some simple algebraic manipulation). Since $\hat{f}_1(x)$ and $\hat{f}_2(x)$ are finite, and since ϵ_1 and ϵ_2 may be made arbitrarily small, the bound on ϵ_3 can be made to hold, giving the desired result that $\hat{f}_1\hat{f}_2$ is strongly consistent. ∎

Table 6.2: Information about the natural data bases used in the experiments.

Database	Size	#Class	#Numeric	#Nominal
Breast Cancer (Wisconsin)	699	2	10	0
Cleveland Heart Disease	303	2	6	7
Credit Card Applications	690	2	6	9
Glass Identification	214	7	9	0
Glass (float/non)	163	2	9	0
Horse Colic	368	2	7	15
Fisher's Iris	150	3	4	0
Canadian Labor Negotiation	57	2	8	16
Meta-Learning	528	2	19	3
Pima Diabetes	768	2	8	0
Vehicle Silhouettes	846	4	18	0

Theorem 6.3 (Consistency of FLEXIBLE BAYES)
Let the true conditional distribution of the class given the attributes be $p(C|\mathbf{X}) = \frac{\prod_i p(X_i=x_i|C=c)p(C)}{\prod_i p(X_i=x_i)}$. *(This is the actual conditional distribution, not our estimate.) Then the* FLEXIBLE BAYES *estimate* $\hat{p}(C=c|\mathbf{X}=\mathbf{x})$ *is a strongly consistent estimator of* $p(C=c|\mathbf{X}=\mathbf{x})$.

Proof: By Theorems 6.1 and 6.2, FLEXIBLE BAYES' estimates of $p(\mathbf{X}|C), p(X)$, and $p(C)$ are strongly consistent, thus by Lemma 6.1 and a related lemma regarding the quotient of strongly consistent estimates, FLEXIBLE BAYES' estimate of $p(C|X)$ is strongly consistent. ∎

6.5 Experimental Studies

We have made a number of claims regarding the expected behavior of FLEXIBLE BAYES, and we need to be test them empirically. As with other experiments in this thesis, we ran a set of experiments on real databases. In addition, since some hypotheses were left unanswered by the experiments on UCI data, we designed some artificial databases that let us systematically vary dimensions of interest.

Table 6.3: Ten-fold cross validation results, comparing SIMPLE BAYES and FLEXIBLE BAYES. Accuracies for C4.5 are shown to provide context.

Data set	SIMPLE	FLEX	FLEX Better?		C4.5
Breast Cancer (Wisc.)	95.9± 0.2	96.7± 0.2	✓	(99.0%)	95.4
Cleveland Heart Disease	83.3± 0.6	80.0± 0.6	×	(97.5%)	72.3
Credit Card Application	74.8± 0.5	78.3± 0.6			85.9
Glass	42.9± 1.7	66.2± 0.9	✓	(99.5%)	65.4
Glass2 (Float/Non)	61.9± 1.4	83.8± 0.7	✓	(99.5%)	70.6
Horse Colic	73.3± 0.9	69.7± 1.0	×	(99.5%)	85.1
Iris	96.0± 0.3	95.3± 0.4			95.3
Labor Negotiation	86.0± 1.3	84.0± 1.6			85.7
Meta-Learning	67.1± 0.6	76.5± 0.5	✓	(99.5%)	72.6
Pima Diabetes	75.1± 0.6	73.9± 0.5			71.6
Vehicle Silhouette	44.9± 0.6	61.5± 0.4	✓	(99.5%)	70.0

6.5.1 Experiments on Natural Data

To determine the relevance of our approach to real-world problems, we first selected 11 databases with continuous attributes from the UCI machine learning repository (Murphy & Aha 1994) and elsewhere. Table 6.3 summarizes the number of records, the number of classes, and the number of nominal and numeric attributes in each data set. Because a sizeable fraction of each domain's attributes were numeric, they seemed likely candidates for contrasting the behavior of FLEXIBLE BAYES and SIMPLE BAYES.

For each domain, we used ten-fold cross validation to evaluate the generalization accuracy of the two induction algorithms. That is, we randomly partitioned the data into ten disjoint sets, then provided each algorithm with nine of the sets as training data and used the remaining set as test cases. We repeated this process ten times using the different possible test sets and averaged the resulting accuracies. We also carried out this procedure with C4.5 (Quinlan 1993), a well-known algorithm for classification-tree induction, to provide a reference point for comparison.

Table 6.3 shows the results of the runs on natural domains, including the mean accuracy and standard deviation for each, whether one method was significantly better than the other on a paired t test, and the significance level of that test. The table

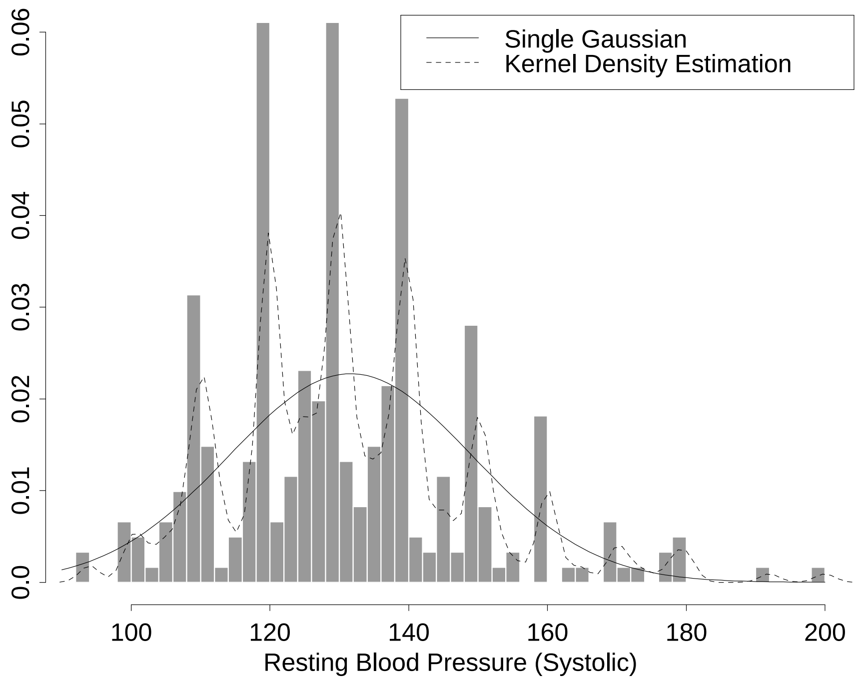

Figure 6.2: Systematic measurement errors in the Cleveland heart disease database.

indicates that Flexible Bayes was significantly more accurate than Simple Bayes in five of the 11 domains, less accurate in two domains, and not significantly different in four cases. In two domains (Glass2 and Meta-Learning), the simple scheme was significantly worse than C4.5, whereas the flexible method did significantly better.

Curiously, the two domains where Simple Bayes outperformed Flexible Bayes were both medical domains. Possibly doctors tend to define diseases such that important continuous attributes are roughly normal, given whether a patient has a disease. If the conditional densities truly are Gaussian, Simple Bayes should perform better than Flexible Bayes (in the small-sample scenario) since the normality assumption is correct. To investigate further, we plotted histogram estimates of the density of each continuous attribute in the Cleveland heart disease database, conditioned on the class. On visual inspection, we did find that the distributions were roughly normal.

We were also quite surprised to discover systematic measurement errors. Figure 6.2 shows a surprising phenomenon working against the Flexible Bayes algorithm: the blood pressure of patients is sometimes rounded to the nearest 10 (note

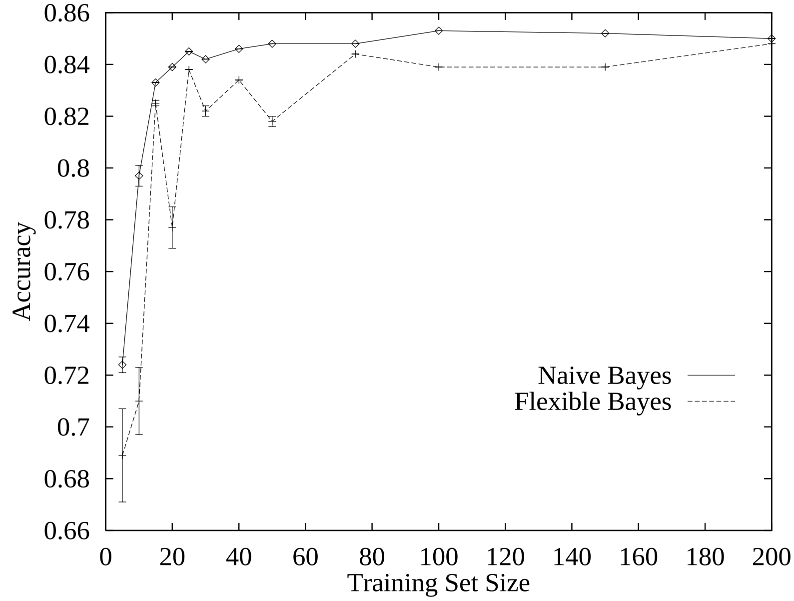

Figure 6.3: Learning curves testing the first hypothesis about behavior when the normality assumption holds.

the peaks at 120, 130, 140), and sometimes not. There is no medical reason to suspect that a human's blood pressure is likely to be a multiple of ten, but when using the standard mercury barometer and stethoscope apparatus for measuring blood pressure, it is difficult to get more than two significant figures, so rounding is common (Vida Tigrani, personal communication). The smooth curve shows the Gaussian density estimate, while the rough curve shows the kernel density estimate, which is confused by the rounding errors in recording the blood pressure of patients. We hypothesized that if this attribute were removed, Flexible Bayes and Simple Bayes would perform comparably. We ran a ten-fold cross-validation experiment with the blood pressure attribute removed, and found that both methods achieved an accuracy rate of 84.66%.

6.5.2 Artificial Domains

The positive results on the natural domains give us confidence in the usefulness of Flexible Bayes as a tool for data mining, but since the real domains are themselves

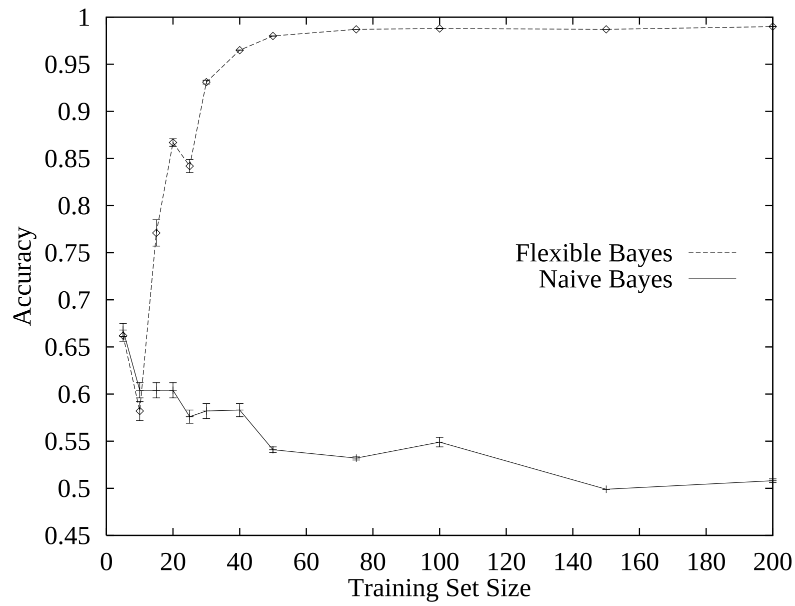

Figure 6.4: Learning curves for the second hypothesis about behavior when normality is violated.

poorly understood, scientifically they do not let us draw any hard conclusions. Several of our expectations could not be directly tested on the natural databases. These expectations were:

1. When the normality assumption holds (when each $p(X|C)$ is normal), Flexible Bayes and Simple Bayes should exhibit the same asymptotic performance, but Simple Bayes should reach asymptote faster (with smaller training sets).

2. When the independence assumption holds but the normality assumption does not, Flexible Bayes should reach the Bayes optimal error rate while Simple Bayes should not.

To test these hypotheses, we defined appropriate joint probability distributions and then independently sampled varying-sized training sets from these distributions in order to plot learning curves for the two algorithms. We constructed a test set containing one thousand records sampled from the same distribution, which we used to evaluate the models induced from the training sets. For each training set size, we constructed ten independent samples, and we report the average and standard deviation for the resulting ten runs.

To examine the first hypothesis, we defined a joint probability distribution over one continuous attribute and one Boolean class, such that the conditional distribution of the attribute given the class was Gaussian. The learning curves in Figure 6.3 support our hypothesis: although Flexible Bayes performs worse than Simple Bayes when the size of the training set is small, its performance approaches that of Simple Bayes as the training set grows.

To test the second hypothesis, we defined a joint probability distribution over one continuous attribute and one Boolean class, such that the conditional distribution of the attribute given the class was a mixture of two Gaussians. The learning curves in Figure 6.4 again support our hypothesis. Simple Bayes is helpless because it is learning with an overly parsimonious model, while Flexible Bayes' flexibility lets it fit this distribution as well. With a training set of 200 records, Flexible Bayes' accuracy was 99.0%. Using numerical integration, we estimated the Bayes optimal error rate in this domain to be 1.863%. The apparent impossibility of the situation (namely, Flexible Bayes outperforming the Bayes optimal error rate) is explained by realizing that the estimate of the algorithm's performance comes from a test set of 1000 records. We expect that, had we used an order of magnitude more test records, the algorithm's error would match the Bayes optimal error.

In summary, we have compared our Flexible Bayes algorithm with its natural benchmark, Simple Bayes, on a variety of natural and artificial domains, finding encouragement from the former and evidence for our hypotheses in the latter.

6.6 Discussion

Although our approach to Bayesian induction is novel, it does bear some similarities to other research in machine learning and statistics. The use of density estimation figures prominently in several learning algorithms. Specht & Romsdahl (1994) is the latest in a series of papers on kernel estimation in the guise of "probabilistic neural networks." In contrast to our approach, their method only handles continuous attributes, and makes no independence assumptions, so that a single d-dimensional

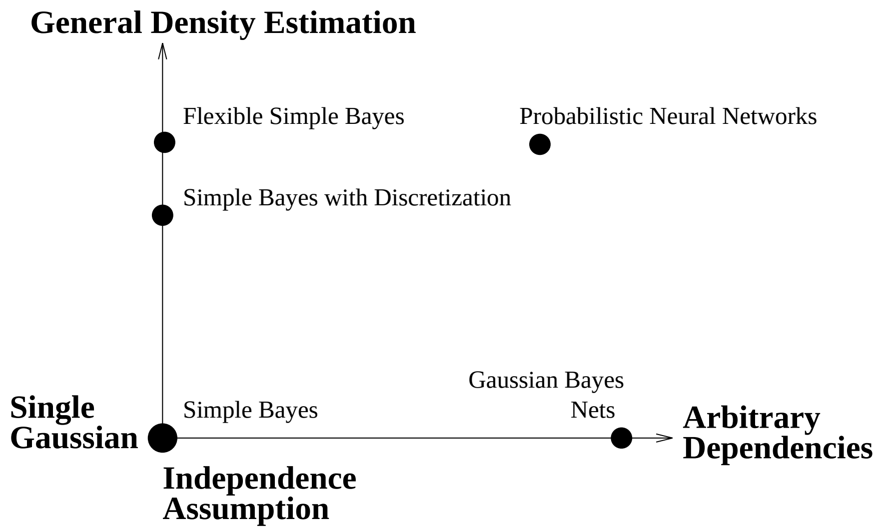

Figure 6.5: The space encompassing the algorithms discussed in this chapter.

density estimation is done per class, rather than d one-dimensional estimations. A novel attribute is their use of the conjugate gradient algorithm to optimize their cross-validation estimate of error over the space of smoothing parameters (one parameter per dimension). This gave tremendous improvement on some domains, and we expect that an adaptive kernel width would improve our results as well.

A histogram is one of the oldest and simplest methods of density estimation. Kononenko (1993) reports the use of experts to discretize continuous attributes, which specifies the locations and widths of the histogram bins. In contrast, Harper, Duewer, Kowalski & Fasching (1976) discuss automated construction of histograms, combined with smoothing methods which should result in a density estimate that looks more like a kernel estimate with a high bandwidth. Dougherty et al. (1995) study several methods for automatically discretizing continuous attributes based on statistical and information-theoretic metrics. They report that a simple Bayesian classifier combined with discretization gives higher accuracy than C4.5, averaged over many domains.

The algorithms explored in this chapter are but two samples from a large space of possible algorithms that our framework suggests. Figure 6.5 gives a perspective on the work discussed in this chapter, showing how Simple Bayes and Flexible Bayes relate to each other and to previous work.

The representational power of the flexible simple Bayesian classifier seems quite similar to that for generalized additive models of Hastie & Tibshirani (1990) for regression. Their method predicts the value of a continuous variable, given various nominal and numeric input variables, using

$$\hat{y} = f_0(\sum_i f_i(x_i)) \ ,$$

where the f_i are arbitrary (possibly nonlinear) functions. By taking logarithms, Equation 6.1 may be transformed into a generalized additive model. However, note that although the models are the same, their parameters are fitted very differently and thus the semantics of the parameters is different. In the FLEXIBLE BAYES case, parameters are fitted using maximum likelihood and the kernel density method, and we hope the end result will work well for classification. In the GAM case, an objective function such as classification accuracy of the model is defined and the parameters of the model are tuned to optimize this criterion. Thus, while one numeric parameter in the FLEXIBLE BAYES model may stand for the probability that $X = x$ given the class, the corresponding parameter in a FLEXIBLE BAYES-equivalent GAM model can be thought of as the value that *ought to be* the probability that $X = x$ given the class, in order for the model to optimize its objective function.

Two other approaches seem close to our method at first glance, but upon close inspection there are important differences. Geiger & Heckerman (1994) present a method for learning "Gaussian networks", which are Bayes nets with some continuous variables estimated by a single Gaussian distribution, in contrast to our approach in which densities are estimated by kernels. Kononenko (1993) uses discretization with a "fuzzy" modification. Rather than assigning the value of a continuous attribute to a single interval, Kononenko "fuzzifies" the value using a Gaussian and assigns it probabilistically to several intervals. The use of the Gaussian at first seems similar, but our approach uses such kernels only to obtain a smooth density estimate on the training data.

Other authors have investigated the use of kernel density estimation to improve the accuracy of other kinds of data mining algorithms. Smyth, Gray & Fayyad (1995) show how the accuracy of a classification tree algorithm may be improved by using kernel density estimation at the leaves of a tree, instead of just reporting the majority class from the training data.

Even though kernel density estimation resulted in higher accuracy in our experiments, the improvement comes at a computational price. In future research, instead of using kernels, we plan to experiment with density estimation using Gaussian mixture models fitted by the EM algorithm(Dempster, Laird & Rubin 1977). Now that our experiments with FLEXIBLE BAYES have shown that one can achieve significant improvements in accuracy by using more flexible density estimation, the next step is to find a way to reduce the storage requirements and application-time computational requirements. Gaussian mixtures have the same functional form as equation 6.5, but the μ_i range only over a small set (usually less than 20) of Gaussians, not every single data record.

6.7 Conclusion

In this chapter we reviewed the simple Bayesian classifier and the assumptions on which it relies, including the common use of a single Gaussian distribution for each predictive attribute. We argued that this assumption might be violated in some domains, and we proposed instead to use a kernel estimation method to approximate more complex distributions. Experiments with natural domains showed that, in a number of cases, this flexible Bayesian classifier generalizes better than the version that assumes a single Gaussian. Experimental studies with artificial data further suggest that the approach behaves as expected in two important scenarios. Although more work remains to be done, our results to date indicate that, because of its speed, accuracy, and interpretability, the flexible Bayesian classifier is a useful addition to a data mining toolbox.

Chapter 7

Trees: Greedy Attribute Selection

Classification trees are another important family of methods used in data mining. They are popular because of their speed, flexibility, and interpretability. As we discussed in Chapter 2, trees are a a nonparametric family of methods and as such can approximate a classification function arbitrarily well, given sufficient data. Their interpretability is a very important feature, as we discussed in Chapter 1. Their speed derives largely from the use of *greedy*, top-down methods used for building the trees.

All top-down classification tree induction algorithms are repeatedly faced with the problem of selecting a test to use to partition the database at a given node. Numerous greedy criteria have been proposed in recent years but they have seldom been carefully empirically compared with previous methods. In this chapter, we evaluate seven common attribute selection criteria and three new criteria on 24 domains taken from the UCI and Statlog database collections. The new splitting criteria proposed and tested include a method based on a pessimistic error estimate, a method for choosing a criterion using cross-validation, and a new kind of greedy method that employs geometric information. In a sense, the results both support and refute prevailing wisdom that the choice of criterion does not have a significant effect on accuracy—although no single method is a clear winner overall (there do seem to be some clear losers), accuracy can vary widely on a single domain depending on the selection criterion used.

7.1 Introduction

In top-down induction of classification trees, the tree-building process is driven by a mechanism for repeatedly selecting an attribute to test at a node in the tree. The selection process is usually a simple exhaustive search over the space of all attributes (or attributes and cutpoints in the case of numeric attributes), evaluating each candidate attribute, and finding the best. Thus all of the brains in such a tree-building algorithm lie in this evaluation function.

Although one might expect the choice of evaluation function to have an important effect on the accuracy of learned trees, prevailing wisdom holds that the attribute selection metric, or *splitting criterion*, has no significant effect on the accuracy of the induced trees (Mingers 1989, Breiman et al. 1984). But for Fayyad & Irani (1992), most recent work on splitting criteria involves purely theoretical attempts to address a problem noticed by Quinlan (1986); namely, that the standard information-gain criterion is biased towards selecting attributes with many values. Although this has resulted in interesting new splitting criteria with various provable properties (Zhou & Dillon 1991, López de Mántaras 1991) and statistical properties (White & Liu 1994), these new criteria have not been thoroughly tested experimentally on real databases. Buntine & Niblett (1992) present the only rigorous experimental treatment of splitting criteria, although Morgan & Messenger (1973) report a study in the same vein.

Since Buntine and Niblett were able to craft a careful empirical study to show that two splitting criteria did in fact produce trees with higher test-set accuracy than others, and since several new criteria have been proposed since their paper, it seemed worthwhile to perform similar careful experiments using the recently developed criteria. In this chapter we study seven existing splitting criteria: entropy, or information gain (Morgan & Messenger 1973, Quinlan 1986), gain ratio (Quinlan 1986), the Gini index of diversity (Morgan & Messenger 1973, Breiman et al. 1984), sum-minority (Heath, Kasif & Salzberg 1993, Morgan & Messenger 1973), López de Mántaras' distance function (López de Mántaras 1991), ORT (Fayyad & Irani 1992), and χ^2 (Quinlan 1986, White & Liu 1994).

Data mining research is not always a good spectator sport, so we have added our own entries into the fray, adding three new splitting criteria to the seven existing methods. The most interesting new criterion is based on the approach of defining the characteristics of a good partition in the multidimensional input space (rather than just looking at class frequencies), then looking for a split with those characteristics. The second method uses C4.5's (Quinlan 1993) pessimistic error estimate (used during pruning to estimate the expected error at a node) to choose splits resulting in low immediate estimated error. Another method treats the choice of splitting criterion as a parameter-tuning problem. Since there are only a few possible values of this parameter, we can use the wrapper method with exhaustive search to choose the splitting criterion. Whichever criterion results in lowest cross validation-estimated error is then used to build a tree using the entire training set.

We begin in Section 7.2 defining information-theoretic terms used in splitting criteria along with the seven existing criteria we used. We then define our new methods and motivate their development. Section 7.3 describes the experiments—the hypotheses we intended to test, our methodology, and the results. In Section 7.4 we discuss related work, and Section 7.5 presents our conclusions regarding splitting criteria.

7.2 Definitions of Splitting Criteria

Many of the methods for choosing an attribute are based on information theory. We will first give the pertinent definitions of information theory and then give the classification tree context. Table 7.1 gives the basic definitions of information theory (Cover & Thomas 1990). In this chapter, we will only be dealing with discrete random variables, so we use the discrete sum version of the definitions. In the table and in the following, $\sum_x$ should be read as the sum over all elements in the domain of X. Given a random variable $X \in \mathcal{X}$, $H(X)$ is a lower bound on the average length of the shortest description of X. That is, if one randomly picks a value x from $\mathcal{X}$ according to some distribution $p(X)$, one cannot *expect* (in the statistical sense) to communicate "I chose x" to someone else in fewer than $H(X)$ bits, even if they know $p(X)$.

Turning to the use of entropy and other measures in classification trees, let us use

Table 7.1: Basic definitions of information theory.

Symbol	Meaning
$p(X)$	A *probability distribution* on X. Similarly $p(Y)$.
$p(Y\|X)$	The *conditional* distribution of Y given X
$H(X)$	The *entropy* of the random variable X: $H(X) = -\sum_x p(x)\log_2 p(x)$
$H(X,Y)$	The *joint entropy* of the variables X and Y: $H(X,Y) = -\sum_x \sum_y p(x,y)\log p(x,y)$
$H(Y\|X)$	The *conditional entropy* of Y given X: $H(Y\mid X) = \sum_x p(x) H(Y\mid X=x) = -\sum_x \sum_y p(x,y)\log p(y\mid x) = H(X,Y) - H(X)$
$I(X;Y)$	The *mutual information* between Y and X, which is the *relative entropy* or *Kullback-Leibler distance* between the joint distribution and the product distribution: $I(X;Y) = \sum_x \sum_y p(x,y)\log \frac{p(x,y)}{p(x)p(y)} = H(X) + H(Y) - H(X,Y)$

A and C instead of X and Y to denote that we are making statements about the probabilities of attributes and classes. It is important to remember the context in which the decision of which attribute to select takes place: we are "at" a node during the classification tree construction. Some subset S of the original training set has filtered down to this node. We are now trying to evaluate how good it would be to split on attribute A. Let $n(a, c)$ be the number of records in S with attribute $A = a$ and class $C = c$. Similarly let $p(a, c)$ be the probability of drawing a record uniformly from S with $A = a, C = c$. Define $p(a), p(c), n(a)$, and $n(c)$ in the obvious way, and let $n = |S|$. This completely defines all statistics of the set S used by the common attribute selection criteria. For simplicity, in this notation we are directly treating only nominal attributes.

Classification trees commonly split on a continuous attribute X by defining a threshold t that partitions the domain of the attribute into two half-spaces: $\{X : X \leq t\}$ and $\{X : X > t\}$. We can imagine a dummy binary attribute A that takes on the value 0 when $X \leq t$ and 1 when $X > t$, and we can then use A instead of X in evaluating the split on X at t. Thus the treatment of nominal attributes is sufficient,

Table 7.2: The seven common attribute selection criteria, which evaluate a candidate split on a nominal attribute A.

Criterion	Definition
Information-Gain	$gain(A) = I(A;C) = H(C) - H(C\|A)$
Gain-Ratio	$gain(A)/IV(A) = I(A;C)/H(A)$
LdM Distance	$\frac{H(A\|C)+H(C\|A)}{H(A,C)} = 2 - \frac{H(A)+H(C)}{H(A,C)}$
Gini Index	$\sum_a p(a) \sum_{c_i \neq c_j} p(c_i\|a)p(c_j\|a)$
χ^2	$\sum_a \sum_c (np(a)p(c) - n(a,c))^2 / (np(a)p(c))$
Sum-Minority	$\sum_a n(a) - n(a, c_{\text{majority}})$
GORT	Let $n(a)$ be a vector such that $n(a)_i = n(a, c_i)$. $GORT =$ $\sum_{a_i \neq a_j} (p(a_i) + p(a_j)) (n(a_i).n(a_j)/\|n(a_i)\|\|n(a_j)\|)$

since the dummy variable A is itself nominal.

7.2.1 Existing Attribute Selection Criteria

We are now ready to define all seven of the common attribute selection criteria, and they are given in Table 7.2.[1] For each criterion, we first use notation corresponding to that used in the paper introducing it for use in classification trees, then we rewrite using the standard information theory notation above.

GORT is a generalization of the ORT (orthogonality) criterion proposed in Fayyad & Irani (1992). The orthogonality criterion was meant to be used only with binary trees. When applied to binary splits, the generalized criterion is equivalent to the original, but when applied to multi-way splits, GORT essentially looks at all pairs of children and applies the ORT criterion to each pair, then takes the weighted average of the results from each pair. We have chosen this generalization instead of using the original version directly as it was intended (*i.e.*, with the binary split restriction) so that in our experiment all factors are constant except for the splitting criterion. Although Fayyad & Irani (1992) claim that binary trees are better, we chose not to address the issue here.

[1] In the following, "LdM distance" shall refer to López de Mántaras' distance function.

Since we only care about the ranking that these criteria induce on a set of candidate splits, we are free to add or subtract constants or multiply by constants, in order to simplify the calculations required. We will use the notation $\overset{\text{rank}}{=}$ to denote criteria that rank attributes in the same order. For example, in the information gain criterion, since $H(C)$ is constant for all attributes, we can maximize $gain(A)$ by minimizing $H(C|A)$, the conditional entropy of the class given the attribute value. We can simplify the Gini index so that instead of minimizing the expression in Table 7.2, we maximize $\sum_{a,c} p(a)p(c|a)^2$:

$$\begin{aligned}
\sum_a p(a) \sum_{c_i \neq c_j} p(c_i|a)p(c_j|a) &= \sum_a p(a)\left(1 - \sum_c p(c|a)^2\right) \\
&= 1 - \sum_{a,c} p(a)p(c|a)^2 \\
&\overset{\text{rank}}{=} -\sum_{a,c} p(a)p(c|a)^2
\end{aligned}$$

We can simplify the Chi-Squared criterion so that instead of maximizing the expression in Table 7.2, we instead maximize $\sum_{a,c} \frac{p(a,c)^2}{p(a)p(c)}$:

$$\begin{aligned}
\sum_{a,c} \frac{(np(a)p(c) - n(a,c))^2}{np(a)p(c)} &= \sum_{a,c} \frac{(np(a)p(c))^2 - 2np(a)p(c)n(a,c) + n(a,c)^2}{np(a)p(c)} \\
&= \sum_{a,c} n - 2n + \frac{n(a,c)^2}{np(a)p(c)} \\
&= -n + n\sum_{a,c} \frac{p(a,c)^2}{p(a)p(c)} \\
&\overset{\text{rank}}{=} \sum_{a,c} \frac{p(a,c)^2}{p(a)p(c)}
\end{aligned}$$

Note how similar Chi-Squared and Gini are. Minimizing the Gini criterion is equivalent to maximizing $\sum_{a,c} p(a)p(c|a)^2$, which is equivalent to $\sum_{a,c} p(a,c)^2/p(a)$, which is almost the same as our final expression for Chi-Squared, $\sum_{a,c} \frac{p(a,c)^2}{p(a)p(c)}$, but for the factor of $p(c)$.

Quinlan (1986) introduced the machine learning community to information gain as a splitting criterion, although the rank-equivalent *entropy* criterion had been used earlier in statistics and pattern recognition (Sethi & Sarvarayudu 1982, Henrichon, Jr. & Fu 1969, Kanal 1979, Morgan & Messenger 1973). Moret (1982) gives an excellent historical account of work in classification tables, trees, and graphs, including early work incorporating information theory.

Since Quinlan (1986) pointed out a problem with information gain (its bias towards selecting many-valued attributes), several researchers have attempted to remedy this shortcoming. First was Quinlan himself in the same paper, proposing the gain-ratio criterion. More recently, López de Mántaras (1991) introduced his distance metric and proved that the distance does not decrease as granularity (the number of values of the splitting attribute) increases. Zhou & Dillon (1991) propose the "symmetric τ" function as a way to handle the granularity problem. Their function has exactly the same form as LdM distance, except that the logs are replaced with quadratic terms. White & Liu (1994) claim that gain-ratio and the LdM distance function are both flawed and advocate χ^2, which they show to be less biased via Monte Carlo experiments.

In recent work in a different spirit, Breiman (1996*b*) analyzed the Gini and entropy criteria in the setting where all possible binary partitions of a set of data into two subsets were possible and found that Gini will always make one subset out of the records representing the majority class and will put the rest of the classes into the other partition, while Entropy will partition the data into two subsets such that records of the same class are in the same subset, and the total number of records in each subset is as close to the same as possible. One implication Breiman mentions is that splits tend to be fairly well-determined for Gini, but for Entropy, the difference between the best and next-best split may be very small, so trees built using Entropy may be unstable in the sense that small perturbations in the training data can produce significantly different trees.

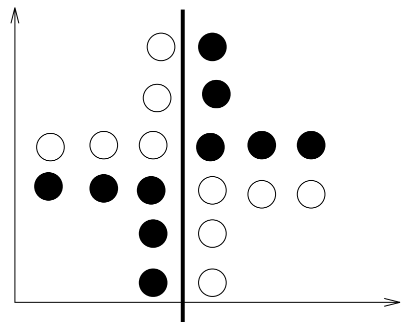

Figure 7.1: Motivation for the Δ splitting criterion.

7.2.2 New Attribute Selection Criteria

In this section we define three new splitting criteria. Perhaps the most novel is the Δ criterion, described next. The other two methods involve using a pessimistic estimate of error (as a correction to the overly optimistic sum-minority), and a wrapper (Section 4.5) method to choose from among many criteria using cross validation.

7.2.3 The Δ Criterion

All of the methods mentioned above use only the statistics mentioned, namely $p(a)$, $n(c)$, and so forth. There is rich geometric information in the data that is being ignored. The question is how to best use the information at our disposal (the complete information about all values of all attributes, not just the single attribute being considered and the class). The key insight is that a property of any good partition should be that the records near the partition on one side (call them the "left border" records) should be opposite in class from the "right border" records, as in Figure 7.1. We called the criterion based on this idea the Δ criterion, since Δ is often used to represent difference or change, and the key idea is that the points on one side of the partition should be different from the points on the other side. Even though the partition represented by the thick vertical line in Figure 7.1 looks awful from an information-theory perspective (the entropy of each partition is the same as the original set), it is clear that, because each point's nearest neighbor on the other side of the partition is of the opposite class, any tree *must* eventually divide the space here

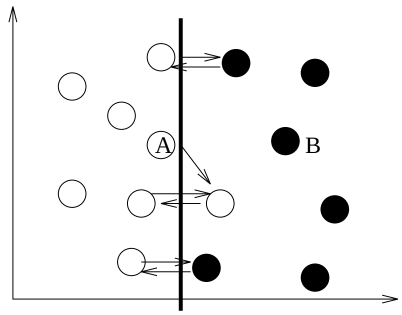

Figure 7.2: Labeling border points in the Δ criterion.

to fit the training data well, and the more well-defined the border, the more likely that it is a real pattern that will be found in the test data as well.

There are three main steps in calculating the Δ criterion:

1. Label all border points. Point A is a border point iff there exists some foreign point B (in the other partition) such that A is B's nearest foreign neighbor.

2. For each border point, locate its nearest foreign neighbor.

3. If the nearest foreign neighbor is of the same class, increment `matches`, otherwise increment `mismatches`.

Figure 7.2 shows the algorithm after step 2. Each border point has an arc pointing to its nearest neighbor across the border. (For example, point A was included in the left border because it is the nearest neighbor of point B.) Distance is calculated in a straightforward way. Given two records (vectors of attribute values) x, y, let d_i be the distance between them along attribute i. If attribute i is nominal,then $d_i = 0$ if $x[i] = y[i]$, 1 otherwise. If attribute i is numeric, $d_i = x[i] - y[i]$. (All numeric values are normalized by scaling linearly so that the middle 80% of values are in $[0, 1]$.) Then $d = \sqrt{\sum_i d_i^2}$.

We chose `mismatches/(mismatches + matches)` as the criterion function to maximize, since we want to maximize the percentage of mismatches. (We might also want to simultaneously maximize the total number of border points, but functions combining the two goals are not explored further here.) Note that the general idea is much broader than this final objective function. We might have used any function that makes use of the matches/mismatches statistics. Likewise, similar methods for counting matches and mismatches might work as well or better.

As Figure 7.1 would indicate, in developing this criterion we had in mind databases where the classes are fairly well-separated. When this does not hold, the ideal split imagined by the Δ criterion just does not exist in the database, and trees built using the criterion may perform poorly.

7.2.4 The Pessimistic Error Estimate Criterion

We next describe a modification of the sum-minority criterion. Sum-minority evaluates an attribute by essentially pretending that tree growing will stop immediately after partitioning the set of training examples on that attribute. If this were the case, then each child of the current node in the tree would be a leaf, and the number of errors the current node could make on the training set would be the sum of all minority-class cases in each leaf. Sum-minority adds these errors and returns the total (the sum of the minorities) as the badness of the split. As we shall see later, sum-minority was the worst splitting criterion tried, but the idea of estimating error is not all bad.

Quinlan (1993) describes a pruning method based on minimizing estimated error. Knowing that error on the training set is optimistically biased, Quinlan tries to correct for this bias in his pessimistic method for error estimation. The error of a node in a classification tree is estimated pessimistically, viewing the number of errors made by that node as a binomial distribution whose mean is the number of errors made on the training set. The estimated number of errors is taken as the upper 25% confidence limit of the distribution. This has the effect of significantly increasing the estimated number of errors over the training set estimate for small nodes (nodes built from only a few records) while leaving large nodes relatively unchanged.

The pessimistic error estimate criterion essentially uses sum-minority criterion, but rather than using the straightforward number of minority records at each node to estimate the number of errors, it instead uses C4.5's pessimistic error estimate. The goal in selecting an attribute would be to minimize this measure. The new criterion should perform better than sum-minority, since that criterion naively uses the training set estimates of error, which are known to be biased.

7.2.5 Cross Validation-Selected Splitting Criteria

Finally, note that the splitting criterion to be used can be viewed as a parameter to the classification tree algorithm. Following the *wrapper method*, we combine some search mechanism with some evaluation mechanism to automate part of the induction task. In experiments reported here we have augmented the basic C4.5 algorithm to include all splitting criteria previously discussed (except for Δ, which is too time-consuming to be included inside a cross-validation step). The *XV* attribute selection method uses 10-fold cross-validation repeated ten times to evaluate the accuracy resulting from using each splitting criterion, and then picks the one with the best estimated accuracy. On the ten repeats of cross validation, we use a 10% trimmed mean so that the final estimated accuracy ignores the best and worst runs and is an average of the remaining eight. We tried no other values for the number of folds or number of repeats. For large databases (over 2000 records) we used three repeats instead of ten.

7.3 Experiments

We had several goals in mind when designing our experiments, in the form of questions to answer. These included:

1. Does the splitting criterion make a difference?

2. Is information theory really the right tool for the job?

3. Is the ignorance of geometric information an impediment?

4. Can cross-validation be used to choose the best splitting criterion?

We describe the experimental methods used to answer these questions, and then present and discuss the results.

7.3.1 Methodology

We ran three separate experiments. The first was a comprehensive study designed to evaluate the accuracy of the various attribute selection measures on 24 databases from the UCI (Murphy & Aha 1994) and Statlog (Michie et al. 1994) collections. We used stratified 10-fold cross-validation to get a good estimate of the mean accuracy on each database. (Note that, although one splitting criterion itself uses cross-validation, this step happens within the outer cross-validation evaluation performed on all splitting criteria.) The databases are the same ones used in Chapter 5. All splitting criteria were implemented within C4.5 (Quinlan 1993), which we modified slightly by removing a test that prevents splitting when the criterion's value is below a threshold.

The second experiment was actually a component of the first—within the XV splitting criterion (which uses repeated cross-validation to select which one of the more basic methods to use) we recorded the number of times XV chose a particular splitting method as the best. XV's choice of criterion should be more reliable than our main experimental results since the main results were generated by one run of cross-validation while the XV runs repeated cross-validation ten times.

The third group of experiments test the performance of the Δ criterion on a parity target function with several irrelevant attributes, using learning curves. In the first case there are two relevant attributes, three irrelevant attributes and no noise. In the second case there are four relevant attributes, eight irrelevant attributes, and 20% noise. We felt that the use of geometric information in Δ should enable it to distinguish irrelevant attributes from relevant attributes using far fewer records than standard splitting criteria. The two experiments were designed to test this hypothesis on an easy problem (the first) and a more difficult problem (the second).

7.3.2 Experimental Results

Table 7.3 may be somewhat daunting. The main lesson of the table is that no single splitting method is the best on every database, not even the XV criterion. Since in many cases the accuracies of the basic splitting criteria are similar, variance overcomes the cross-validation accuracy estimates, and it is unable to choose the best. However,

Table 7.3: Accuracy estimates from 10-fold cross-validation.

Database	χ^2	Δ	Dist	Ent	Gini	GnR	Pess	Gort	$\sum$mn	XV
australian	85.7	84.4	86.1	83.9	85.7	85.1	85.4	82.2	84.7	83.9
breast-cancer	71.7	68.2	74.2	71.7	72.3	74.5	75.2	70.3	73.5	74.2
breast	95.4	95.7	95.3	95.1	95.6	95.0	95.7	95.4	95.3	95.1
chess	99.4	99.7	99.5	99.4	99.5	99.5	99.5	99.4	99.0	99.4
cleve	79.2	75.6	75.5	82.2	79.2	75.2	72.6	77.5	77.6	75.9
crx	85.4	86.1	84.9	84.4	85.5	84.8	83.3	84.5	84.8	84.7
diabetes	72.8	70.8	73.2	71.3	72.4	71.6	71.9	72.9	72.4	72.8
german	73.4	72.4	72.3	72.6	73.4	74.0	71.7	70.9	74.3	71.4
glass	69.6	63.3	68.7	70.1	71.4	67.8	70.5	70.5	70.9	70.0
glass2	81.0	77.9	78.6	80.4	81.0	76.5	80.4	75.5	77.3	77.9
heart	80.4	75.2	76.0	79.3	80.4	73.7	75.2	75.6	74.8	77.6
hepatitis	80.5	80.0	84.5	81.9	78.6	77.9	81.2	84.5	82.5	84.4
horse-colic	73.9	82.3	82.3	84.8	65.3	85.9	74.3	72.4	69.3	85.4
hypothyroid	99.0	98.8	99.0	99.8	99.0	99.1	98.9	99.1	98.5	98.9
iris	94.7	92.7	94.7	94.7	94.7	94.7	94.7	94.7	93.3	94.7
labor-neg	67.1	79.7	79.0	75.7	65.4	77.7	67.4	79.3	72.3	79.1
lymphography	71.4	79.1	72.2	72.9	72.2	76.9	75.6	68.9	72.2	73.6
pima	73.7	71.5	73.8	74.6	73.4	71.7	72.3	72.0	73.3	73.9
segment	96.7	95.2	96.5	96.8	96.6	96.4	95.8	96.3	95.9	96.2
sick-euthyroid	97.8	97.9	97.7	97.6	97.4	97.6	97.8	97.7	97.8	97.6
tic-tac-toe	86.7	85.2	87.0	86.2	86.7	86.2	87.0	86.6	84.3	85.8
vehicle	71.9	67.5	70.9	72.0	72.1	68.0	69.7	67.7	70.0	72.7
vote	88.5	88.7	88.5	86.9	88.5	88.7	88.3	85.2	87.8	88.3
vote1	94.5	95.4	95.0	94.7	94.5	94.7	94.7	95.0	95.2	95.2
average	82.9	82.6	83.5	**83.7**	82.5	83.5	82.5	82.3	82.4	**83.7**

looking at an individual domain (such as horse-colic), there is an extremely wide range in the performance of the many attribute selection methods. In such cases XV is able to distinguish between the high and low accuracy methods.

Surprisingly, first place in the average results was a tie between cross validation, which we expected to do well, and entropy, one of the oldest proposed methods of attribute selection. (However, average results should not be over-interpreted and are only included as a rough measure of overall performance. Average results are irrelevant if a particular database under consideration possesses certain characteristics

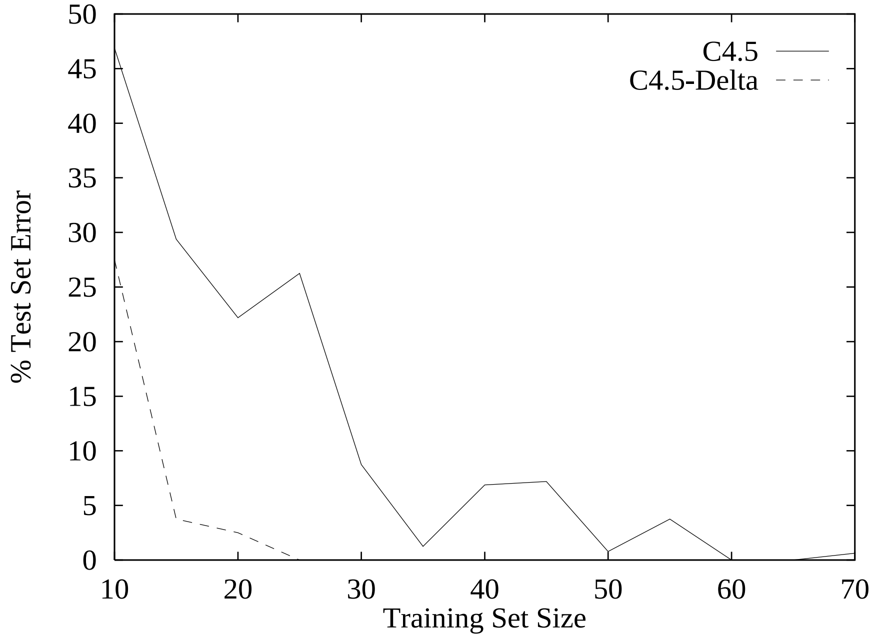

Figure 7.3: Learning curve showing Δ criterion versus gain-ratio, with two relevant attributes, three irrelevant, and 0% Noise. Each plotted point is the average of 20 trials.

that we know to be problematic for some splitting criteria but not others.) Table 7.3 indicates that on average the methods split into two basic categories, those whose average was 83.5-83.7 (LdM distance, entropy, gain ratio, and XV), and those whose average was 82.3-82.6 (Gini, the pessimistic estimate, GORT, and sum minority), with χ^2 being somewhere between the two groups.

We have not yet addressed our third question—whether using additional geometric information is helpful—and this is difficult to answer using Table 7.3. Comparing Δ against gain-ratio (picked because it is the default for C4.5), neither is a clear winner—the experiments are roughly evenly split between a win for one, a win for the other, or a tie.

One of the advantages of using the geometric information in the database should be that bad splits, such as those on irrelevant attributes, could be avoided. We ran experiments on two artificial databases to test this hypothesis. In each case, all attributes were binary, and the target function was the parity of the relevant

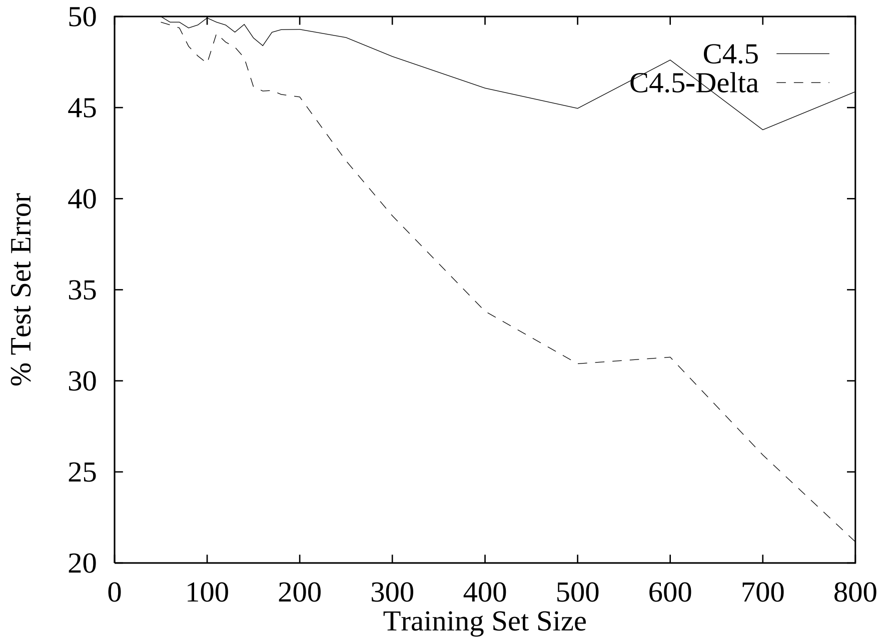

Figure 7.4: Learning curve showing Δ criterion versus gain-ratio, with four relevant attributes, eight irrelevant, and 20% Noise. Each plotted point is the average of 20 trials.

attributes. The first experiment, shown in Figure 7.3, used a database with two relevant attributes, three irrelevant attributes, and no class noise. The graph shows that the Δ criterion attains 0% error on the first database with a training set of 25 records, while gain-ratio is still making some errors with 70 records. In Figure 7.4, we used a more difficult artificial database with four relevant attributes, eight irrelevant, and 20% class noise, so that the best-attainable error rate on the test set is 20%. Gain-ratio shows little sign of progress over the learning curve in Figure 7.4, still having an error around 45% after 700 records, by which point the Δ criterion has nearly achieved an optimal error rate.

7.4 Related Work

Buntine (1992) and Friedman (1977) discuss *lookahead* in classification tree induction, searching down a level in the tree before committing to the choice of a particular attribute at a node. They only do lookahead up to two levels, which would not be enough to solve the problems in Figures 7.3 and 7.4. Taking lookahead to an extreme, Utgoff (1994) discusses a promising approach called *direct attribute selection*, which he uses with his ITI (incremental tree induction) algorithm. In these methods an attribute is evaluated directly (Utgoff views heuristics such as Gini and Entropy as indirect methods) by building out the entire tree using that attribute to split at the current node, and then evaluating the resulting tree. This would ordinarily be computationally prohibitive, but ITI is an incremental algorithm that can support such operations efficiently.

Another avenue of research is to explore the use of more than one splitting criterion during tree construction. For example, obviously a decision node whose children are all leaves should be constructed to maximize accuracy. Brodley (1995) considers this idea and develops an algorithm that switches between the information gain and accuracy (sum-minority) criteria, using information gain high in the tree and accuracy near the leaves. In some cases the new algorithm strongly outperformed either criterion by itself.

7.5 Conclusions

We introduced a new kind of splitting criterion, Δ, which uses geometric information in the training data to attempt to find a split matching our intuitions about the characteristics of a good split. Δ worked extremely well compared to gain-ratio in artificial experiments, but averaged over all domains was nearly indistinguishable from Gini and slightly worse than gain-ratio.

We defined a new splitting criterion, XV, that uses cross-validation to choose from among eight basic splitting criteria and found that, on average, the XV criterion was as good as the best of the basic criteria. On any given database, XV does not often improve upon the performance of the best basic method, but it does provide a margin of safety and was never the worst method.

As observed by others (Breiman et al. 1984), the attribute selection measures are all overall roughly comparable with no clear winner, but there did seem to be a set of winners. The information-theoretic splitting criteria were the best—along with López de Mántaras' distance function and the wrapper method, they achieved average accuracies about 1% higher than the rest of the methods (a relative error rate decrease of about 5%). On most of the databases there was a small significant difference between many of the splitting algorithms, so we cannot take this indifference too far. Even though no single criterion is best on all databases, on any single database the choice of splitting criterion is important.

Chapter 8

Trees: Multivariate Partitioning

In this chapter, we present a new method for the induction of classification trees with multivariate linear discriminants as the partitioning function at each internal node. We define a novel objective function called *soft entropy* for finding linear discriminant splits in classification trees. Some previous research in classification trees with linear splits has resorted to mathematically convenient criteria that were not well-motivated for use as splitting criteria in classification trees. Our *soft entropy* criterion takes the best simple splitting criterion from Chapter 7, entropy, and extends it (*softens* it) in a straightforward way to handle multivariate linear splits. We use soft entropy in our Slic (Soft LInear Classification) algorithm, and compare its unpruned trees with unpruned versions of a similar linear discriminant method, OC1 (Murthy, Kasif & Salzberg 1994), showing that Slic's trees are somewhat smaller. We then describe the Slice (Slic-Esteem) algorithm, which uses the Esteem method from Chapter 3 during pruning, and we compare the results with the pruned versions of OC1 and CART (Breiman et al. 1984).

8.1 Introduction

As we have previously discussed, classification trees are an important type of pattern in data mining. Their wide and successful use in fielded applications and their simple intuitive appeal make data mining algorithms for classification trees an important area

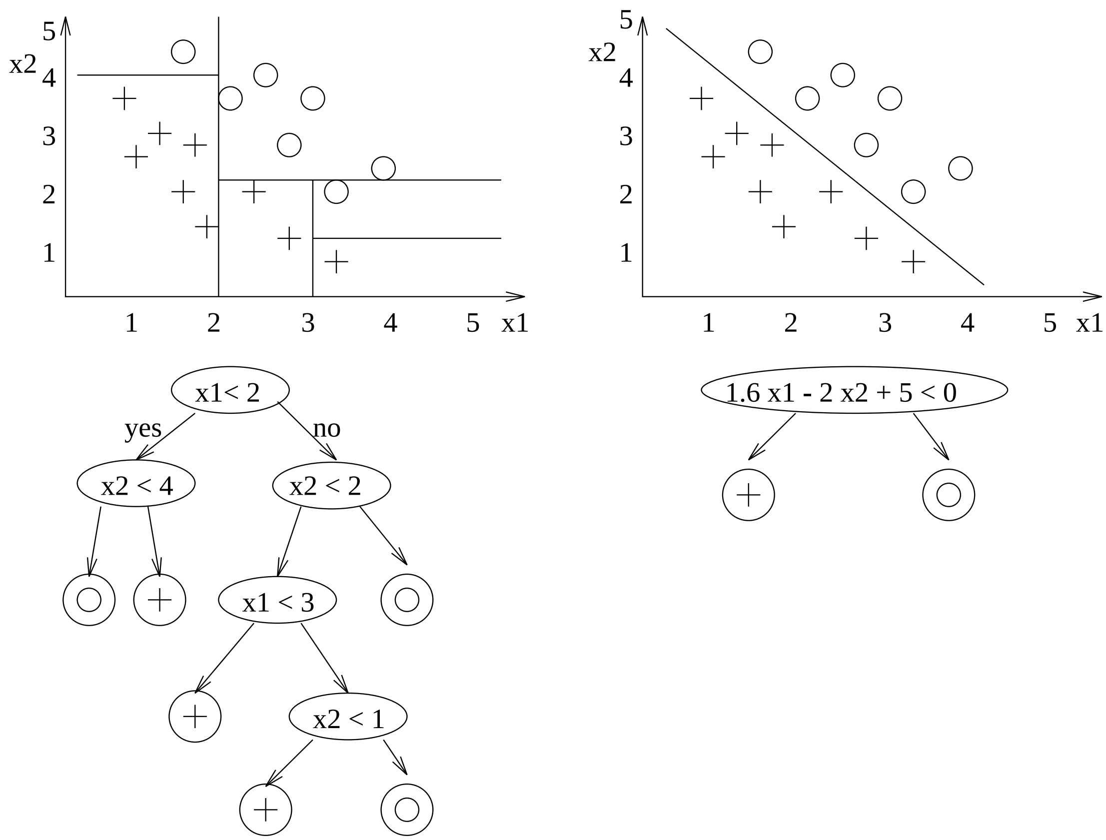

Figure 8.1: The limited representational capacity of univariate splits.

of study. However, because traditional classification trees make *univariate* partitions of the input space, they may perform poorly on even simple target functions if the divisions between classes in the input space are not axis-parallel. Figure 8.1 shows an abstract example database where the records are only discriminated with some difficulty when using univariate partitions, but the figure also shows they can be easily discriminated by a single linear multivariate split.

In the algorithms described in this chapter, we follow the traditional top-down, recursive partitioning approach to tree building, but rather than partitioning the data on a single axis we instead employ a linear discriminant at each node to recursively split the data (Henrichon, Jr. & Fu 1969, Friedman 1977, Breiman et al. 1984). The issue of how to find a good discriminant naturally arises. We discuss problems with previous approaches and propose a new splitting criterion, *soft entropy*. The SLIC algorithm uses recursive partitioning on this criterion to build a pure tree, one that correctly classifies each record in the training set.

A problem with studying classification tree splitting criteria is that one seldom uses a classification tree without first pruning it in an attempt to avoid overfitting. Thus, results on the superiority of one splitting criterion over another should be in the context of the use of some regularization algorithm in order to mirror the results we expect to observe in practice. To address the overfitting problem, we developed SLICE, an algorithm that prunes the pure tree using a method similar to a stopping rule, then uses the ESTEEM method from Chapter 3 to remove incorrectly-classified records and retrain.

In Section 8.2 we formalize the problem of finding a splitting function for an arbitrary splitting criterion as a function optimization problem. Section 8.3 discusses problems with splitting criteria used in previous work in classification trees, and presents the soft entropy criterion. Section 8.4 presents the pruning algorithm, and experimental results comparing SLIC and SLICE with related algorithms are discussed in Section 8.5. Section 8.6 discusses related work, and Section 8.7 gives directions for future work. Finally, Section 8.8 presents our conclusions about multivariate splits.

8.2 Finding Splitting Functions

In this section we frame the problem of finding the best linear split at a node in the tree as function optimization. Just as univariate decision trees search for the best split at each node in the tree, so does SLIC. The essential difference is that there are relatively few possible splits to choose from in a standard tree, whereas with linear splits the space of candidate splits is continuous and multidimensional, so we cannot search the entire space and instead use the heuristic function optimization method of gradient descent, which is just hill climbing in a continuous space.

Another difference in the SLIC approach is that, because the space of candidate splits is so large, standard splitting criteria assign the same value to many different splits, so we generalized standard splitting functions that are based on cardinalities of sets to *soft* splitting functions that are based on cardinalities of weighted sets. A weighted set of records S has a membership weight function w_S, such that for any record $\mathbf{x}$, if $\mathbf{x} \in S$, $w_S(\mathbf{x}) \in (0, 1]$, and $w_S(\mathbf{x}) = 0$ for all $\mathbf{x} \notin S$. Unweighted sets are a special case where $w_S(\mathbf{x}) \in \{0, 1\}$.

Let θ be the vector of parameters for a splitting function. (In the case of linear splitting functions, θ is the vector of coefficients.) Let g_θ be a *soft* binary splitting function, mapping records into $[0, 1]$. (*Hard* splitting functions have the range $\{0, 1\}$.) Given a weighted set S, define S/g_θ to be the tuple of weighted sets (S_0, S_1) produced by applying g_θ to each record $z \in S$ such that $w_{S_0}(z) = g_\theta(z) w_S(z)$ and $w_{S_1}(z) = w_S(z) - w_{S_0}(z)$. In other words, g_θ partitions the records in S into two sets S_0 and S_1, such that the weight of any record $\mathbf{x}$ in S_0 and S_1 sums to its weight in S, and g_θ determines the portion of the weight assigned to S_0.

We can now define the problem of choosing a splitting function at a node of a classification tree as a function optimization problem. Given a weighted set S of training records we desire to split and some fixed parametric form of the splitting function g_θ, our goal is to find the optimal vector of parameters θ^* that results in an optimal splitting function g_θ, according to some objective function. We use I to denote the objective function or splitting criterion, since impurity measures are a common family of splitting criteria. I maps two weighted sets to the real numbers $\mathbb{R}$, and the optimal split has the parameter vector

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmin}} \; I(S/g_\theta) \ . \tag{8.1}$$

We could as well search over several different forms of g, so that the minimization is over both g and θ. We have investigated this approach, using simple three-layer neural networks as splitting functions, but we will not pursue it further in this chapter. Briefly, the results were that the nonlinear splitting functions sometimes worked well, but the parameter vector for the splitting function would very often get stuck in a bad local minimum, causing a bad split that resulted in an overly large tree with low accuracy.

In defining a problem as function optimization there are three important decisions: the domain in which the optimization should occur, the objective function, and the function optimization method. The domain, Θ, is defined by whatever functional form we set for g—all parameters remaining to be set constitute Θ. The optimization algorithm could be any method for unconstrained optimization, including methods using first derivative information if $\partial I/\partial\theta$ can be calculated.

Fortunately, by the chain rule $\partial I/\partial\theta$ can be separated into the product of two terms: $\frac{\partial I}{\partial g_\theta}\frac{\partial g_\theta}{\partial\theta}$, so that we need only calculate the derivatives for the splitting criterion with respect to the outputs of the splitting function (whatever it may be) and the derivative of the output of the splitting function with respect to its parameters θ (regardless of the splitting criterion being used). This extremely useful property will later allow us to provide a general framework for the softening of any criterion based on simple statistics, such as as the entropy, Gini, and Chi-squared criteria described in Chapter7.

The essential ingredient of the optimization process is the objective function: the splitting criterion I which we wish to minimize. All of the common splitting criteria discussed in Chapter 7 are defined on sets but have straightforward extensions to weighted sets. The criteria are all just functions of the following counts: the total number of records in S, the total number in S_0 and S_1, and the number of records of each class in S_0 and S_1. For weighted sets, we define cardinality as $|S| = \sum_{z\in S} w_S(z)$. Simply by replacing sets with weighted sets and the standard cardinality measure by the new cardinality measure, we can define *soft* versions of most of our the common splitting criteria.

As an example, consider the softened linear discriminant splitting function

$$g_\theta(\mathbf{x}) = 1/(1+\exp(-\theta^T\mathbf{x})) \ , \tag{8.2}$$

where θ^T is the (row-vector) transpose of the column vector θ, and $\mathbf{x}$ is the column vector of input attribute values. Note that the range of g is $(0,1)$, thus when using g_θ to partition a set S, all $z\in S$ are partially assigned to both S_0 and S_1. Geometrically, such a soft split is like a slice through the input space, such that training points far from the slice on one side are assigned values close to 0, points far away on the other side get values close to 1, and the values get closer to .5 as the points get closer to

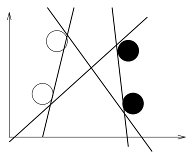
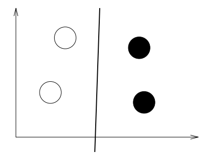

Figure 8.2: Left: all four splits have equivalent entropy (and Gini, Δ, *etc.*). Right: the split found by minimizing soft entropy.

the slice itself.

Our function optimization algorithm repeatedly evaluates $I(S/g_\theta)$ to get the value of the current θ, makes adjustments to θ (perhaps based on $\partial I/\partial\theta$) and evaluates the new θ. Eventually the optimization algorithm stops, outputting $\widehat{\theta}$, its estimate of θ^* from Equation 8.1.

8.3 Building Trees with Soft Entropy

Although several approaches to building linear discriminant trees have used a soft splitting criterion (often mean-squared error in logistic regression), and many other approaches have used a hard criterion based on entropy or Gini, we are aware of no approach combining the desirable properties of both. Below we discuss why we prefer a criterion based on entropy and why we prefer a soft criterion. We then give the details of the tree construction algorithm Slic.

8.3.1 Why *Soft* Entropy?

The left graph in Figure 8.2 shows several candidate hard splits of a small database. All of the splits shown have equivalent counts (number of records of each class in each subset) and hence they are evaluated equally by a hard splitting criterion. Because they use sets instead of weighted sets, hard criteria possess limited granularity and are unable to distinguish between splits that are quite different. The split they select as the best will thus depend on random factors such as the initial splitting hyperplane weights, or the order of the data records, and this kind of instability is generally undesirable for data mining algorithms. Domain experts tend to lose faith in data

Figure 8.3: Left: The soft linear discriminant minimizing mean squared error fails to partition the data. Right: The soft linear discriminant minimizing soft entropy.

mining algorithms if they are not demonstrably consistent.

Since the soft split assigns records partially to both the left and right subset (in the case of Equation 8.2, based on the distance between the record and the splitting hyperplane), the hyperplanes in Figure 8.2 on the left are assigned different values by the splitting criterion. The unique best soft split is shown in Figure 8.2 on the right. Additionally, since soft criteria are differentiable they are amenable to the use of a wider variety of function optimizers.

8.3.2 Why Soft *Entropy*?

Given that softness is a desirable property, much work has used the soft linear discriminant of Equation 8.2 combined with an error criterion such as sum-squared error, common in logistic regression and neural networks. However, Breiman et al. (1984) argue against error as a splitting criterion, and our example in Figure 8.3 demonstrates a shortcoming of error. Figure 8.3 shows two linear discriminant functions that each optimize some splitting criterion. (In this case they happened to be soft linear discriminants, but the same phenomenon occurs with hard splits.) The entropy-minimizing discriminant does an excellent job of partitioning the data, while the error-minimizing discriminant fails to partition the data at all. The discriminant with lowest error classifies all points as the majority class, making only three errors.

8.3.3 Finding the Best Discriminant

To build trees using the soft entropy criterion, we must combine several ingredients. Perhaps most important is the agenda for constructing the tree itself. We will use the standard top-down, recursive partitioning approach, at each node finding the splitting function that optimizes some splitting criterion, splitting the data with this splitting function, and then recursively building trees from the resulting subsets.

The splitting criterion we wish to minimize is still $I(S/g_\theta)$, where I is now specified to be the entropy measure. Equation 8.2 defines the splitting function g. In this case, calculating $I(S/g_\theta)$ and $\partial I/\partial\theta$ is straightforward, so we used the simplest function optimization method using first derivative information: steepest descent.

The function optimization technique used in Slic is a modified version of steepest descent. These optimizers begin with some θ_0, then repeatedly move some distance in the direction of the gradient of the objective function at θ_0. Typically, either the user sets the distance as a parameter, or it is found automatically by a computationally expensive line minimization (Press et al. 1992). In Slic we have employed a few tricks to get reasonable speed while retaining the basic steepest-descent algorithm. (We do not claim to be at the cutting edge of function optimization here, but we include the algorithm for replicability.) The update equation for going from θ_i, the parameters currently under consideration, to θ_{i+1}, the next parameters to be considered, is:

$$\theta_{i+1} = \theta_i - 0.1\frac{\frac{\partial I}{\partial\theta}|_{\theta_i}}{||\frac{\partial I}{\partial\theta}|_{\theta_i}||^{0.8}} + 0.8(\theta_i - \theta_{i-1}) \ . \tag{8.3}$$

The algorithm stops when the current $I(S/g_\theta)$ drops below 0.00002, when $i > 1000$, or when 20 steps fail to produce a change in I of at least 0.00005. The whole process repeats once, starting from a different θ_0, in an attempt to avoid the perils of getting trapped in a local minimum.

It is important to note that, once the best θ is found, we use a hard split to partition the data into a left and right subset, which are themselves recursively split. The goal of the work on soft splitting functions was just to discover a good hard split. In the case of the soft linear discriminant, it is replaced by its hard cousin:

$$g_\theta(\mathbf{x}) = 1 \text{ if } \theta^T\mathbf{x} \geq 0, 0 \text{ otherwise} \ . \tag{8.4}$$

This concludes the Slic algorithm for building a tree.

8.4 Pruning a Tree

While studying the trees induced by the algorithm, we found that most trees contained several leaves matching on the order of 50 to 100 records from the training set, and a few leaves matching only a few records. We conjectured that these isolated records constituted noise in the training data, and that we might improve performance by pruning such records.

Our regularization algorithm contains a simple pruning step and a retraining step. The pruning step replaces a decision node with a leaf whenever the number of positive or negative records matching that node drops below k. (In our experiments, k was arbitrarily set to 5.) This is applied to all nodes in the tree in a top down fashion. We then use the ESTEEM method described in Chapter 3 to discard the incorrectly-classified records and retrain, repeating this process until all records in the reduced database are classified correctly.

8.5 Experiments

We ran cross-validation experiments comparing SLIC and SLICE with CART and a CART-like algorithm, OC1. First we discuss experimental methodology, then present and analyze the results.

8.5.1 Methodology

We used five databases from the UCI (Murphy & Aha 1994) and Statlog (Michie et al. 1994) collections. All databases were processed (either by the author, or in the case of the Statlog databases, by others) so that all unordered categorical attributes were encoded using 0-1 indicator variables and all ordered categorical attributes were encoded as integers. All missing values were either removed or replaced with their mean or mode.

We used three different algorithms: the soft entropy algorithms discussed in this chapter, CART (Breiman et al. 1984), and OC1 (Murthy et al. 1994, Murthy, Salzberg & Kasif 1993). We set CART's parameters so that it would use linear splits whenever

Table 8.1: Test set accuracy and tree size results from 10-fold cross-validation.

		Vote1	Australian	Breast-Wisc	Diabetes	Heart
SLICE	Acc	89.0±4.0	85.0±4.6	95.1±2.2	73.7±5.8	77.8±10.5
	Size	8.8±3.7	8.0±3.0	6.2±2.3	16.6±5.1	7.4±2.3
SLIC	Acc	85.5±4.8	80.2±4.3	94.3±1.9	68.9±6.7	74.5±11.0
	Size	20.4±3.3	85.8±11.0	19.8±4.3	127.6±10.1	25.0±2.7
OC1-pruned	Acc	89.9±4.2	85.1±2.9	95.6±2.7	74.3±3.5	75.5±6.8
	Size	5.8±6.0	10.2±9.6	3.6±1.0	13.6±12.1	9.6±10.2
OC1	Acc	87.1±3.5	81.2±4.6	94.9±1.6	68.4±4.3	72.6±7.5
	Size	36.2±5.3	83.6±11.0	35.2±5.4	149.8±7.7	42.2±6.3
CART	Acc		82.0±2.9	96.2±3.1	74.6±4.5	81.5±8.9
	Size		3.0±0.0	3.8±1.8	3.6±9.7	3.4±1.3

the number of records at the node was greater than ten times the number of attributes. (When using linear splits, the version of CART we were using seemed to be quite numerically fragile. This high setting was required to get it to run on most of the databases.) CART used ten-fold cross-validation and the 1-SE rule to select the correct pruned tree. We used default values for all other CART parameters. OC1 is similar to CART in that it also learns trees with linear splits, but it uses a modified version of CART's optimization algorithm that the authors claim is better at avoiding local minima. It does not offer the option of using cross-validation to select a pruning parameter; rather, it randomly picks 10% of the training set to use as a pruning set. Default values were used for all OC1 parameters.

We used ten-fold cross-validation to estimate the accuracy of each algorithm on each database. (This is a separate step from CART's own internal cross-validation.) For each of the ten runs, we report the average and standard deviation of both the accuracy of the trees and their size.

8.5.2 Results and Discussion

Table 8.1 gives 10-fold cross-validation results for the soft entropy algorithms, the OC1 algorithm, and CART. Each box contains two measurements for the ten runs: the top is the accuracy, and the bottom is the tree size.

No results are reported for CART on the Vote1 domain because CART failed with a numerical error on each of the ten folds. On the diabetes and heart domains, all ten runs completed successfully; on the rest, only a few runs completed without errors.

We applied the paired t test to every comparable set of results. Comparing the tree sizes of OC1-unpruned and Slic, we found that Slic produces smaller trees on all domains except Australian with over 99.5% confidence. Within the soft-entropy algorithms, Slice is superior to Slic with 95% confidence on vote1 and Australian. Slice is superior to Slic with 90% confidence on breast and diabetes. Comparing CART, OC1, and Slice we found that none of the differences in accuracy between these algorithms is significant at the 90% level, except that CART performed better than OC1 on the Heart data at the 95% confidence level. Comparing the accuracy of OC1-unpruned and Slic, OC1-unpruned is better than Slic at the 90% confidence level on vote1.

Although the accuracy results are inconclusive, the sizes of the unpruned trees support the idea that soft entropy is a good splitting criterion, at least in terms of producing smaller trees. Regarding the Esteem method, it seems promising that Slice results were competitive with OC1-pruned and CART, both of which employ more sophisticated methods for pruning. When combined with these better pruning methods, Esteem might yield an even better boost in performance.

8.6 Related Work

John (1994*b*) presents preliminary experiments with Slice, comparing against other neural network and classification tree algorithms. Friedman (1977) gives an excellent discussion of practically every issue related to the induction of classification trees from data, including linear discriminant splits using Fisher's linear discriminant (Duda & Hart 1973). The earliest attempt at building linear discriminant trees seems to be Henrichon, Jr. & Fu (1969), who use the maximum eigenvector of the covariance matrix to determine the split. Lin & Fu (1983) use k-means clustering to find the splitting function, whereas Bennett & Mangasarian (1992) use a parallel pair of hyperplanes as a splitting function and use linear programming to minimize their splitting criterion, which was the distance between the pair. Sankar & Mammone (1991) present "neural tree networks," which seem to be simply soft linear discriminant trees using mean squared error as the criterion. Loh & Vanichsetakul (1988) discuss linear discriminant trees where the discriminant hyperplane may be solved for exactly by making normality and homoscedasticity assumptions about the data (*i.e.*, assuming the distribution of the data from each class is normal and has the same covariance matrix). Qing-Yun & Fu (1983) present a sophisticated method for not only finding linear splits at each node (based on regression) but also selecting the best subset of variables to use in the split. Brodley & Utgoff (1995) present a similar algorithm, also based on linear regression with subset selection.

Although the work in this chapter was developed independently, Brent (1991) discusses essentially the same technique for softening a related splitting criterion. However, Brent's focus was not on finding a better splitting criterion for multivariate classification trees; rather, he used his measure to train neural networks one node at a time. Sethi (1990) gives the original description of the mapping between neural nets and classification trees, and proposes converting the tree (built with univariate splits) to neural net form for further refinement. Along the same lines, Jordan & Jacobs (1993) present a method that allows simultaneous optimization of all coefficients in the tree by defining the model likelihood and using the EM algorithm (Dempster et al. 1977) for optimization. They use soft splits not only for tree construction but also during classification. However, their method requires the user to specify the tree structure a priori. Sahami (1993) investigates various optimization algorithms for finding an error-minimizing discriminant at each node in a classification tree.

Bichsel & Seitz (1989) gives an algorithm for training a tree-structured net very similar to the approach by Heath et al. (1993), using simulated annealing to search the space of hyperplanes at each node. The procedure for adding nodes is somewhat similar in spirit to that used in top-down induction methods, but their actual construction algorithm assumes the tree will be very small. Koutsougeras & Papachristou (1988) discuss a related approach, learning entropy-minimizing hyperplanes in a tree-structured neural net, but in the actual algorithm they instead implement Fisher's linear discriminant function, which minimizes error under very restrictive assumptions rather than minimizing entropy.

8.7 Future Work

A known problem with classification trees is that, as one goes deeper in the tree, less data is available to each node, due to the recursive hard partitioning. Since SLIC already deals with partial assignments of records to the left and right subsets, it would be a simple matter to use the soft partitions in the recursive step of the tree-building algorithm and during classification as in Jordan & Jacobs (1993).

As mentioned in the introduction, the procedure for finding a good splitting function is just one part of a whole classification tree learning algorithm. The most important issue with respect to generalization ability is regularization (pruning). Future work should extend the simple pruning strategy to use cross-validation to pick the threshold, rather than arbitrarily setting it at five. As Brodley & Utgoff (1995) point out, pruning in multivariate trees can be carried out at a finer granularity than removing entire nodes: subset selection should be investigated as an alternative means of regularization.

Finally, another benefit of this approach is that one can easily generalize it to learn splitting functions other than linear splits. Neither the function optimizer nor the objective function depend on the actual splitting function used, so this is a completely independent module. We have investigated the use of more complex splitting functions (neural networks with few hidden units) with mixed results but plan to investigate this further.

8.8 Conclusion

We have provided a general framework to address the problem of finding a splitting function in classification tree induction. Within this framework we have identified a splitting criterion, soft entropy, that allows the optimization of any splitting function that is differentiable. Soft entropy appears to combine the good properties of its hard counterpart, entropy, as well as soft criteria such as mean squared error in the logistic regression setting. Experimental results with linear splits indicate that soft entropy is a better objective function than its hard counterpart. We employed a rudimentary pruning algorithm to address the overfitting problem, and when combined with the Esteem method, this led to results competitive with OC1 and CART. Accuracy results for the Slice algorithm as compared with CART and OC1 are inconclusive. More experiments are needed to isolate the effects of the splitting criterion and pruning methods.

The Slic and Slice algorithms we have described are a useful part of a data miner's toolbox, since the models they produce have a significantly different bias from the other common mining tools. The soft entropy criterion has desirable properties from a data mining perspective, namely more stability resulting from its higher granularity than hard splits in other work and the use of entropy rather than criteria motivated more by mathematical convenience than appropriateness for use as splitting criteria in building classification trees.

Chapter 9

Analyzing Results: Stock Selection

The preceding chapters have addressed many of the steps in the data mining process, up through applying the data mining algorithm. Any lengthy treatment of data mining would be incomplete without some description of the results analysis step. This chapter explores the analysis of results during a data mining project in stock selection. In contrast to the usual stories about data mining projects, in this project we spent the majority of our time in the results analysis step rather than the data extraction, cleaning, and engineering steps. Almost a year elapsed between the time the data mining runs had finished and the time we had prepared the final version of the second conference paper describing the results. In the interim, a significant amount of time was spent taking courses in portfolio management and derivative securities (both of which were necessary to understand how to correctly simulate the trading strategy used), meeting with experts to solicit feedback, and reading papers in financial journals describing trading strategies and simulation methodology. In this chapter we discuss the process of analyzing the results of the data mining runs, not giving all of the analyses but rather a representative sample chosen to illustrate the evolution of the analysis as it matured and became more valuable to an expert in the finance industry.

9.1 Introduction

Quantitative investors, or "quants" as they are known on Wall Street, attempt to make investment decisions using mathematical models that describe the behavior of a stock's future price in terms of variables that are presently observable. This contrasts with the traditional style of fundamental analysis, which involves personal scrutiny of available data on a company, including subjective assessments of the company's management and competitive situation. Ultimately, the goal of both types of investors is to make money, and both broadly agree on how to go about doing it—the key difference is that quantitative analysts focus their efforts on defining a precise method by which objective information on a company should be processed in order to predict future performance, while traditional analysts repeatedly process this information themselves for each stock they track.

High-quality financial databases have existed for many years, but human analysts can only scratch the surface of the wealth of knowledge buried in this data. In this chapter we approach quantitative stock selection as a data mining problem. Primarily, this chapter chronicles our attempts to properly analyze the stock selection models that we discovered. During the analysis we repeatedly sought feedback from experts, who often surprised us by asking questions we could not answer (and which led to further analysis) or by pointing out incorrect assumptions or methods.

In the specific stock selection problem we discuss in this chapter, the database contained quarterly information on 1,160 to 1,480 companies during the period October 1, 1987 through June 30, 1993. For each quarter, we were given a database table in plain ASCII file format, with one record per stock and nearly one hundred different attributes per stock. All of the attributes were publicly available at the beginning of the quarter, except for the return of the stock over the quarter, which of course was only known at the end of the quarter. Table 9.1 shows a hypothetical fragment of the database for the first quarter of 1993, with two example stocks and attributes such as price to earnings ratio, price to book ratio,[1] the number of analysts tracking the

[1] The definitions of these and some other financial terms are tangential to the point of the chapter, and thus are not given. See Sharpe, Alexander & Bailey (1994) for the definitions.

Table 9.1: Hypothetical fragment of one table in a historical database.

	Known As of January 1, 1993					Future
Stock	P/E	P/B	#Analysts	⋯	Trend	3-mo Return
IBM	13	2.2	14		12.1%	14.2%
Apple	19	3.2	8		-3.7%	-6.1%
⋮						

company, and measures of the stock's price trend. The table also includes the return an investor would have received on an investment in the stock, held over the entire quarter.

9.2 Applying the Data Mining Process

The process we followed in mining this database parallels the description in Chapter 1. During the first pass through the data mining process, the steps taken were:

- **Define the Problem**. This was a pilot project. A mutual fund company wanted to see how well a commercial data mining tool would work on stock data. The company's ultimate goal was to use data mining to build models that would assist the fund managers in their stock selection. For the pilot project, since we had no particular expertise in portfolio management, we chose to let the models pick stocks entirely autonomously. For the first pass through the process, the specific problem was to build a model for each quarter that would rank stocks by their predicted return, using the previous four quarters as training data.

- **Extract Data**. The database provided by the mutual fund company was already in the proper format (delimited ASCII files in this case), so no extraction was needed.

- **Clean the Data**. As mentioned in Chapter 1, we spent some time exploring the database and found that the "Return" column had been mislabeled. After fixing the column names, we randomly compared several values of the new Return field with an independent database to ensure the values were correct.

- **Data Engineering**. The database already included many attributes containing information on trends of the basic attributes (*e.g.*, the change in a ratio between the current and preceding quarter, or the change in price over the same period), and it already included most of the common financial ratios and other derived attributes, so we created no new attributes. The data mining algorithm we used was invariant to any rank-preserving transformations, so some derived attributes in the database became redundant and were removed. Stock selection is a somewhat unique domain for data mining, since everyone in the world who is working on the problem basically has access to the same data, so one can leverage the collective experience of the community in this step of the process. Many academic papers in finance discuss ratios of basic attributes that are thought to be informative, such as the price-earnings and price-book ratios mentioned earlier.

- **Algorithm Engineering**. The algorithm we used could only do binary classification, whereas we really wanted a ranking of the stocks by estimated future return. We chose to rank stocks by return and identify the top twenty percent in each quarter as the "exceptional return" stocks, and we identified the rest as "unexceptional." We added this new binary-valued attribute to each of the database tables as the class to be predicted. Next, we decided to use the previous four quarters of data to predict the class in a given quarter. We set the algorithm's main parameter to what is essentially the "slower but probably more accurate" mode, and left other parameters at their default settings.

- **Run the Data Mining Algorithm**. The algorithm we used is a descendant of the algorithms described by Kerber (1991, 1992). It discretizes all numeric attributes and then learns a set of rules whose antecedents are value assignments to some (usually small) number of input attributes, and whose consequents are a value assignment to the class. When making a prediction, several rules may all match a given input record and their predictions are combined into a single

> score. For each quarter in the database, we concatenated the preceding four quarters of data, used them as a training set, built a model, and applied that model to the target quarter. The result of this application step was a score for each record. Scores near 1 indicated that the stock was estimated to have a high probability of exhibiting exceptional return during the current quarter, and scores near 0 indicated a low probability of exceptional return. We held out the last four quarters to use as a final independent test later.

We iterated through this process a number of times, experimenting with different values for the mining algorithm's fast/slow parameter, the definition of *exceptional*, and the number of quarters of historical data to use in building each model, yielding twelve total runs through the process. This is somewhat similar to the parameter wrapper discussed in Chapter 5, except that in the iterations through the stock selection process we also varied the problem formulation. We manually tuned not only the parameters of the data mining algorithm but also the parameters of the precise problem definition and the algorithm engineering step.

Recall from Chapter 1 that, when performing many iterations through the data mining process, one needs a *holdout set* on which to compare the various models produced by the different parameter settings. The stock selection problem presents a new twist: rather than having a single database and building a single model, we had many database tables arranged chronologically and we built many models. We built each model with the goal of predicting the *exceptional* attribute in a given quarter, but the model was only trained using data from previous quarters. Thus each model's target quarter constituted a holdout set for that model. We will not describe the details of the process by which we selected the final parameters, but the key was that the objective function we used to choose the parameters was the same objective that was truly important in the data mining problem. Chapter 5 discussed using cross-validation to measure accuracy when tuning parameters, but in the stock selection problem, accuracy may be only loosely correlated with what we really care about—the return of a portfolio constructed using the model's predictions of future return. We turned the predicted scores into portfolios by selecting the fifty highest-scoring stocks and giving each stock a portfolio weight proportional to its market value.[2] Using the cumulative portfolio return as an objective function, we picked the best parameter setting for the data mining process.

Recall that, when comparing several approaches to a data mining problem, a holdout set provides a useful benchmarking tool. However, if the holdout data are used to select the best approach, the estimated performance of the winning approach on the holdout data may be biased; after all, the winning approach was *chosen* based on its ability to do well on the holdout data. Independent *test* data are required to give an unbiased estimate of the performance of the winning approach. In our experiments, we set aside the last year of data to look at only *once*, after we had picked one of the twelve methods.

9.3 Analyzing Results

Now we can consider the results analysis step of the process. For every quarter (including the last four, now that we have decided on the parameters of our solution) we have a corresponding model that we trained on the preceding four quarters. We apply each model to its target quarter of data, generating a prediction of the likelihood that each stock will exhibit exceptional performance during the quarter. There are a variety of ways to analyze the results at this point—many papers have been written about the application of data mining to stock selection and other financial problems in which authors simply present the accuracy or mean squared error of their models.[3] Members of the investment community could not seriously evaluate such results because the authors did not take the time to perform analyses that an investor would care about. A variety of questions could still be asked, such as "So what if your model is right 55% of the time—how much money could I really make if I used your model to trade, net of trading costs?" Since the stated goal of our stock selection study was to identify stocks with high return, we used our model to rank stocks by predicted

[2] The *market value* or *market capitalization* of a stock is the number of outstanding shares times the share price.

[3] Peter Lynch (1993) wrote that his doorman, who greeted him every day by saying "Market's going up today!", was one of the most accurate market forecasters he knew.

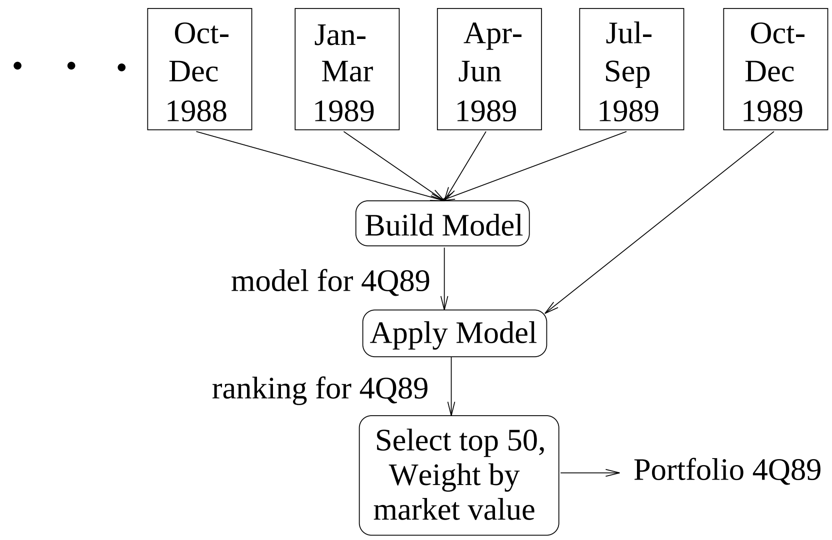

Figure 9.1: The process of building a portfolio from current and historical data, using the portfolio held in simulation from October 1, 1988, through December 31, 1988, as an example.

return and we simulated trading a portfolio of the top 50 stocks. We pretended that we began on October 1, 1988, with ten million dollars. We took the model that was built on the preceding four quarters and applied it to the data available on October 1, 1988. We picked the fifty highest-scoring stocks and formed a portfolio, investing our $10M fully in the stocks, weighted by their market value.

The portfolio construction process is depicted graphically in Figure 9.1. Nothing happens in the simulation until the end of the quarter, at which time the portfolio value grows based on the actual return of the constituent stocks during the quarter. Then, at the beginning of the next quarter on January 1, 1989, we took the new desired portfolio (which we obtained through the same process as before) and sold the stocks we no longer wished to hold (receiving cash) and bought the new stocks we wanted to hold, so that at the end our portfolio was exactly the desired portfolio for the new quarter, with no money in cash. We simulated 1% *round trip* transaction costs by subtracting 1% from the amount of cash we received from the sale of stock shares. In Figure 9.1 we are reporting results on holdout data and test data. The last four quarters of data were only used *once* after we had fully specified the data

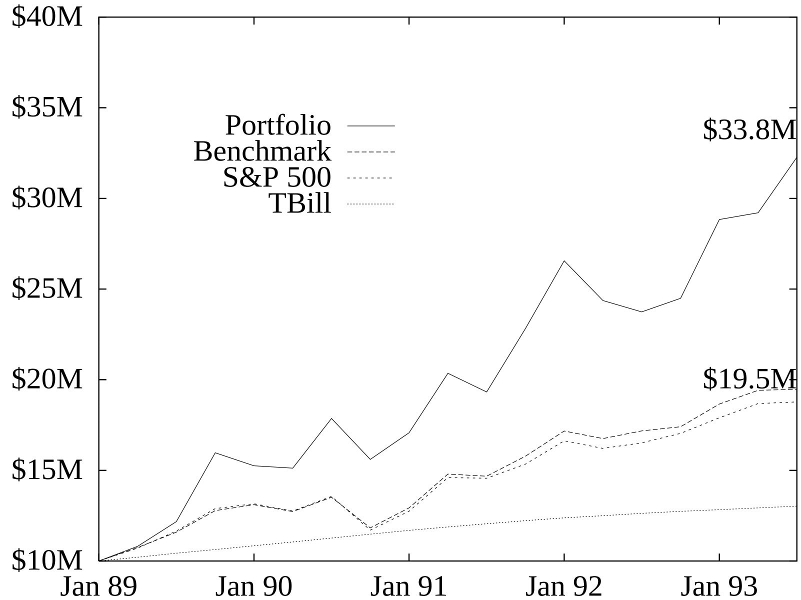

Figure 9.2: The cumulative returns to our portfolio and benchmark portfolios.

mining process. The first fourteen quarters are the holdout period, which was also used to set the parameters for the study. Our portfolio outperformed its benchmark even more soundly during the test period than during the holdout period, so we felt that in this case the holdout performance was not overly optimistically biased. Also, it was important to display results for a more extended period of time than the one year of test data and, lacking more data, the best we could do was to show results from the holdout data.

The work described up to this point culminated in the graph of Figure 9.2. The graph shows the total return of four hypothetical investments over the period. The first is our simulated portfolio and the second is a custom benchmark designed for our portfolio. Any investment strategy must be evaluated with respect to an appropriate benchmark, which should be the strategy that would have been used if the primary strategy had not been followed. The mutual fund management company did not give a description of the screening process used in creating the selection universe, the set of stocks from which we built our portfolios. Thus, our best alternative strategy was to buy a market-cap-weighted portfolio of all the stocks in the universe from which our data mining models were selecting. This is the "Benchmark" portfolio in Figure 9.2.

Table 9.2: Some common financial measures of our portfolio and the benchmarks.

Portfolio	Total Return	Quarterly Avg Return	Quarterly Std. Dev.	Sharpe Ratio	Beta
Our Portfolio	238%	7.41%	11.8%	.58	1.56
Benchmark	95	3.96	6.24	.39	1.00
S&P500	92	3.84	6.29	.38	1.01
Tbills	32	1.5	.5	0.00	0.00

We assessed no trading costs to the benchmark for portfolio re-weighting. For comparison, we also show the total return to the Standard and Poor's 500 portfolio, along with the cumulative return of Treasury bills during the period.

We thought at this point that we were finished. Actually using the models to build a portfolio and comparing the portfolio to a proper benchmark were important steps in the analysis process that we had not seen in previous papers on data mining for stock selection, and thus we were confident that we had done an exceptionally thorough evaluation. Then we showed the results to an institutional equity manager, who seemed unduly unimpressed.

The main theme of the manager's feedback was that we had not adequately characterized the risk of our portfolio, and we had not done any performance attribution—attempting to determine why the portfolio performed so well. Table 9.2 shows part of the requested analysis, again considering our portfolio, the benchmark, the S&P 500 and Treasury bills. The total return column in Table 9.2 is just the percent change in our simulated $10M holdings from the beginning to the end of the period. The quarterly return figures are simply the average return for each portfolio over the eighteen quarters in the simulation. Our portfolio's return is substantially higher than the benchmark, but our financial expert was worried that our portfolio might involve much more risk. Standard deviation of returns is a common way to measure risk, the idea being that a portfolio with constant return of some fixed percent every period is preferable to another portfolio whose returns have high variance, even if the average return is the same for both portfolios. Our portfolio's returns did have much higher standard deviation than the benchmark. Thus, our return was higher but so was our risk, so where did this leave us?

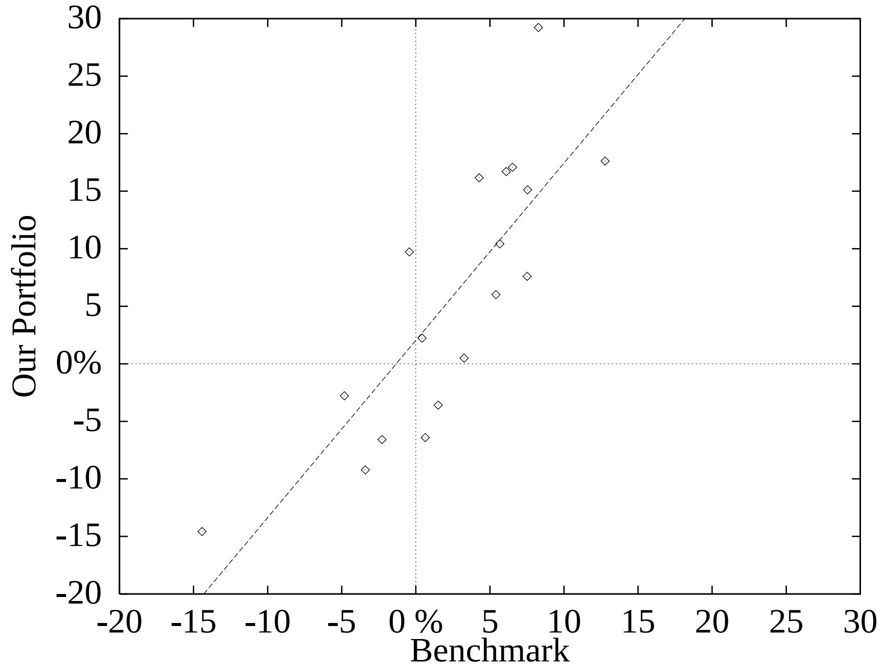

Figure 9.3: A linear regression of our portfolio's quarterly excess return versus the Benchmark excess return.

The Sharpe ratio (Sharpe 1994) is the amount of excess return a portfolio adds per unit of risk, and is a common way to measure the risk/return tradeoff. The Sharpe ratio for our portfolio was much higher than the benchmark, which means that our portfolio provides an investor with a better means to obtain returns than the benchmark portfolio. One implication is that if an investor wishes to be conservative and have an objective of, say, 4% return per quarter, it would be better (less risky) to invest in a combination of our portfolio and Treasury bills than to invest entirely in the benchmark portfolio.

Finally, Beta is a measure of how a portfolio's returns vary with respect to a benchmark portfolio's returns. Figure 9.3 shows the scatter plot of our portfolio's excess returns versus the benchmark's excess returns, where Beta is the slope of the fitted line.[4] Beta is a useful and common metric that measures the relative risk or

[4]The excess return to a portfolio is defined as its return minus the risk-free rate (the return one could have received by investing in a riskless instrument), such as a U.S. Treasury bill.

volatility with respect to the market as a whole. If an investor is comfortable investing in the market, then a portfolio with Beta near 1.0 should also be acceptable. Also, Beta gives some measure of how rough a "ride" an investor would experience if she invested in a portfolio. In Table 9.2, all Betas are with respect to the benchmark portfolio, thus the benchmark's own Beta is exactly one. Note that the S&P 500 portfolio has a Beta of almost one, and that in all other respects the S&P 500 and benchmark portfolios are quite similar. This is encouraging—a market value-weighted portfolio such as our benchmark ought to exhibit returns similar to the S&P 500, so the similarity gives us confidence that the stocks in our database are representative of the market as a whole.

The other analysis that the equity fund manager had mentioned was performance attribution. There are several known effects that might result in superior performance of a portfolio, such as the size effect and growth/value effect (Sharpe et al. 1994, Elton & Gruber 1991). If our portfolio's performance is attributable solely to one of these factors, then our model has only discovered previously-known patterns. In this chapter we will only consider the growth/value effect. Figure 9.4 gives an area chart that shows the percentage of the portfolio invested in growth or value stocks during each quarter. Obviously, the composition varies quite a bit, so the data mining models are not simply buying growth or value stocks alone, but rather more complicated patterns are guiding the stock selection process. A similar chart for size shows the same thing: the composition of the portfolio by stock size also varies during the period. We formed a custom growth/value benchmark and a custom size benchmark, both of which we constructed to have the same distribution of assets by growth/value or size, respectively, as our portfolio. The cumulative returns were 109% for the custom growth/value portfolio and 67% for the custom size portfolio. From this we can tell that a significant portion of the performance is attributable to individual stock selection, not to broad weightings like size or growth/value.

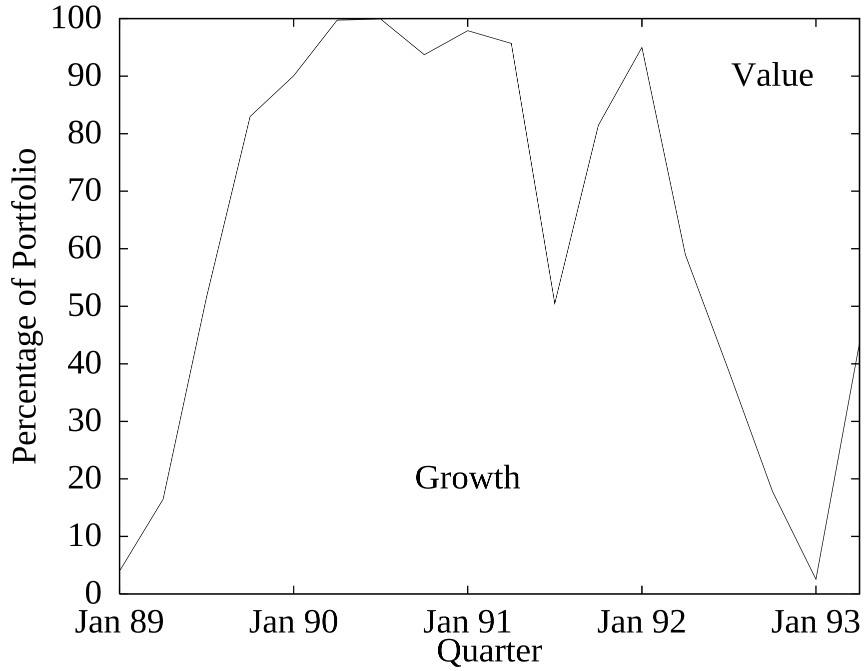

Figure 9.4: The percentage of our portfolio invested in growth and value stocks over time.

9.4 Long/Short Portfolios

In the course of building expertise in portfolio management, we learned of a new strategy that relies more heavily on the use of quantitative methods. By buying the strong performers and selling the underperformers short (essentially, buying a negative quantity) a manager hopes to achieve positive returns while potentially decreasing risks (Jacobs & Levy 1993). Because of low short-selling volume (only 3% of NYSE and NASDAQ volume as of late 1993), underperforming stocks are likely to be less efficiently priced, leaving more room for profit by astute short pickers (Price 1989, Dravo 1993). Short-only portfolios are quite risky, which has motivated portfolios combining both long and short positions. Primarily because of the complicated investment objectives involved in a long/short portfolio, many long/short managers use quantitative methods.

In the financial community, authors who write about long/short trading strategies give few details on the actual mechanics of such strategies. Realistic simulation of long-short trading is significantly more complicated than for long-only strategies. The mechanics involve the use of S&P 500 futures positions to reach a desired Beta and margin accounts for the short positions. The precise method used in our simulation was based on interviews with a local pension fund manager and a business school professor, and a phone call to the Chicago Mercantile Exchange.

Again, it is important to consider results of trading relative to an appropriate benchmark. Jacobs & Levy (1993) suggest Treasury bills as a benchmark for a "market-neutral" long/short portfolio, one with a Beta near zero. Indeed, they manage a long/short fund and are actually compensated based on their performance relative to Tbills (White 1991). Our own long/short portfolio had a small (actually negative) Beta, but it had much higher volatility than Tbills (6.71% vs. 0.49%), so we instead used a combination of S&P 500 futures and Tbills to construct custom benchmarks with the same risk as our own portfolio.

We will not give a detailed analysis of the results for the long/short portfolio. The most important finding was that by combining assets with low correlation (specifically, our long/short portfolio and an S&P 500 portfolio) we can get extra returns with no penalty in risk: adding a 35% investment in long and short portfolios gives 1.72% higher quarterly return than the S&P portfolio with the same standard deviation.

Caution is called for when interpreting results of simulations such as ours. Freeman (1992) gives a good consumer's guide to understanding simulated returns, delineating many possible errors that might lead to optimistically biased results. Although the paper is specific to evaluating portfolios, many of the issues apply to data mining in general, such as avoiding the use of a biased sample for training or evaluation, and the important distinction between the holdout set and the test set. Markowitz & Xu (1994) and Black (1995) mention several ways in which out-of-sample data can be overused, leading to optimistic results. We did use the first fourteen quarters of our out of sample period several times to set a few experimental parameters. The last four quarters constitute a *strict* out-of-sample set, also sometimes called the "out-of-out-of-sample" set, which was only used *once*. Because our portfolios still strongly outperform their benchmarks during the strict out-of-sample year, we do not feel that the previous fourteen quarters of results are optimistic.

9.5 Concluding Remarks

Much of the related work applying neural nets and other nonlinear modeling techniques to the capital markets has focused on a single asset (usually an index or derivative thereof), studying methods for using relatively high frequency timeseries data to predict future prices and generate trading signals (Trippi & DeSieno 1992, Hutchinson, Lo & Poggio 1994). Levin (1996) used a neural network on monthly data to form equitized (positive-Beta) long/short portfolios, generating an excess return of 175.4% during the period August 1990 – February 1995. Levin reports positive results in real-time trading, with live trading expected soon. Apte & Hong (1995) used a similar rule-induction algorithm to form long portfolios on IBM retirement fund data, achieving an excess return of 160% over 5 years of monthly trading. Both studies assumed 1% round-trip transaction costs and used the S&P 500 as an appropriate benchmark. Interested readers may wish to refer to John & Miller (1996) and John et al. (1996*b*) for more details on the long/short and long-only portfolio analyses, respectively.

During this analysis an important issue in the application of data mining to business problems repeatedly manifest itself: understanding the criteria by which the model is evaluated. For example, the entire section on performance attribution resulted from questions by an anonymous reviewer and other financial experts to whom we presented early results. When applying data mining to a problem, a prerequisite for success is to learn the language of the application field and be able to converse about the properties of the models in terms that are important to its target users. This is much more than a minor terminological issue, such as reporting accuracy rather than error rate. The actual data that one must gather to evaluate results can vary significantly across application fields.

Chapter 10

The Future of Data Mining

Broadly, we expect future developments in data mining to lie along two tracks: first, enhancing data mining products by importing more cutting edge research from statistics, machine learning, databases, and visualization, and second, making data mining technologies easier for an average domain expert to use. These two tracks are not completely independent—making the technologies easy to use involves not only writing nice graphical user interfaces and discovering new ways for domain experts to efficiently interact with data mining systems, but also leveraging recent results in research to automate many tasks that an expert data analyst would perform manually.

10.1 Data Mining for the Rest of Us

Certain tasks that confront an expert data analyst beg for some automated assistance. Users of data mining tools in the next few years will find increasing automated help in the tasks of data extraction, data cleaning, attribute selection, and parameter selection discussed earlier in this dissertation.

A current trend in data mining is to move away from the old standard of analyzing flat ASCII files, in which the entire file is loaded into main memory. Currently, many data mining tools can be run on data tables stored in a database, but the mining algorithms do not make intelligent use of the database interface. They still extract excessive amounts of data from the database management system and process the data in temporary storage. This is a very coarse-grained interaction with the database management system. In the future, ideas such as those in Chapter 2 will lead to query languages for data mining, and data warehouses optimized to service such queries will become common. A much finer-grained interaction between the mining algorithm and the database will emerge. This will allow mining algorithms to be parallelized "for free" —simply by running against a parallel database management system, much of the speedup to be gained by parallelizing the algorithm will be achieved with serial data mining algorithms that issue fine-grained queries to a parallel database management system.

Another task that can be partially automated is data cleaning, discussed in Chapter 3. In general, no automated algorithm can safely decide what data is or is not clean, but we can develop algorithms that present "suspicious" data to a user and ask for a decision. Other data cleaning and sanity-checking steps are an entirely manual part of the data mining process today, but in the near future we expect automated "data cleaning assistants", programs that check for common errors and present options to the user on how to fix problems.

Manual attribute selection can be prone to error. For example, in recent work IBM did for an auto manufacturer, the customer had a database of car purchases and wanted to discover the factors that result in a car being a "hit" in the marketplace. The customer wanted to throw out many attributes, but IBM analysts included these attributes in an initial run and found that some were highly predictive. In this case, one lesson learned about data mining is that if the users restrict the attributes to a small set they feel they understand very well, they are very unlikely to get surprising and interesting results from data mining. More generally, as discussed in Chapter 4, selecting the "optimal" set of attributes is more difficult than one would imagine—since all data mining algorithms are somewhat heuristic in nature, actually *removing* an informative attribute from a database can sometimes improve the predictive power of the discovered model. The wrapper idea we discuss is general and we expect to see further research in making efficient implementations for other data mining operations.

Algorithm selection and parameter selection are other problems confronting the novice user. Once a user has decided that a business problem would most profitably be posed as, say, a classification problem, she must still decide which particular algorithm to use to solve the problem and specify certain parameters. Researchers have made some attempts to discover rules that prescribe the use of one algorithm versus another by looking at various characteristics of the data (Brazdil, Gama & Henery 1993), but not with great success. As a result, the best methods for choosing an algorithm and parameter settings are experimental—try several and choose the best, using the appropriate objective function and a holdout set. As long as some criterion can be used to judge how well an algorithm and parameter setting perform on a problem (and this may include some measure of interpretability in addition to the model's error), methods similar to the one proposed in Chapter 5 will apply. We expect this idea to seed development of more efficient versions of this approach to parameter and algorithm selection.

The current trend in the data mining marketplace is to embed data mining technologies into vertical products that are targeted at specific applications, such as stock selection, credit scoring, or fraud detection. The most important and unique aspect of these vertical tools will be the methods used to report and analyze results. This will require the data miners and developers to become experts in their vertical application areas, to learn what kind of reports and graphs are commonly used to report results, as in Chapter 9.

Finally, one issue not covered in the thesis but likely to be important in the near future is choosing a proper sample. The usual motivation for sampling is speed—it takes a long time for many data mining operations to run, so to speed things up we artificially reduce the number of records by taking a sample of the entire database and then mine this sample. People in the data mining business often refer to taking "a statistically valid sample," but usually what is meant by this is simply taking a random sample, which is statistically valid only in the sense that its population is identically distributed with the original database; thus, for some kinds of operations, we expect[1] to get the same results by using the sample as we would have gotten by using the entire database. John & Langley (1996) take a first step towards formalizing

[1] In the statistical sense; that is, if we repeated the process many times and average.

the properties of a desirable sample and give an algorithm designed to prescribe the sample size required. In the future, sampling will likely become transparent to the user and the data mining algorithm will carry it out automatically, internally and at a finer granularity, as in Catlett (1992). His *peepholing* idea is one of several fertile areas of technological development in data mining that promises to have a significant impact on the field in the near future.

10.2 Fertile Technological Developments

There are a variety of other promising research topics currently being investigated. An area needing much more research is the incorporation of the true objective function into the data mining process. For example, in traditional linear regression the model is penalized by the sum of the squared errors of its predictions, but is this really what the user cares about? Similarly, in classification, many algorithms attempt to maximize accuracy, but again is this the right objective function? Current association rule-mining algorithms find rules within some range of confidence and support, but there are other criteria that a user might find more important.

In regression, it is often the case that a more appropriate error function is the sum of the absolute values of all of the errors (called the L1 norm). For example, if the problem is to predict the profit from a potential customer over the next year, it might be appropriate to penalize a mistake of $30K three times as much as a mistake of $10K in the predicted profit, but the squared error function would penalize nine times as much. This has the effect of making the model try hard to avoid making large errors at the expense of making many small errors, which may not be ideal. Even the L1 norm may not be the right objective function—the real objective function should be customized for the problem being solved. For example, in projecting the profit resulting from acquisition of a certain potential customer, overestimating profit may result in the company expending significant resources to acquire the customer, whereas underestimating profit results in the company attempting to acquire a different customer instead, whose associated profit might be only slightly smaller. Thus the cost of errors might not even be symmetric.

Recently, researchers have shown methods for the decomposition of the zero-one loss function (*i.e.*, accuracy) for classification into bias and variance (Friedman 1997, Kohavi & Wolpert 1996). The upshot of this work is a new way of understanding the well-known phenomenon that simple models typically do not perform well (because they lack the complexity to fit the data well) and complex models typically do not perform well (because they have so much complexity they can fit the noise in the training data and generalize poorly). More than just a tool for understanding the behavior of existing algorithms, the bias-variance tradeoff yields insight into a new way of making data mining algorithms more accurate: reducing variance by running the algorithm many times on different samples of the training set, and combining the resulting models.

Stacked generalization (Wolpert 1992) was the first general tool for combining multiple models. After its publication, another technique known as *bagging* also became popular (Breiman 1996*a*). Both methods prescribe a way to take a single training set and get several training sets that come from the same distribution. Experimental results from many researchers indicate that these methods do in fact give combined models with better performance than a single model, on a variety of real-world databases. These results are very recent and only a few commercial products are making use of these technologies. Boosting (Drucker, Schapire & Simard 1993, Schapire 1990) is another method that combines many models into a single predictive model. In contrast to stacking and bagging, which decrease the error of a model by reducing variance, boosting can also decrease bias—even if a data mining algorithm produces models that are by themselves too simple to capture the complex patterns in a database under study, the boosting procedure can produce a set of models that together do capture the patterns in the data.

In addition to studying methods for combining multiple models, a significant area of research is still involved with inventing new mining algorithms and extending existing algorithms by forming hybrids, or by breaking down an algorithm into its components and experimenting with new solutions to one component. Chapters 6–8 all present examples of this latter method of research, respectively investigating continuous density estimation in simple Bayesian classifiers, greedy splitting criteria in

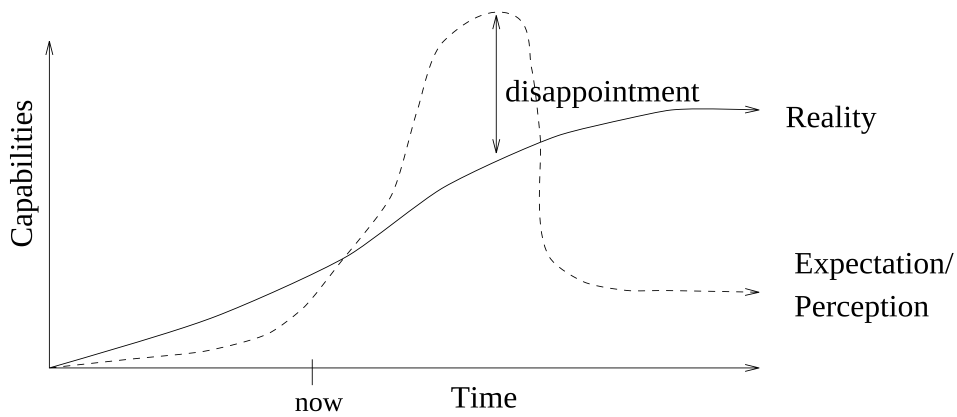

Figure 10.1: Perceived and real capabilities of an over-hyped technology, as a function of time.

univariate classification trees, and greedy splitting criteria in multivariate classification trees. Understanding that no single algorithm can be the best on every problem, the value of such research lies in finding new ways of trading off interpretability, running time, and the quality of the discovered patterns.

10.3 Conclusion

This all sounds very promising—and it is—but there is nothing magical in this technology. Data mining can only discover patterns that already exist in a database, and human analysts are still needed to verify the discovered knowledge and put it to use. "Garbage in, garbage out" still holds as a general rule.

The future of data mining is bright, and it is true that return on investment for data mining projects can be very high. Still, it is important for those in the community to manage expectations. When expectation of the usefulness of a new technology exceeds reality, *i.e.*, when a lot of hype surrounds a new area, there can be a big disappointment when the expectations are not met, and the new technology suffers as a result. Figure 10.1 shows the generic "hype graph," of how high expectations can be the downfall of a new technology. (The graph is based on one used by Gregory Piatetsky-Shapiro in a talk at the second international conference on knowledge discovery and data mining, and a graph by Eric Brethenoux of Gartner Group.)

Note that in the advanced stages of an over-hyped technology, real progress in the development of the technology is reduced by waning interest, as it becomes common knowledge that the technology "doesn't work." Fortuitously, sufficiently many researchers and members of the press covering data mining have had experience with previous waves of hype and disappointment in AI, so that hopefully the data mining community can avoid the disappointment that has come after other new technologies did not live up to their billing.

We feel that the current state of data mining as a technology and an industry is somewhere on the graph just before reality and expectation meet (Figure 10.1). Most data mining products today are extremely close to the programs developed by professors and students just a few years ago. The next few years hold a number of useful developments in data mining products, making them easier to use and allowing them to find more advantageous patterns. The contributions in this dissertation constitute important steps in this direction, but the process of data mining will remain interactive and will still likely require a collaboration between a data mining analyst and a domain expert to define the problem precisely and apply the results.

Bibliography

AAAI-92: Proceedings of the Tenth National Conference on Artificial Intelligence (1992), AAAI Press/MIT Press.

Agrawal, R., Imielinski, T. & Swami, A. (1993), "Mining association rules between sets of items in lage databases", *SIGMOD Record* **22**(2), June, pp. 207–216.

Aha, D. W. (1991), "Instance-based learning algorithms", *Machine Learning* **6**(1), January, pp. 37–66.

Allen, L. E. (1996), "Mining gold from data bases", *Mortgage Banking* **56**(8), May, pp. 99–100.

Almuallim, H. & Dietterich, T. G. (1991), Learning with many irrelevant features, in *AAAI-91: Proceedings of the Ninth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, AAAI Press/MIT Press, pp. 547–552.

Apte, C. & Hong, S. (1995), Predicting equity returns from securities data with minimal rule generation, in U. Fayyad, G. Piatetsky-Shapiro & P. Smyth, eds, *Advances in Knowledge Discovery and Data Mining*, AAAI Press/MIT Press, chapter 22, pp. 537–556.

Bajcsy, R., ed. (1993), *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann.

Barnett, V. & Lewis, T. (1993), *Outliers in Statistical Data*, third edition, John Wiley & Sons, New York.

Ben-Bassat, M. (1982), Use of distance measures, information measures and error bounds in feature evaluation, in P. R. Krishnaiah & L. N. Kanal, eds, *Handbook of Statistics*, Vol. 2, North-Holland Publishing Company, pp. 773–791.

Bennett, K. P. & Mangasarian, O. L. (1992), Neural network training via linear programming, in P. M. Pardalos, ed., *Advances in Optimization and Parallel Computing*, North Holland, Amsterdam, pp. 56–67.

Bers, J. S. (1996), "Maximizing direct mail revenue with data mining", *Bank Systems & Technology* **33**(6), June, p. 44.

Bichsel, M. & Seitz, P. (1989), "Minimum class entropy: A maximum information approach to layered networks", *Neural Networks* **2**, pp. 133–141.

Black, F. (1995), "Estimating expected return", *Financial Analysts Journal* **51**(1), January-February, pp. 168–171.

Boyce, D., Farhi, A. & Weischedel, R. (1974), *Optimal Subset Selection*, Springer-Verlag.

Brazdil, P., Gama, J. & Henery, B. (1993), Comparisons of ML and statistical approaches using meta level learning, in *Proceedings of the ECML93 Workshop on Real-World Applications of Machine Learning*, Vienna, Austria.

Brazdil, P., Gama, J. & Henery, B. (1994), Characterizing the applicability of classification algorithms using meta-level learning, in F. Bergadano & L. D. Raedt, eds, *Machine Learning: ECML-94. European Conference on Machine Learning*, Lectures Notes in Artificial Intelligence, Springer-Verlag, Catania,Italy, pp. 83–102.

Breiman, L. (1996*a*), "Bagging predictors", *Machine Learning* **24**(2), pp. 123–140.

Breiman, L. (1996*b*), "Some properties of splitting criteria", *Machine Learning* **24**(1), pp. 41–47.

Breiman, L., Friedman, J., Olshen, R. & Stone, C. (1984), *Classification and Regression Trees*, Chapman & Hall, New York.

Brent, R. P. (1991), "Fast training algorithms for neural networks", *IEEE Transactions on Neural Networks* **2**(3), May, pp. 346–354.

Brodley, C. E. (1995), Automatic selection of split criterion during tree growing based on node location, *in* Russell & Prieditis (1995), pp. 73–80.

Brodley, C. E. & Friedl, M. A. (1996), Identifying and eliminating mislabeled training instances, in *AAAI-96: Proceedings of the Fourteenth National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, AAAI Press/MIT Press.

Brodley, C. E. & Utgoff, P. E. (1995), "Multivariate decision trees", *Machine Learning* **19**(1), April, pp. 45–76.

Buntine, W. (1992), "Learning classification trees", *Statistics and Computing* **2**, pp. 63–73.

Buntine, W. & Niblett, T. (1992), "A further comparison of splitting rules for decision-tree induction", *Machine Learning* **8**, pp. 75–85.

Cardie, C. (1993), Using decision trees to improve case-based learning, in *Proceedings of the Tenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 25–32.

Caruana, R. & Freitag, D. (1994), Greedy attribute selection, *in* Hirsh & Cohen (1994), pp. 28–37.

Casella, G. & Berger, R. L. (1990), *Statistical Inference*, Wadsworth & Brooks/Cole.

Catlett, J. (1992), Peepholing: choosing attributes efficiently for megainduction, *in* Sleeman & Edwards (1992), pp. 49–54.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W. & Freeman, D. (1988), Autoclass: A Bayesian classification system, in *Machine Learning: Proceedings of the Fifth International Workshop*, Morgan Kaufmann, pp. 54–64.

Cohn, D. (1994), "Improving generalization with active learning", *Machine Learning* **15**(2), pp. 201–221.

Connor, L. (1996), "Mining for data", *CommunicationsWeek*(596), Feb 12, pp. 37–41.

Cover, T. M. & Thomas, J. A. (1990), *Elements of Information Theory*, John Wiley & Sons.

Craven, M. W. & Shavlik, J. W. (1993), Learning symbolic rules using artificial neural networks, *in* Utgoff (1993), pp. 73–80.

Dempster, A. P., Laird, N. M. & Rubin, D. B. (1977), "Maximum likelihood from incomplete data via the EM algorithm", *Journal of the Royal Statistical Society B* **39**, pp. 1–38.

Devijver, P. A. & Kittler, J. (1982), *Pattern Recognition: A Statistical Approach*, Prentice-Hall International.

Devroye, L. (1983), "The equivalence of weak, strong, and complete convergence in l_1 for kernel density estimates", *The Annals of Statistics* **11**, pp. 896–904.

Dougherty, J., Kohavi, R. & Sahami, M. (1995), Supervised and unsupervised discretization of continuous features, *in* Russell & Prieditis (1995).

Draper, N. R. & Smith, H. (1981), *Applied Regression Analysis*, 2nd edition, John Wiley & Sons.

Drastal, G., Meunier, R. & Raatz, S. (1989), Error correction in constructive induction, in A. M. Segre, ed., *Proceedings of the Sixth International Conference on Machine Learning*, Morgan Kaufmann, pp. 81–83.

Dravo, E. (1993), "The long (and short) of short selling", *Financial World*, September 14, pp. 58–60.

Drucker, H., Schapire, R. & Simard, P. (1993), Improving performance in neural networks using a boosting algorithm, *in* Hanson, Cowan & Giles (1993).

Duda, R. & Hart, P. (1973), *Pattern Classification and Scene Analysis*, Wiley.

Edelstein, H. (1996), "Mining data warehouses", *Informationweek*(561), Jan 8, pp. 48–51.

Elton, E. J. & Gruber, M. J. (1991), *Modern Portfolio Theory and Investment Analysis*, fourth edition, John Wiley & Sons, New York.

Fayyad, U. M. & Irani, K. B. (1992), The attribute selection problem in decision tree generation, in *AAAI* (*AAAI-92: Proceedings of the Tenth National Conference on Artificial Intelligence* 1992), pp. 104–110.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1995), From data mining to knowledge discovery, in U. Fayyad, G. Piatetsky-Shapiro & P. Smyth, eds, *From Data Mining to Knowledge Discovery*, AAAI Press/MIT Press.

Fayyad, U., Piatetsky-Shapiro, G. & Smyth, P. (1996), "Data mining and knowledge discovery in databases: An overview", *Communications of the ACM* **39**(11), November.

Finnoff, W., Hergert, F. & Zimmerman, H. G. (1993), "Improving model selection by nonconvergent methods", *Neural Networks* **6**(6), pp. 771–783.

Frawley, W. J., Pietetsky-Shapiro, G. & Matheus, C. J. (1992), "Knowledge discovery in databases: An overview", *AI Magazine* **13**(3), pp. 57–70.

Freeman, J. D. (1992), "Behind the smoke and mirrors: gauging the integrity of investment simulations", *Financial Analysts Journal*, November-December, pp. 26–31.

Friedman, J. H. (1977), "A recursive partitioning decision rule for nonparametric classification", *IEEE Transactions on Computers*, April, pp. 404–408.

Friedman, J. H. (1997), "On bias, variance, 0/1-loss, and the curse of dimensionality", *Knowledge Discovery and Data Mining* **1**(1).

Friedman, N. & Goldszmidt, M. (1996), Building classifiers using Bayesian networks, in *AAAI-96: Proceedings of the Thirteenth National Conference on Artificial Intelligence*, AAAI Press/MIT Press.

Gasser, T., Kneip, A. & Kohler, W. (1991), "A flexible and fast method for automatic smoothing", *Journal of the American Statistical Association* **86**(415), pp. 643–652.

Geiger, D. & Heckerman, D. (1994), Learning gaussian networks, in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence* (*Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence* 1994), pp. 235–243.

Gennari, J. H., Langley, P. & Fisher, D. (1989), "Models of incremental concept formation", *Artificial Intelligence* **40**, pp. 11–61.

Gessaroli, J. (1995), "Data mining: a powerful technology for database marketing", *Telemarketing* **13**(11), May, pp. 64–68.

Ghahramani, Z. & Jordan, M. I. (1993), Supervised learning from real and discrete incomplete data, CLNL '93 Workshop Submission.

Graefe, G. (1993), "Query evaluation techniques for large databases", *ACM Computing Surveys* **25**(2), June, pp. 73–170.

Gray, J., Chaudhuri, S., Bosworth, A., Layman, A., Reichart, D., Venkatrao, M., Pellow, F. & Pirahesh, H. (1996), "Data cube: A relational aggregation operator generalizaing group-by, cross-tab, and sub totals", *Knowledge Discovery and Data Mining* **1**(1).

Greenfeld, N. (1996), "Data mining", *UNIX Review* **14**(5), May, pp. 9–14.

Gupta, A., Harinarayan, V. & Quass, D. (1995), Aggregate-query processing in data warehousing environments, in *Proceedings of the 21st Very Large Databases Conference*, pp. 358–369.

Gupta, H., Harinarayan, V., Rajaraman, A. & Ullman, J. D. (forthcoming), Index selection for OLAP, in *International Conferece on Data Engineering*.

Guyon, I., Boser, B. & Vapnik, V. (1993), Automatic capacity tuning of very large VC-dimension classifiers, *in* Hanson et al. (1993), pp. 147–154.

Hanson, S. J., Cowan, J. & Giles, C. L., eds (1993), *Advances in Neural Information Processing Systems*, Vol. 5, Morgan Kaufmann.

Harinarayan, V., Rajaraman, A. & Ullman, J. (1996), Implementing data cubes efficiently, in *Proceedings of the 1996 ACM-SIGMOD Conference*, pp. 205–216.

Harper, A. M., Duewer, D. L., Kowalski, B. R. & Fasching, J. L. (1976), ARTHUR and experimental data analysis: the heuristic use of a polyalgorithm, in B. R. Kowalski, ed., *Chemometrics: Theory and Application*, American Chemical Society, Washington, D.C., chapter 2, pp. 14–36.

Hastie, T. J. & Tibshirani, R. J. (1990), *Generalized Additive Models*, Chapman and Hall.

Heath, D., Kasif, S. & Salzberg, S. (1993), Induction of oblique decision trees, *in* Bajcsy (1993).

Henrichon, Jr., E. G. & Fu, K.-S. (1969), "A nonparametric partitioning procedure for pattern classification", *IEEE Transactions on Computers* **C-18**(7), July, pp. 614–624.

Hirsh, H. & Cohen, W., eds (1994), *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann.

HNC (1994), "Database mining workstation: The power to predict", Marketing Literature.

Holte, R. C. (1993), "Very simple classification rules perform well on most commonly used datasets", *Machine Learning* **11**(1), April, pp. 63–90.

Huber, P. (1981), *Robust Statistics*, Wiley, New York.

Huber, P. J. (1977), *Robust Statistical Procedures*, Society for Industrial and Applied Mathematics, Pittsburgh, PA.

Hutchinson, J. M., Lo, A. W. & Poggio, T. (1994), "A nonparametric approach to pricing and hedging derivative securities via learning networks", *Journal of Finance* **49**(3), July, pp. 851–889.

Izenman, A. J. (1991), "Recent developments in nonparametric density estimation", *Journal of the American Statistical Association* **86**(413), March, pp. 205–223.

Jacobs, B. I. & Levy, K. N. (1993), "Long/short equity investing", *Journal of Portfolio Management* **20**(1), pp. 52–63.

John, G. H. (1994*a*), Cross-validated C4.5: Using error estimation for automatic parameter selection, Technical Report STAN-CS-TN-94-12, Computer Science Department, Stanford University.

John, G. H. (1994*b*), Finding multivariate splits in decision trees using function optimization, in *Proceedings, Tenth National Conference on Artificial Intelligence*, AAAI Press, p. 1463. Abstract.

John, G. H. (1995), Robust linear discriminant trees, in *Fifth International Workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, pp. 285–291.

John, G. H. & Langley, P. (1995), Estimating continuous distributions in Bayesian classifiers, in P. Besnard & S. Hanks, eds, *Eleventh Annual Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann Publishers, San Mateo.

John, G. H. & Langley, P. (1996), Static versus dynamic sampling for data mining, in E. Simoudis, J. Han & U. Fayyad, eds, *Proceedings, Second International Conference on Knowledge Discovery and Data Mining*, AAAI Press, Menlo Park, CA, pp. 367–370.

John, G. H. & Miller, P. (1996), Building long/short portfolios using rule induction, in *Computational Intelligence in Financial Engineering*, IEEE/IAFE, New York, New York.

John, G. H., Kohavi, R. & Pfleger, K. (1994), Irrelevant features and the subset selection problem, in H. Hirsh & W. Cohen, eds, *Machine Learning: Proceedings of the Eleventh International Conference*, Morgan Kaufmann Publishers, pp. 121–129.

John, G. H., Miller, P. & Kerber, R. (1996*a*), Stock selection using Recon, in Y. Abu-Mostafa, J. Moody, P. Refenes & A. Weigend, eds, *Neural Networks in Financial Engineering*, World Scientific.

John, G. H., Miller, P. & Kerber, R. (1996*b*), "Stock selection using rule induction", *IEEE Expert* **11**(5), October, pp. 52–58.

Jordan, M. I. & Jacobs, R. A. (1993), Supervised learning and divide-and-conquer: A statistical approach, *in* Utgoff (1993).

Kanal, L. (1979), "Problem-solving models and search strategies for pattern recognition", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-1**(2), April, pp. 193–201.

Kearns, M. (1993), Efficient noise-tolerant learning from statistical queries, in *Proceedings of the Twenty-Fifth Annual ACM Symposium on the Theory of Computing*, ACM, New York, New York, pp. 392–401.

Kerber, R. (1991), Learning classification rules from examples, in *Proceedings of the AAAI-91 Workshop on Knowledge Discovery in Databases*, AAAI.

Kerber, R. (1992), Chi Merge: Discretization of numeric attributes, in *AAAI* (*AAAI-92: Proceedings of the Tenth National Conference on Artificial Intelligence 1992*), pp. 123–128.

Kibler, D. & Aha, D. W. (1987), Learning representative exemplars of concepts: An initial case study, in *Machine Learning: Proceedings of the Fourth Workshop*, Morgan Kaufmann Publishers, San Mateo, CA, pp. 24–30.

Kira, K. & Rendell, L. (1992*a*), The feature selection problem: Traditional methods and a new algorithm, in *AAAI* (*AAAI-92: Proceedings of the Tenth National Conference on Artificial Intelligence 1992*), pp. 129–134.

Kira, K. & Rendell, L. (1992*b*), A practical approach to feature selection, *in* Sleeman & Edwards (1992), pp. 249–258.

Kohavi, R. & John, G. H. (1995), Automatic parameter selection by minimizing estimated error, in A. Prieditis & S. Russell, eds, *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann Publishers, San Francisco.

Kohavi, R. & John, G. H. (forthcoming), "Wrappers for feature subset selection", *Artificial Intelligence Journal*.

Kohavi, R. & Wolpert, D. H. (1996), Bias plus variance decomposition for zero-one loss fuctions, in *Machine Learning: Proceedings of the Thirteenth International Conference*.

Kononenko, I. (1993), "Inductive and bayesian learning in medical diagnosis", *Applied Artificial Intelligence* **7**, pp. 317–337.

Koutsougeras, C. & Papachristou, C. A. (1988), Training of a neural network for pattern classification based on an entropy measure, in *IEEE International Conference on Neural Networks*, IEEE Press, pp. 247–254.

Lang, K. J., Waibel, A. H. & Hinton, G. E. (1990), "A time-delay neural network architecture for isolated word recognition", *Neural Networks* **3**, pp. 23–43.

Langley, P. & Sage, S. (1994*a*), Induction of selective bayesian classifiers, in *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, Morgan Kaufmann, Seattle, WA, pp. 399–406.

Langley, P. & Sage, S. (1994*b*), Oblivious decision trees and abstract cases, in *Working Notes of the AAAI94 Workshop on Case-Based Reasoning*, Seattle, WA.

Langley, P. & Simon, H. A. (1995), "Applications of machine learning and rule induction", *Communications of the ACM* **38**, November, pp. 55–64.

Levin, A. U. (1996), Stock selection via nonlinear multi-factor models, in *Advances in Neural Information Processing Systems*, Morgan Kaufmann Publishers, San Francisco, California.

Levy, A. Y. (1993), Irrelevance Reasoning in Knowledge Based Systems, PhD thesis, Stanford University.

Lin, Y. K. & Fu, K. S. (1983), "Automatic classification of cervical cells using a binary tree classifier", *Pattern Recognition* **16**(1), pp. 69–80.

Linhart, H. & Zucchini, W. (1986), *Model Selection*, Wiley.

Littlestone, N. (1988), "Learning quickly when irrelevant attributes abound: a new linear-threshold algorithm", *Machine Learning* **2**(4), pp. 285–318.

Loh, W.-Y. & Vanichsetakul, N. (1988), "Tree-structured classification via generalized discriminant analysis", *Journal of the American Statistical Association* **83**(403), September, pp. 715–725.

López de Mántaras, R. (1991), "A distance-based attribute selection measure for decision tree induction", *Machine Learning* **6**, pp. 81–92.

Lynch, P. (1993), *Beating the Street*, Simon & Schuster, New York.

Mallows, C. L. (1973), "Some comments on c_p", *Technometrics* **15**, pp. 661–675.

Marill, T. & Green, D. M. (1963), "On the effectiveness of receptors in recognition systems", *IEEE Transactions on Information Theory* **9**, pp. 11–17.

Markowitz, H. M. & Xu, G. L. (1994), "Data mining corrections", *Journal of Portfolio Management*, Fall, pp. 60–69.

Mendenhall, W. & Sincich, T. (1988), *Statistics for the engineering and computer sciences*, Dellen Publishing Company, San Francisco, CA.

Michalewicz, Z. (1992), *Statistical and Scientific Databases*, Ellis Horwood.

Michie, D., Spiegelhalter, D. J. & Taylor, C. C. (1994), *Machine Learning, Neural and Statistical Classification*, Prentice Hall.

Miller, A. J. (1990), *Subset Selection in Regression*, Chapman and Hall.

Mingers, J. (1989), "An empirical comparison of selection measures for decision-tree induction", *Machine Learning* **3**, pp. 319–342.

Moore, A. & Lee, M. (1994*a*), Efficient algorithms for minimizing cross-validation error, *in* Hirsh & Cohen (1994), pp. 190–198.

Moore, A. W. & Lee, M. S. (1994*b*), Efficient algorithms for minimizing cross validation error, Unpublished report.

Moore, A. W., Hill, D. J. & Johnson, M. P. (1992), An empirical investigation of brute force to choose features, smoothers and function approximators, in *Computational Learning Theory and Natural Learning Systems Conference*.

Moret, B. M. (1982), "Decision trees and diagrams", *ACM Computing Surveys* **14**(4), pp. 593–623.

Morgan, J. N. & Messenger, R. C. (1973), *THAID: a sequential analysis program for the analysis of nominal scale dependent variables*, University of Michigan.

Mucciardi, A. N. & Gose, E. E. (1971), "A comparison of seven techniques for choosing subsets of pattern recognition properties", *IEEE Transactions on Computers* **C-20**(9), September, pp. 1023–1032.

Murphy, P. M. & Aha, D. W. (1994), "UCI repository of machine learning databases", Available by anonymous ftp to `ics.uci.edu` in the `pub/machine-learning-databases` directory.

Murthy, S. K., Salzberg, S. & Kasif, S. (1993), "OC1", Available by anonymous ftp in `blaze.cs.jhu.edu:pub/oc1`.

Murthy, S., Kasif, S. & Salzberg, S. (1994), "A system for induction of oblique decision trees", *Journal of Artificial Intelligence Research* **2**, pp. 1–32.

Narendra, P. M. & Fukunaga, K. (1977), "A branch and bound algorithm for feature subset selection", *IEEE Transactions on Computers* **C-26**(9), September, pp. 917–922.

Neter, J., Wasserman, W. & Kutner, M. H. (1990), *Applied Linear Statistical Models*, 3rd edition, Irwin: Homewood, IL.

Nilsson, N. (1980), *Principles of Artificial Intelligence*, Morgan Kaufmann.

Pazzani, M. (1995), Searching for attribute dependencies in bayesian classifiers, in D. H. Fisher & H. Lenz, eds, *Fifth International workshop on Artificial Intelligence and Statistics*, Ft. Lauderdale, FL, pp. 424–429.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. (1992), *Numerical Recipes in C: The Art of Scientific Computing*, second edition, Cambridge University Press.

Price, K. (1989), "Long on safety: don't sell short selling short", *Barron's*, November 6, pp. 16,38.

Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence (1994), Morgan Kaufmann, Seattle, WA.

Qing-Yun, S. & Fu, K. S. (1983), "A method for the design of binary-tree classifiers", *Pattern Recognition* **16**(6), pp. 593–603.

Quinlan, J. R. (1986), "Induction of decision trees", *Machine Learning* **1**, pp. 81–106.

Quinlan, J. R. (1993), *C4.5: Programs for Machine Learning*, Morgan Kaufmann.

Richman, D. (1996), "Data mining chisels its niche", *Computerworld* **30**(5), Jan 29, p. 49.

Rosen, M. (1996), "Data mining digs deeper", *Midrange Systems* **9**(9), Jun 14, pp. 1, 68.

Ross, J. R. (1996), "Data mining: Digging deeper for information treasures", *Stores* **78**(5), May, pp. 66–68.

Rousseeuw, P. J. & Leroy, A. M. (1987), *Robust Regression and Outlier Detection*, John Wiley and Sons, New York.

Russell, S. & Prieditis, A., eds (1995), *Machine Learning: Proceedings of the Twelfth International Conference*, Morgan Kaufmann.

Sahami, M. (1993), Learning non-linearly separable boolean functions with linear threshold unit trees and madaline-style networks, in *AAAI-93: Proceedings of the Eleventh National Conference on Artificial Intelligence*, American Association for Artificial Intelligence, AAAI Press/MIT Press, pp. 335–341.

Sahami, M., Hearst, M. & Saund, E. (1996), Applying the multiple cause mixture model to text categorization, in *ICML 96: Proceedings of the Thirteenth International Conference on Machine Learning*, Morgan Kaufmann, pp. 435–443.

Sankar, A. & Mammone, R. J. (1991), Optimal pruning of neural tree networks for improved generalization, in *IJCNN-91-SEATTLE: International Joint Conference on Neural Networks*, IEEE Press, Seattle, WA, pp. II: 219–224.

Sarawagi, S., Agrawal, R. & Gupta, A. (1996), On computing the data cube, Technical report, IBM, Almaden Research Center, San Jose, CA.

Schaffer, C. (1993), "Selecting a classification method by cross-validation", *Machine Learning* **13**, pp. 135–143.

Schaffer, C. (1994), A conservation law for generalization performance, *in* Hirsh & Cohen (1994), pp. 259–267.

Schapire, R. E. (1990), "The strength of weak learnabilty", *Machine Learning* **5**, pp. 197–227.

Sethi, I. K. (1990), "Entropy nets: from decision trees to neural networks", *Proceedings of the IEEE* **78**(10), October, pp. 1605–1613.

Sethi, I. K. & Sarvarayudu, G. P. R. (1982), "Hierarchical classifier design using mutual information", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-4**(4), July, pp. 441–445.

Sharpe, W. F. (1994), "The Sharpe ratio", *Journal of Portfolio Management* **21**(1), Fall, pp. 49–58.

Sharpe, W. F., Alexander, G. J. & Bailey, J. V. (1994), *Investments*, fifth edition, Prentice Hall, Englewood Cliffs, New Jersey.

Shoshani, A. (1982), Statistical databases: characteristics, problems, and some solutions, in *Proceedings of the Eighth Conference on Very Large Databases*, Mexico City, pp. 208–213.

Siedlecki, W. & Sklansky, J. (1988), "On automatic feature selection", *International Journal of Pattern Recognition and Artificial Intelligence* **2**(2), pp. 197–230.

Silberschatz, A., Stonebreaker, M. & Ullman, J. (1995), "Database research: Achievements and opportunities into the 21st century", Report of an NSF Workshop on the Future of Database Systems Research.

Silverman, B. W. (1986), *Density estimation for statistics and data analysis*, Chapman and Hall, New York.

Simoudis, E. (1996), "Data mining: A reality check", *IEEE Expert* **11**(5), October, pp. 26–33.

Skalak, D. (1994), Prototype and feature selection by sampling and random mutation hillclimbing, *in* Hirsh & Cohen (1994).

Sleeman, D. & Edwards, P., eds (1992), *Machine Learning: Proceedings of the Ninth International Workshop*, Morgan Kaufmann.

Smyth, P., Gray, A. & Fayyad, U. M. (1995), Retrofitting decision tree classifiers using kernel density estimation, *in* Russell & Prieditis (1995), pp. 506–514.

Specht, D. F. & Romsdahl, H. (1994), Experience with adaptive pobabilistic neural networks and adaptive general regression neural networks, in *IEEE International Conference on Neural Networks*, Orlando, FL.

Srikant, R. & Agrawal, R. (1996), Mining sequential patterns: generalizations and performance improvements, in S. Apers, M. Bouzeghoub & G. Gardarin, eds, *Proceedings of the Fifth International Conference on Extending Database Technology*, pp. 3–17.

Thrun, S. B. et al. (1991), The monk's problems – a performance comparison of different learning algorithms, Technical Report CMU-CS-91-197, Carnegie Mellon University.

Trippi, R. R. & DeSieno, D. (1992), "Trading equity index futures with a neural network", *Journal of Portfolio Management* **19**(1), Fall, pp. 27–33.

Ullman, J. D. (1988), *Principles of Database and Knowledge-Base Systems: Volume 1: Classical Database Systems*, Addison-Wesley.

Utans, J. & Moody, J. (1991), Selecting neural network architectures via the prediction risk: Application to corporate bond rating prediction, in *The First International Conference on Artificial Intelligence Applications on Wall Street*, IEEE Computer Society Press.

Utgoff, P. (1994), An improved algorithm for incremental induction of decision trees, *in* Hirsh & Cohen (1994), pp. 318–326.

Utgoff, P., ed. (1993), *Proceedings of the Tenth International Conference on Machine Learning*, Morgan Kaufmann.

Vafaie, H. & De Jong, K. (1992), Genetic algorithms as a tool for feature selection in machine learning, in *Fourth International Conference on Tools with Artificial Intelligence*, IEEE Computer Society Press, pp. 200–203.

Venables, W. N. & Ripley, B. D. (1994), *Modern Applied Statistics with S-Plus*, Springer-Verlag, New York.

Wahba, G. (1990), *Spline Functions for Observational Data*, CBMS-NSF Regional Conference Series, SIAM, Philadelphia.

Weigend, A. S. (1994), On overfitting and the effective number of hidden units, in M. C. Mozer, P. Smolensky, D. S. Touretzky, J. L. Elman & A. S. Weigend, eds, *Proceedings of the 1993 Connectionist Models Summer School*, Erlbaum Associates, Hillsdale, NJ, pp. 335–342.

Weigend, A. S. (1996), Clearning, in Y. Abu-Mostafa, J. Moody, P. Refenes & A. Weigend, eds, *Neural Networks in Financial Engineering*, World Scientific, pp. 511–522.

Weisberg, S. (1985), *Applied Linear Regression*, John Wiley & Sons.

Whang, K.-Y., Kim, S.-W. & Wiederhold, G. (1994), "Dynamic maintenance of data distribution for selectivity estimation", *VLDB Journal*, January, pp. 29–51.

White, A. P. & Liu, W. Z. (1994), "Bias in information-based measures in decision tree induction", *Machine Learning* **15**(3), June, pp. 321–329.

White, J. A. (1991), "How Jacobs and Levy crunch stocks for buying – and selling", *The Wall Street Journal*, March 20, p. C1.

Wolpert, D. H. (1992), "Stacked generalization", *Neural Networks* **5**, pp. 241–259.

Wolpert, D. H. (1994), The relationship between PAC, the statistical physics framework, the Bayesian framework, and the VC framework, Technical report, The Santa Fe Institute, Santa Fe, NM.

Xu, L., Yan, P. & Chang, T. (1989), Best first strategy for feature selection, in *Ninth International Conference on Pattern Recognition*, IEEE Computer Society Press, pp. 706–708.

Zhou, X. J. & Dillon, T. S. (1991), "A statistical-heuristic feature selection criterion for decision tree induction", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(8), pp. 834–841.