Knowledge Discovery in Databases for Intrusion Detection, Disease Classification and Beyond

by

Gideon Berger

A dissertation submitted in partial fulfillment of the requirements for the degree

of Doctor of Philosophy

Department of Computer Science

Graduate School of Arts and Science

New York University

January 2001

Dissertation Advisor

Acknowledgments

I would like to take this opportunity to thank two people without whom this work, very literally, would not have been possible. First, I would like to thank Alex Tuzhilin. Alex introduced me to the field of data mining in general, and to the area of pattern discovery in temporal databases in particular. Without his involvement, not only would this work have not reached its fruitful conclusion but the approach we took would probably have never even crossed my mind.

Second, I would like to thank Bud Mishra. Bud introcuded me to Freidman's work and to Multivariate Adaptive Regression Splines technique. He also gave me the first paper I ever read about Stein shrinkage and the bias/variance tradeoff. I cannot overstate the importance of the guidance and mentoring that he has provided to me over the preceeding years. I would also like to thank him for the host of lessons he has taught me about issues far removed from the topic of this thesis. Specifically, I have learned volumes about game theory, intellegent agents, and bioinformatics. He has always been willing to share his ideas with me and all of his students about the wide variety of research topics that he works on, and I feel that as a result I have gained a new and deeper understanding of how difficult problems are best attacked and novel research is done. I am confident that these lessons will serve me well as I go forward. I truly enjoyed my time spent working with Bud and would welcome the opportunity to collaborate with him on other research projects in the future.

I am deeply in debt for the love, support and encouragement that my family has given me throughout the course of my academic career. They have instilled in me a great appreciation for and a great love of learning. For those I will always be grateful.

My wife Cassandra I can't thank enough for the inspiration, enthusiasm and comfort she offered while I completed this thesis. She gave me confidence when mine was lacking that I could write a thesis that I would be proud of. And I have.

Abstract

Knowledge Discovery in Databases for Intrusion Detection, Disease Classification and Beyond

As the number of networked computers and the amount of sensitive information available on them grows there is an increasing need to ensure the security of these systems. Passwords and encryption have, for some time, provided an important initial defense. Given a clever and malicious individual these defenses can, however, often be circumvented. Intrusion detection is therefore needed as another way to protect computer systems.

This thesis describes a novel three stage algorithm for building classification models in the presence of nonstationary, temporal, high dimensional data, in general, and for detecting network intrusion detections, in particular. Given a set of training records the algorithm begins by identifying "interesting" temporal patterns in this data using a modal logic. This approach is distinguished from other work in this area where frequent patterns are identified. We show that when frequency is replaced by our measure of "interestingness" the problem of finding temporal patterns is NP-complete. We then offer an efficient heuristic approach that has proven experimentally effective.

Having identified interesting patterns, these patterns then become the predictor variables in the construction of a Multivariate Adaptive Regression Splines (MARS) model. This approach will be justified by its ability to capture complex nonlinear

relationships between the predictor and response variables which is comperable to other heuristic approaches such as neural networks and classification trees, while offering improved computational properties such as rapid convergence and interpretability.

After considering several approaches to the problems of overfitting which is inherent when modelling high dimensional data and nonstationarity, we describe our approach to addressing these issues through the use of truncated Stein shrinkage. This approach is motivated by showing the inadmissability of the maximum likelihood estimator (MLE) in the high dimensional (dimension ≥ 3) data.

We then discuss the application of our approach as participants in the 1999 DARPA Intrusion Detection Evaluation where we exhibited the benefits of our approach.

Finally, we suggest another area of research where we believe that our work would meet with similar success, namely, the area of disease classification.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Commentators have coined the phrase "information revolution" to describe a series of technological advancements beginning with the microprocessor and culminating with the internet and other information dissemination media that allow large quantities of information to be rapidly stored and distributed. This language has been chosen to suggest a similarity in societal impact to previous periods of technological change like the industrial revolution. Amidst all of this hyperbole have come a host of predictions about how this new technology will impact all aspects of life from how governments govern, to how corporations function, to how individuals lead their lives. It has been suggested that tyrannical regimes will no longer be able to close their borders and suppress their citizens as information about the benefits of freedom will be widely and freely available via the internet. Businesses are forecast to change in a wide variety of ways. Suppliers will receive real time notification of the need for additional parts thus mitigating the need for large inventories while achieving better capacity utilization. Petroleum companies will be able to store and review large volumes of geological data to better inform their investments in oil drilling expeditions. Companies will be able to better manage risk as their ability to identify risk factors increases. They will be better able to target potential customers through direct marketing as people's buying habits are analyzed by credit card companies and internet advertising firms. Individuals' lives will be changed as well. Access to economic information and the financial markets has resulted in a proliferation of individuals investing in the stock market. People are no longer limited to buying goods available in their neighborhood, but rather now have access to stores all over the world. Prices can be compared quickly and easily between companies to ensure a good deal.

While these benefits are all promised by the information revolution they are predicated on our ability to aggregate and analyze all of this data produced by our new technologies. The aggregation of large quantities of data has been made possible by the evolution of database technology and the decreased cost of digital storage media. The analysis of large quantities of data is the topic of this thesis.

Because of the growth in database size and use, knowledge discovery in databases (KDD), often called data mining, has emerged in the 1990's as a visible research area with the explicit goal of developing tools to facilitate the discovery of higher-level information or knowledge in large databases. The KDD process is often described as the "non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data." The discovered knowledge can be rules describing properties of the data, frequently occurring patterns, clusters of objects in the database, etc.

Broadly speaking, there are two kinds of patterns discovered by data mining algorithms: predictive and descriptive[35]. Predictive algorithms are built to solve a specific problem of predicting one or more attributes in a database based on the rest. Predictive patterns are not always attempts at predicting 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 attributes. By contrast, the point of descriptive algorithms is simply to present interesting patterns that a domain expert might not already know. In other words, with descriptive patterns there are no right answers only interesting ones. Because of this, descriptive 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. Our work, the work described in this thesis, has focused on addressing problems in both of these areas, applied to a specific class of data namely, non-stationary, temporal, high dimensional data.

In some domains the distribution of database records is stationary. For example, Mobil Oil Corporation has developed a data warehouse capable of storing over 100 terabytes of data related to oil exploration[36]. One could imagine that Mobil would use geological data from known oil wells in order to predict whether unexplored land might be promising for future oil exploration. In building a model of which geological features predict the existence of underground oil the modelers do not have to worry that data gathered in a certain time and place will be useless in predicting oil in another time and place. If the existence of certain minerals correlates highly with the presence of underground oil in the North Sea then this correlation will almost certainly also hold in the Indian Ocean. If this correlation holds in 1970 it also holds in 2000. In other domains, however, this assumption that data is stationary is flawed. In these domains one cannot simply use training data (e.g. data from an existing oil well) to build a model of which geological features predict the existence of underground oil, and then use this model to make predictions about out of sample data (e.g. yet unexplored land). The distribution of feature values varies between the in-sample and out-of-sample data thus rendering predictions based on in-sample data flawed. In our work we have expanded on tools developed within the statistics community in order to addressed this particular issue.

In some databases there is no interaction between a given database record and the records that come before and after it. For example, one could imagine in a supermarket's database that each record corresponds to the basket of items purchased by a customer as well as demographic information about that customer perhaps available if the purchase was made by credit card. This information could be used in several ways. First, the supermarket would like to determine if there are any patterns in customer purchases in order to improve store layout. One well documented supermarket buying pattern is the correlation between the purchase of diapers and beer. With this knowledge a supermarket may want to put beer close to diapers on the store shelves in order to make shopping faster. Additionally, the supermarket might want to use demographic information about its customers for marketing purposes. At any rate, the data mining process in this case is simplified by the fact that when looking for patterns in the database each record can be considered individually. The purchases and demographics of one customer have no impact on the purchases or demographics of another. In other domains, however, there is a causal relationship between database records. In some cases this is the result of temporal features in the data. In other words, when analyzing the data it is important to account for the time ordering of the database records. In other cases, this causal relationship is the result of physical characteristics of the data. For example, when analyzing human genome data it is never enough to know that a given nucleotide occurred at a given location on a given chromosome without knowing the nucleotide sequences that preceded and followed it. This vastly complicates the data mining process in that we are no longer simply looking for patterns within a given database record but also for patterns amongst different records.

One's ability to build predictive models of data is fundamentally dependent on two factors - the complexity of the underlying (true) model, and the quality of the available training data. Once a problem domain has been selected the complexity of the underlying model becomes an issue exogenous to the modeling process and therefore an issue over which the modeler has little impact except for ensuring that the techniques being used are robust enough to handle the domain's inherent complexities. The quality and size of the training data, while typically limited by real world constraints, must be addressed in a more proactive manner. As mentioned previously, nonstationarity is a qualitative feature of data that we address. Another important issue is the size of training data. The size of the training data is intimately related to the dimensionality of that data. In most complex domains the number of independent variables and thus the number of degrees of freedom is very large. Despite the availability of thousands or even millions of training records this results in poor coverage of the space being modeled which in turn results in high variance in the models constructed. Virtually all techniques for dealing with small sample statistics involve some final pruning phase where increased model bias is traded for decreased variance through a reduction in the number of degrees of freedom. We have explored many of these alternatives and our results using truncated shrinkage will be discussed in detail.

The two problem domains that we wish to address are network intrusion detection and cancer classification. While, on the surface, these problems seem completely unrelated the algorithmic issues related to both problems are actually quite similar in that they both involve the mining of temporal, non-stationary, high dimensional data. The problem of network intrusion detection involves analyzing computer network activity and attempting to identify the existence of intrusive or malicious behavior. In building a model of both normal and malicious activity we cannot assume that our data is stationary. First, as networks and network protocols evolve the activities of normal users will change. Second, as intruders become more sophisticated the activity patterns exhibited by malicious individuals will also change. Therefore, we must account for these changes in building our model. Or more specifically, we must allow for these changes when using a model which was built based on a given network, at a specific time, when trying to identify intrusions on a different network or at a different time. Furthermore, if each record in our data represents a single network connection, it is important that we analyze relationships between these records since intrusions typically do not occur in a single network connection but rather are the result of a series of actions taken over many connections.

Cancer classification has been traditionally based on morphological appearance of the tumor. This approach has two significant limitations. First it is subject to potential human error and second it is too imprecise, i.e. there are almost certainly cancer classes that have yet to be discovered. Since cancer is a disease caused by genetic mutations we take a genetic approach to cancer classification. That is, we consider the expression of genes as RNA in both healthy and diseased individuals and attempt to identify genetic patterns that both distinguish healthy from cancerous cells as well as different classes of cancerous cells. In addressing this problem, however, we cannot assume that our data is non-stationary. Gene expression inevitably differs within groups of individuals. If we use a healthy individual to model what constitutes normal gene expression we may incorrectly predict that another individual has cancer because their gene expression differs from our first healthy individual. In reality this difference may simply be due to normal variances in the population. Therefore, when building a model of healthy and cancerous gene expression we must account for differences within the normal population as well as temporal differences since gene expression changes as one ages. Additionally, if we view our data as a sequence of gene expression levels it is important to consider interactions between these levels as many cancers are polymorphic, that is they involve interacting mutations in more than one gene.

1.1 Problem Statement and Our Approach

Classification involves learning a function that maps (classifies) a data item into one of several predefined classes. Both intrusion detection and cancer classification are fundamentally classification problems. In both cases we build a function of a set of predictor variables in order to predict the value of one or several predicted variables.

In cancer classification, each record in our data represents the gene expression level for one particular gene. In addition, we augment each record with a set of features that capture the relationship between the expression of the gene in the given record and the expression levels of other genes. These additional features are the ones deemed to be most relevant in classifying cancer. Once these features have been selected a classification model is built based on a set of in sample (already labeled) data and we again employ shrinkage in order to addresses the issues of overfitting, non-stationarity, and combining classifiers.

Given that the classification process is virtually identical in both of these problem domains, throughout this thesis we will present the details of our approach in the context of solving the generic classification problem in the presence of non-stationary, high dimensional data in which interactions exist amongst the individual records. The algorithm is divided into three steps: feature selection, classification, and shrinkage.

1.1.1 Feature Selection

Initially, our data consists of an ordered set of database records, each with a set of features deemed relevant, given some domain knowledge, for predicting the value of some dependant variable. In our data there are also important relationships between different records. Therefore, our goal in this initial step is to extract from the data a set of temporal features that are most relevant for predicting the value of the dependent variable. To accomplish this we discover patterns using propositional linear temporal logic and define an interestingness measure over these patterns. We have chosen to use propositional linear temporal logic both because of its ability to express the types of patterns we are looking for as well as the fact that we have devised an efficient algorithm for finding such patterns. In some domains it would be useful to use a more expressive predicate logic. This, however, may result in an explosive growth in the computational complexity over our pattern search space. One aspect of future research will be to develop efficient algorithms over more expressive logics. Given a language for describing the patterns we additionally define an interestingness measure over these patterns. The precise interestingness measure used is domain dependent. In general, it is a measure of how much the occurrence of the pattern correlates with the occurrence of a single value of the predicted variable.

These newly discovered patterns along with the features from the original database records become the independent variables of the next stage of our approach - classification.

1.1.2 Classification

Classification is a problem that is well known to members of the data mining community. The most widely used classification technique is decision trees. There are more powerful (in the sense that they are able to represent a more robust class of functions) classification techniques like neural networks, however, practitioners often shy away from them due to their computational complexity and lack of transparency. One important feature of a classifier that is important to data miners is that the resulting function be ultimately understandable. We like to be able to understand why a prediction made by our classifier was made so that we can better understand relationships that exist in our problem domain. Neural networks are ultimately a black box and while their predictions may be accurate they lead to little insight about the problem at hand.

We have settled on an alternative technique known as **MARS** (Multivariate Adaptive Regression Splines) [41]. This is not a novel technique - it was developed over 10 years ago. However, at that time practitioners generally lacked the computational resources to use this technique and it has been generally overlooked in the interim. **MARS** can be described as an extension of decision trees. It, however, is a nonlinear technique that overcomes many of the shortcomings of standard decision trees while remaining computationally tractable and ultimately interpretable.

1.1.3 Overfitting and Combining Forecasts

In all classification techniques, the introduction of additional degrees of freedom (in decision trees or **MARS** via additional splitting) reduces the in sample error (bias) of the model while increasing the model variance. This frequently results in poor approximations of out of sample data. To address this problem virtually all classification methods include some technique for reducing the model bias - typically via reduction in its degrees of freedom. In both CART[16] and C4.5[66] this is done via a pruning technique which replaces pairs of nodes with their parent node in the tree. This results in increased bias in the model while reducing its variance and, in practice, reduced out-of-sample error. In his original paper Freidman suggested a similar technique be employed when building a **MARS** model. He suggested an iterative procedure in which, in each iteration, a single basis function is removed from the model. The decision as to which basis function to remove is determined by which removal results in the smallest increase in in-sample error. At the conclusion of this process one has a sequence of models each with one less basis function (and therefore at least one less degree of freedom) than the previous. From this sequence the best model is chosen.

Combining forecasts is typically done via averaging resulting in a maximum likelihood estimator (MLE). To evaluate the correctness of this approach consider the more general situation of trying to estimate a parameter T by $t(x)$. If $\mathsf{E}[\![t(x)]\!] = T$, then $t(x)$ is said to be an unbiased estimator of T and a measure of the precision of this estimator is $\mathsf{E}[\![t(x) - T]\!]^2$, i.e. its variance. If, instead, $Et(x) \neq T$, then $t(x)$ is known as a biased estimator of T. A measure of its precision is still $\mathsf{E}[\![t(x) - T]\!]^2$, but now since $Et(x) \neq T$, this quantity is not the variance, but rather is known as the mean squared error (abbreviated, MSE). We now show[70],

$$
\begin{aligned}
\mathsf{E}[\![[t(x) - \theta]^2]\!] &= \mathsf{E}[\![[t(x) - \mathsf{E}[\![t(x)]\!] + \mathsf{E}[\![t(x)]\!] - \theta]^2]\!] && (1.1)\\
&= \mathsf{E}[\![[t(x) - \mathsf{E}[\![t(x)]\!]]^2]\!] + (\mathsf{E}[\![t(x)]\!] - \theta)^2 + && (1.2)\\
&\quad 2(\mathsf{E}[\![t(x)]\!] - \theta)\mathsf{E}[\![t(x) - \mathsf{E}[\![t(x)]\!]]\!] && (1.3)\\
&= \mathsf{E}[\![[t(x) - \mathsf{E}[\![t(x)]\!]]^2]\!] + (\mathsf{E}[\![t(x)]\!] - \theta)^2 && (1.4)\\
&= var[t(x)] + (\mathsf{E}[\![t(x)]\!] - \theta])^2 && (1.5)\\
&= var[t(x)] + [Bias(t)]^2 && (1.6)
\end{aligned}
$$

James and Stein showed (Stein, 1956) that in the case where x is of dimension three or greater, by sacrificing increased bias for decreased variance we can achieve uniformly smaller MSE[49]. In other words the MLE is inadmissible. We have extended Stein's original work for shrinking linear models to the nonlinear setting appropriate for our classification technique.

1.2 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 introduces the field of data mining and suggests several areas in the field that will be useful in our domains of interest. Chapter 3 presents the problem of finding interesting patterns in data. This chapter begins with a discussion of an approach for finding frequent patterns in data. Frequent patterns are, however, limited in their expressiveness and we therefore extend this approach to discover a more robust class of patterns. The computational difficulties and our algorithm are discussed here. Chapter 4 offers a survey of classification techniques along with their various strengths and weaknesses. Chapter 5 is a detailed discussion of the classification technique we have used, MARS, addressing theoretical, computational and implementation issues. Chapter 6 discusses various approaches to reducing model variance including bagging and boosting, as well as a detailed analysis of shrinkage from both a frequentist and Bayesian view. Chapter 7 introduces the problem on network intrusion detection, surveying existing approaches as well as our approach. Chapter 8 discusses the results from our involvement in the 1998 DARPA Intrusion Detection Evaluation. Chapter 9 suggests an approach to applying our techniques to the problem of cancer classification. Finally, Chapter 10 summarizes this thesis and outlines directions for future work.

Chapter 2

The KDD Process

A database is a reliable store of information. One of the prime purposes of such a store is the efficient retrieval of information. This retrieved information is not necessarily a copy of information stored in the database, rather, it is information that can be inferred from the database. From a logical perspective, two inference techniques can be distinguished[46].

1. Deduction is a technique to infer information that is a logical consequence of the information in the database. Most database management systems (DBMSs), such as relational DBMSs, offer simple operators for the deduction of information. For example, the join operator applied to two relational tables where the first administrates the relation between employees and departments and the second the relation between departments and managers, infers a relation between employees and managers. Extending the deductive expressiveness of query languages while remaining computationally tractable is pursued in the research area called deductive databases.

2. Induction is a technique for inferring information that is generalized from the information in the database. For example, from the employee-department and the department-manager tables from the example above, it might be inferred that each employee has a manager.

The search for this higher-level information or knowledge is the goal of the knowledge discovery in databases, KDD, process.

Definition 1 *Knowledge discovery in databases (abbreviated, KDD) is the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data[35].*

In the KDD process we are typically looking for patterns P in a language L that we have specified for the particular domain of interest. For example, the language could be association rules or modal logic expressions, as we will see later. An important notion, called interestingness, is usually taken as an overall measure of pattern value, combining validity, novelty, usefulness, and simplicity. Some KDD systems have an explicit interestingness function I that maps expressions in L to a measure space M_l. Other systems define interestingness indirectly via an ordering of the discovered patterns.

Given the notions listed above, we may state our definition of knowledge as viewed from the narrow perspective of KDD[35]..

- **Knowledge:** A pattern P in L is called *knowledge* if for some user specified threshold $T \in M_l$, $I(P) > T$.

- **Data mining** is a step in the KDD process consisting of particular data mining algorithms that, under some acceptable computational efficiency limitations, produces a particular enumeration of patterns over the database.

  Note that the space of patterns is often infinite, and the enumeration of patterns involves some form of search in this space. the computational efficiency constraints place severe limits on the subspace that can be explored by the algorithm.

- **KDD Process** is the process of using data mining methods (algorithms) to extract (identify) what is deemed knowledge according to the specifications of measures and thresholds, using the database D along with any required preprocessing, sub-sampling, and transformations of D.

  There are many characterizations of the KDD process in the literature [2, 37, 48, 52, 57, 58]. They typically differ in organization rather than substance. We will consider one such characterization. The KDD process proceeds in the following steps[50]:

  1. Data Extraction
  2. Data Curation
  3. Data Engineering
  4. Algorithm Engineering
  5. Mining Algorithm
  6. Data Analysis and Evaluation

Data extraction: Once a problem has been defined, relevant data must be collected. In most cases, the relevant data is extracted from an existing database or a data warehouse. Usually, data mining algorithms cannot be run directly against a database and stored in some format (typically a flat file) that can be loaded into main memory and accessed by the mining algorithms. In these cases, the manual extraction step is really just an artifact of the inability of the data mining algorithm to run directly against a database.

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 KDD process is usually iterative, as 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, 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 first step.

Work has been done in using SQL, the standard database query language, in data mining to directly and automatically get the answers to those questions about the data that it needs to ask[50]. The KDD process can be sped up significantly by relying on the relational database management system's query engine.

Data Cleaning and Exploration: Once the requisite data has been collected, it is important to spend some time exploring the database, for two reasons. First, the data analyst must develop intimate knowledge of the data - not only knowing the attribute names and what they are intended to mean, but also 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. It is rare that the extracted data is without any problem.

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. 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. Third, some of the information in the mining base might be better 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 to the best customized mining base for our purposes, where customizing addresses all three problems mentioned above.

Algorithm engineering: There are many algorithmic approaches to extracting useful information from data. As these are the basis for this thesis we will defer discussion of the specifics until later sections.

Running the mining algorithm and evaluation of results: The next step, running the data mining algorithm, is the point where the algorithm designer can just sit back and watch. The one issue that must be addressed at this stage is the decision of how to partition the data into a training set and a validation set in order to facilitate the evaluation of the quality of the algorithm. We will illustrate these

Home	Team	Stats		Road	Team	Stats		
PPG	RPG	APG	SPG	PPG	RPG	APG	SPG	WINNER
98.2	10.3	10.1	2.7	104.1	8.8	7.4	4.1	HW
101.5	10.4	11.1	2.2	89.9	8.9	12.2	3.1	HW
91.4	13.3	9.3	3.2	101.0	10.8	9.0	3.6	RW
93.2	12.1	9.7	3.5	99.3	12.2	8.8	5.1	HW
96.5	8.4	8.4	4.1	97.6	9.0	6.8	4.2	HW
97.8	9.9	10.2	3.2	94.5	11.7	10.0	2.8	RW
105.4	8.1	14.5	1.8	98.6	10.4	11.7	4.0	HW
91.3	13.4	8.8	4.6	96.6	8.8	7.9	3.1	RW

Table 2.1: These are sample data of NBA games. PPG = points per game, RBG = rebounds per game, APG = assists per game, and SPG = steals per game. These statistics are shown for both the home and road teams. The final field is the winner of the game.

steps with an example.

Example:

Suppose we have been hired by a Las Vegas casino to build a system that predicts the winner of professional basketball games. We are not interested in predicting scores, simply winners. We will pose this as a classification problem. Given all relevant data we would like to classify each "game record" into one of the following tow classes: "Home Team Wins" (HW) or "Road Team Wins" (RW). Suppose further that we have the results of an entire basketball season to use for training. We will then test our system by predicting the outcome of the nth game given the results of first $n-1$ games. A sample of what this data might look like is in Table 2.1.

The data in Table 2.1 is the result of completing the first step in the KDD process, that of data extraction. It is possible that the data was not in its present form in the database. Perhaps, a different table was kept for each team, in which case the statistics of the two teams competing in each game had to be merged to form Table 2.1. Or perhaps, a different table was kept for each statistic. In any case, after deciding what data is necessary for our problem (usually done in consultation with a domain expert) the requisite data must typically be put into a single table in order for the data mining algorithm to be able to access it.

The next step in the data mining process is cleaning and exploration. We want to become familiar with the data as well as to make sure that it is consistent and error free. For example, one problem that may exist in Table 2.1 is that the cumulative data may include the current game. That is we may be using training data that includes the nth game for purposes of predicting the outcome of the nth game. This would, of course, invalidate this data for predictive purposes. We would have to go back to the database and extract the data for the previous game(s) played by each team, which would then include all the valid knowledge available to us. This type of temporal infidelity is not uncommon.

The next step is data engineering. We have actually already performed some of the tasks typically associated with this stage of the process in that we only extracted data that we thought would be important for the prediction task at hand. Often some pruning of the extracted data would occur at this point. In addition we may decide that some of the statistics would be better expressed in a different way. For example, we may decide that rather than looking at home team PPG and road team PPG that we would rather use the statistic point differential, i.e. (Home Team PPG - Road Team PPG). In this case we would add an additional field to our table.

There are many algorithms available for the classification problem. At this point one must be chosen and run on our data set. Following this, we would analyze the results in consultation with a basketball expert who may find them useful or may think we should try again, either with a different algorithm or a different set of attributes. The process is inherently iterative.

2.1 Types of Data Mining Patterns

The name "data mining" has been a godsend. Before it became popular 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.

Data mining unites all of these disciplines under the premise that there exists much valuable knowledge in databases, but that due to the tremendous and growing volumes of data involved, advanced computers are necessary for the meaningful analysis of this data as opposed to tedious manual searches 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 collecting and cleaning 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 to has the need for more tools to efficiently analyze the data. The following is a brief summary of a variety of algorithms that are either particularly prevalent in the data mining community or particularly relevant to the problems we have addressed.

Classification: The ability to predict the future would certainly be a highly desirable skill. While no statistical technique can be used to eliminate or explain all of the uncertainty in the world, statistics can be used to quantify that uncertainty. Many techniques have been developed to accomplish the singular task of predicting the value of a dependant variable from knowledge of the values of one or more independent variables. These techniques include regression (both parametric and non-parametric), neural networks, decision trees, Bayesian networks, and others. We utilize one such technique, Multivariate Adaptive Regression Splines (MARS) in classifying normal versus abnormal network activity and normal versus abnormal gene expression levels.

Link Analysis: Link analysis determines relationships between fields in database records. In some settings, such as analyzing purchasing habits of supermarket customers, link analysis represents the end goal. In other settings, link analysis is simply important diagnostic information. For example, in regression, one common source of error is correlation amongst the "independent" (predictor) variables. Approaches, such as ridge regression have been developed to address this problem. Sometimes the correlations are more subtle and exist only amongst the variances of the predictor variables. Again techniques such as ANOVA decomposition, have been developed to address this problem. In either case, a prerequisite to confronting these problems is recognizing them, and link analysis is an important tool in doing this. In the context of evaluating audit records for intrusion detection, correlation between system features is common. For example one observes frequent correlation between command and argument in the shell command history of a user. A programmer, for example, might have the opening of an emacs session highly correlated with the opening of ".cc" or ".hh" files.

Sequence analysis: Orthogonal to link analysis, sequence analysis provides information between database records rather than within them. Sequence analysis algorithms provide information about temporal relationships that exist in the data. Most of these algorithms involve sliding a "window" across the data records and looking for patterns within the current viewing window[60]. Patterns' interestingness are typically measured by the number of windows in which they occur. In our work we have extended this work by using a propositional, linear, modal logic as a description language for discovered patterns. We have also removed the notion of a sliding window, allowing us to identify patterns between database records that are temporally spread out. This is important in the context of network intrusion detection because it is possible for an intruder to separate the stages of an attack over a significant time period in an attempt to avoid detection. Furthermore, we have changed our interestingess measure from simple frequency to "unexpectedness", allowing us to take advantage of probabilistic information about the data. These issues will be discussed in detail in chapter 3.

Association rules: Intimately related to link analysis is the problem of mining a large collection of basket data for association rules between sets of items with some minimum confidence and support. By an association rule, we mean an implication of the form $X \to I_j$ where X is a set of some items in I, and I_j is a single item in I that is not present in X. The rule $X \to I_j$ is satisfied in the set of transactions T with the confidence factor $0 \leq c \leq 100$ if, and only if, at least $c\%$ of the transactions in T that satisfy X also satisfy I_j. Furthermore, we are often interested in identifying rules with a minimum support threshold. These constraints concern the number of transactions in T that support a rule. For an association rule $X \to I_j$ the support for the rule is defined as $\mathsf{Pr}[\![X \cup I_j]\!]$[3, 5, 7, 61].

Clustering: Clustering is concerned with the problem of automatic discovery of classes in data. This is in contrast with classification where class descriptions are generated from labeled examples. In some sense, automatic classification aims at discovering the "natural" classes in the data. These classes reflect basic causal mechanisms that make some cases look more like each other than the rest of the cases. The causal mechanisms may be as trivial as sample biases in the data, or could reflect some important and yet unknown relationship between data in the domain.

There are several approaches to clustering. For example, in the Bayesian approach the goal is to find the most probable set of class descriptions given the data and prior expectations[22]. The introduction of priors automatically enforces a tradeoff between the fit to the data and the complexity of the class descriptions. Another approach is maximum likelihood which tries to find the class descriptions that best predict the data. These models have trouble because the best classification is always a set of single case classes, perfectly fitting each case, with a class for each unique case. This extreme "classification" has little predictive power for new cases and illustrates a potential problem inherent in all modeling techniques - that of overfitting the data. Overfitting occurs when models are built based on their ability to accurately characterize training data. It is often tempting to build models that perform very well on training data but are too specific to the biases in that data that cause them to be poor predictors of unseen data. In all classification problems this issue has to be confronted.

The statement a clustering pattern 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[35]. Formally, if $\mathsf{Pr_i}[\![x]\!]$ is the distribution for cluster i, $\mathsf{Pr}[\![x]\!]$ 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 x is:

$$\mathsf{Pr}[\![x]\!] = \sum_i \mathsf{Pr}[\![x|cluster = i]\!]\ \mathsf{Pr}[\![cluster = i]\!] \tag{2.1}$$

Chapter 3

Finding Interesting Patterns in Temporal Data

In the network intrusion detection setting, our data initially consists of an ordered set of TCP/IP audit records, each with a set of features deemed relevant, given some domain knowledge. In our data there are also important relationships between different records. Therefore, our goal in this initial stage of our algorithm is to augment each record with a set of temporal features that are most relevant for classifying each record as being normal or malicious activity. To accomplish this we discover patterns using propositional linear time temporal logic and define an interestingness measure over these patterns. We have chosen to use propositional linear time temporal logic both because of its ability to express the types of patterns we are looking for as well as the fact that we have devised an efficient algorithm for finding such patterns. [1]

[1] Much of this chapter is excerpted from [10], a chapter authored by myself and Alex Tuzhilin. Rather than referencing that chapter throughout the current chapter I am referencing it once,

The difficulty in finding interesting patterns is in knowing where to look. Work has been done in finding frequent episodes (patterns) in temporal data. In that setting one utilizes a "time window" of user defined width, which is slid across the event sequence. The frequency of an episode is defined as the number of windows in which it appears and a pattern is considered interesting if its frequency exceeds some threshold. In some pattern discovery settings frequency is an appropriate measure of interestingness.

In other settings, such as network intrusion detection, this is not necessarily the case. Attacks, almost by definition, are rare events and therefore occur with low frequency. Furthermore, the component activities of a multistage attack are typically benign, in and of themselves, and are only malicious in concert. Therefore, when searching for patterns that characterize malicious activity we are not simply interested in those patterns that occur frequently in the presence of an attack, but rather those patterns that occur *more* frequently during an attack than they do during normal network activity.

In this spirit we define an the interestingness of a pattern P to be the ratio of the number of occurrences of P during the course of an intrusion to the number of occurrences of P during the course of normal network behavior. This interestingness measure, unlike frequency, will allow us, in principle, to 1) identify patterns, that are rare, yet highly correlated with intrusive behavior, and 2) ignore patterns that occur frequently during an intrusion, but occur just as frequently during normal behavior. Consider, for example, an attack on the HTTP server on a victim machine. The attacker sends a very large number of requests to the

here.

server in an attempt to cause a buffer overflow. The patterns $Service = HTTP$ and $Port = 80$ are both atomic patterns that would appear frequently during normal activity as well as during this type of attack. Additionally, the pattern $P = \{(Service = HTTP)\ AND\ (Port = 80)\}$ would occur freqeuntly during normal activity (since HTTP requests are typically assocociated with port 80) and during this attack. A long sequence of occurrences of pattern P, however, would be uniquely associated with this attack, and it is therefore, the pattern $P \mathsf{N} P \ldots \mathsf{N} P$ (a modal formula describing several successive occurrences of the pattern P) that we would be interested in identifying. [2]

When interestingness is measured in this probabilistic way, the problem of finding interesting patterns becomes NP-complete. In the remainder of this chapter, after offering some contextual background in the field of knowledge discovery in temporal databases we will present the algorithm for discovering frequent temporal patterns and then present our algorithm for discovering interesting temporal patterns. While given the complexity of the problem, our algorithm cannot necessarily find *all* interesting patterns, we have found empirically that it performs quite well and at the end of this chapter we will present some of these results. Throughout this chapter the problem of finding interesting patterns will be approached in a generic setting with its precise application to intrusion detection disucssed in subsequent chapters.

There has been much work done recently on pattern discovery in temporal and sequential databases. Some examples of this work are [12, 13, 24, 53, 59, 60, 62, 71, 74, 78]. Since there are many different types of discovery problems that were

[2]the operator N will be made clear shortly.

addressed in these references, it is important to characterize these problems using some framework. One such characterization was proposed in [24]. In this chapter we review this framework and then focus on one specific problem of discovering *unexpected* patterns in temporal sequences. To find unexpected patterns in a sequence of events, we assume that each event in the sequence, and therefore each pattern occurs with some probability and assume certain conditional distributions on the neighboring events. Based on this, we can compute an *expected* number of occurrences of a certain pattern in a sequence. If it turns out that the *actual* number of occurrences of a given pattern significantly differs for the expected number, then this pattern is certainly *unexpected* and, therefore, is interesting [73, 72]. We present an algorithm for finding such patterns. As we will show in subsequent chapters this generic setting can be extended for use in intrusion detection by assuming that individual events are conditionally independent and that the probability of a specific event or pattern occurring is calculated based on the number of occurrences of the given pattern or event during the course of normal network activity.

3.1 Characterization of Knowledge Discovery Tasks in Temporal Databases

Characterization of knowledge discovery tasks in temporal databses, proposed in [24] is represented by the 2-by-2 matrix presented in Table 3.1. The first dimension in this matrix defines the two types of temporal patterns. The first type of a temporal pattern specifies how data changes over time and is defined in terms of

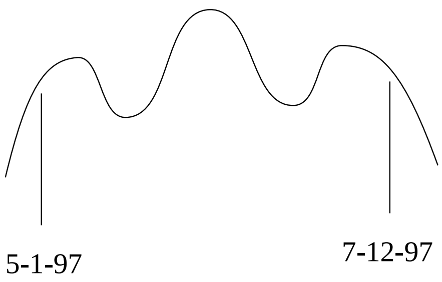


Figure 3.1: An example of the *head_and_shoulder* pattern.

temporal predicates. For example, the pattern

$$head\_and\_shoulder(IBM, 5/1/97, 7/12/97)$$

indicates that the stock of IBM exhibited *head_and_shoulder* trading pattern [56] from 5/1/97 until 7/4/97, as is shown in Figure 3.1). The second type of temporal patterns is *rules*, such as "if a stock exhibits a head-and-shoulder pattern and investor cash levels are low, then bearish period is likely to follow."

The second dimension, the *validation/generation* dimension, refers to the purpose of the discovery task. In *validation* the system focuses on a particular pattern and determines whether it holds in the data. For example, we may want to validate if the *head_and_shoulders* pattern holds for the IBM stock in a given data set or that a certain rule "holds" on the data. The second purpose of discovery can be the *generation* of new predicates or rules that are previously unknown to the system. For example, the system may attempt to discover new types of trading rules in financial applications.

Categorizing patterns in terms of the above two dimensions leads to a two-by-two classification framework of the knowledge discovery tasks, as presented in

Table 3.1. We will describe each of the four categories in turn now.

	Validation	**Generation**
Predicates	I	III
Rules	II	IV

Table 3.1: Types of Knowledge Discovery Tasks.

Class I. The discovery tasks of this type involve the validation of previously defined predicates over the underlying database. For example, assume that we have the temporal database of daily closing prices of stocks at some stock exchange, STOCK(SYMBOL,PRICE,DATE), where SYMBOL is the symbol of a security, PRICE is the closing price of that stock on the date DATE. Consider the following predicate specifying that the price of a certain stock bottomed out and is on the rise again over some time interval:

$$\begin{aligned} bottom\_reversal(x, t_1, t_2) &= (\exists t)(t_1 < t < t_2 \wedge decrease(x, t_1, t) \\ &\wedge \; increase(x, t, t_2)) \end{aligned}$$

where $increase(x, t_1, t_2)$ and $decrease(x, t_1, t_2)$ are predicates specifying that the price of security x respectively "increases" and "decreases" over the time interval (t_1, t_2)[3].

[3] Note that we do not necessarily assume monotonic increases and decreases. Predicates *increase* and *decrease* can be defined in more complex ways, and we purposely leave it unspecified how to do this.

Then we may want to *validate* that the predicate $bottom\_reversal(x, t_1, t_2)$ holds on the temporal relation STOCK(SYMBOL,PRICE,DATE). This validation can take several forms. For example, we may want to find for the predicate *bottom_reversal* if one of the following holds:

$$bottom\_reversal(IBM, 5/7/93, 8/25/93),$$
$$bottom\_reversal(IBM, t1, t2),$$
$$bottom\_reversal(x, 5/7/93, 8/25/93)$$

The first problem validates that the stock of IBM experienced the "bottom reversal" pattern between 5/7/93 and 8/25/93. The second problem finds all the time periods when IBM's stock had "bottom reversal," and the last problem finds all the stocks that had "bottom reversals" between 5/7/93 and 8/25/93.

One of the main issues in the problems of Class I (predicate validation problem) is to find *approximate* matching patterns. For example, for the IBM stock to exhibit the bottom reversal pattern between 5/7/93 and 8/25/93, it is not necessary for the time series of IBM stock to match predicate *bottom_reversal* exactly. Another example of the approximate matching problem of Class I comes from the speech recognition applications where sounds and words are matched only approximately against the speech signal.

There has been extensive work done on Class I problems in signal processing [64], speech recognition [9, 67], and data mining communities. In the data mining community these types of problems are often referred as similarity searches and have been studied in [1, 4, 6, 12, 34, 42].

Class II. Discovery tasks of Class II involve validation of previously asserted *rules*. For example, consider the rule: "If a price correction in a stock is seen before the announcement of big news about the company, then insider trading is likely,"

$$Correction(stock, t_1, t_2) \wedge Big\_news(stock, t_3) \wedge Soon\_after(t_3, t_2)$$
$$\rightarrow \; Insider\_trading(stock, t_1, t_2)$$

where *Correction, Big_news, Insider_trading* and *Soon_after* are user-defined predicates (views) defined on relations STOCKS and NEWS.

Evaluation of this rule on the data entails finding instances of variables $stock$, t_1, t_2, t_3 and the "statistical strength" of the rule (e.g. measured in terms of its confidence and support [3]) that make the rule hold on the data (in statistical terms).

As in the case of Class I problems, one of the main issues in rule validation is the problem of approximate matching. The need for approximate matching arises for the following reasons. First of all, rules hold on data only in statistical terms (e.g. having certain levels of confidence and support). Secondly, some of the predicates in the rule can match the data only approximately (as is the case with Class I problems from Table 3.1). Moreover, certain temporal operators are inherently fuzzy. For example, temporal operator $Soon\_after(t_1, t_2)$ is fuzzy and needs to be defined in "fuzzy" terms[4].

Class III. Discovery tasks of Class III involve the discovery of new interesting predicate-based patterns that occur in the database. In order to discover such patterns, the system should know on what it should focus its search because there

[4]Note that it is not appropriate to define this operator in terms of the temporal operator *Next* because of the inherent ambiguity of the term "soon." Although this operator can be defined in many different ways, one natural approach would be through the use of fuzzy logic [79].

are potentially very many new patterns in the database. In other words, the system should know what to look for by letting the user specify what is *interesting.* For example, the pattern *bottom_reversal* may be interesting because it provides trading opportunities for the user.

Although there are many different measures of interestingness for the user, such as frequency, unexpectedness, volatility, and periodicity [24], the most popular measure used in the literature is *frequency* of occurrence of a pattern in the database [59, 60, 62]. In particular, [59, 60] focus on discovering frequent *episodes* in sequences, whereas [62] discovers frequent patterns in temporal databases satisfying certain temporal logic expressions.

In this chapter, we use a different measure of interestingness. Instead of discovering frequent patterns in the data, we attempt to discover *unexpected* patterns. While it is sometimes the case that the discovery of frequent patterns offers useful insight into a problem domain, there are many situations where it does not. Consider, for example, the problem of intrusion detection on a network of workstations. Assume we define our events to be operating system calls made by some process on one of these workstations. We conjecture, then, that patterns of system calls differ for ordinary users as opposed to intruders. Since intrusion is a relatively rare occurrence the patterns we would discover using frequency as our measure of interestingness would simply be usage patterns of ordinary users offering us no information about intrusions. Instead what we propose is to assign exogenous probabilities to events and then attempt to discover patterns whose number of occurrences differs by some proportion what would be expected given these probabilities. In the example of intrusion detection we would assign the probabilities of events to reflect the frequency of events in the presence of no intruders. Then if an intrusion did occur, it would presumably cause some unexpected pattern of system calls which can be an indication of this event.

As will be demonstrated the new measure of interestingness requires discovery techniques that significantly differ from the methods used for the discovery of frequent episodes. The main reason for that is that unexpected patterns are not monotone. These notions will be made more precise in Section 3.2.

Class IV. Discovery tasks of Class IV involve discovery of new rules consisting of interesting relationships among predicates. An example of a temporal pattern of this type is the rule stating that "If a customer buys maternity clothes now, she will also buy baby clothes within the next few months."

Discovery tasks of Class IV constitute challenging problems because, in the most general case, they contain problems of Class III (discovery of new predicates) as subproblems. The general problem of discovering interesting temporal rules using the concept of an *abstract* [27] has been studied in [11]. Discovery of temporal association rules was studied in [8, 74].

In this section, we reviewed a characterization of knowledge discovery tasks, as presented in [24]. In the rest of this chapter, we will focus on one specific Class III problem dealing with discovery of unexpected patterns. In the next section, we will formulate the problem. Following that we will present an algorithm for finding unexpected patterns, and then present experiments evaluating this algorithm on several applications.

3.2 Discovering Unexpected Patterns in Sequences: The Problem Formulation

We start this section with an intuitive presentation of the problem and then provide its more formal treatment.

We want to find unexpected patterns, defined in terms of temporal logic expressions, in sequences of database records. We assume that each event in each record in the sequence occurs with some probability and assume certain conditional distributions on the neighboring events. Based on this, we can compute an *expected* number of occurrences of a certain pattern in a sequence. If it turns out that the *actual* number of occurrences of a given pattern significantly differs from the expected number, then this pattern is certainly *unexpected* and, therefore, is interesting [72, 73].

The assignment of a probability distribution over the events is necessary for the purpose of determining the *expected* number of occurrences of a pattern P. In general, certain problem domains may suggest a more appropriate way to evaluate these expectations than by calculating them as a function of the frequencies of individual events. In the network intrusion detection setting we calculate the expected number of occurrences of P during an attack based on the frequency of P during normal network activity. In other settings, different methods for determining expectations may be appropriate. The important question is that given some method for computing expectations can we efficiently identify *unexpected* patterns.

In this chapter, we first present a naive algorithm that finds all unexpected patterns (such that the ratio of the actual number of occurrences to the expected number of occurrences exceeds a certain threshold). After that, we present an improved version of the algorithm that finds most of the unexpected patterns in a more efficient manner. We also experimentally compare the naive and the more efficient algorithms in terms of their performance.

More formally, let $E = \{\alpha, \beta, \gamma, \dots\}$ be a finite alphabet of events. We use a subset of propositional linear temporal logic to discover temporal patterns over the events. The basic temporal operators of this system are $\alpha \mathsf{B_k} \beta$ (α before$_{\mathsf{k}}$ β) which intuitively means that α occurs followed by an occurrence of α within k subsequent events, $\alpha \mathsf{N} \beta$ (α next β) α occurs and the next event is β, $\alpha \wedge \beta$ (α and β) which means that α and β occur in the same database record[5], and $\alpha \mathsf{U} \beta$ (α until β) which means before β occurs a sequence of α's occurs.This is often called the *strong until* [77]. While the before operator is actually redundant as $\alpha \mathsf{B} \beta$ can be expressed as $\neg(\neg\alpha \mathsf{U} \beta)$ we have chosen to include it separately for simplicity and efficiency. A pattern of events is defined as a logical expression consisting of ground events connected by these operators. For example, the simplest case is $\alpha \mathsf{N} \beta$. Some additional examples are $\delta \mathsf{U} \alpha \mathsf{N} \beta \mathsf{B} \gamma$ and $\alpha \mathsf{N} \beta \mathsf{N} \gamma$.

In the pattern discovery algorithm presented in Section ??? we consider the following fragment of the Propositional Temporal Logic (PLTL). The syntax of this subset is as follows. The set of formulae of our subset is the least set of formulae generated by the following rules:

[5]Since $\wedge$ is a symmetric operator, throughout this discussion we will assume that an arbitrary ordering has been imposed on the events, such that if α preceeds β in this ordering, then $\alpha \wedge \beta$ is a valid pattern while $\beta \wedge \alpha$ is not. This simply allows us to avoid considering duplicate, symmetric patterns.

(1) each atomic proposition P is a formulae;

(2) if p is a formula and q is a formula containing no temporal operators then $p\mathsf{U}q$, $p\mathsf{B_K}q$, $p\mathsf{N}q$, $p\wedge q$, $q\mathsf{U}p$, $q\mathsf{B_K}p$, $q\mathsf{N}p$, $q\wedge p$ are formulae.[6]

We assume an exogenous probability distribution over the events. While these events may be dependent or independent, depending on the problem domain of interest we assume independence of the events unless explicitly stated otherwise. In any case, given an a priori set of event probabilities, we can compute expected values for the number of occurrences of any temporal pattern in our string. For example, the expected number of occurrences of $\mathsf{E}[\![\alpha\mathsf{B}\beta]\!]$, assuming the events α and β are independent, can be computed as follows. Let X_n be the number of occurrences of the pattern $\alpha\mathsf{B}\beta$ up to the n^{th} element of the input string and α_n the number of α's up to the n^{th} element of the input string. Then

$$\begin{aligned}\mathsf{E}[\![X_n]\!] &= \mathsf{Pr}[\![\beta]\!][X_{n-1}+\alpha_{n-1}] + (1-\mathsf{Pr}[\![\beta]\!])(X_{n-1}) \\ &= \mathsf{E}[\![X_{n-1}]\!] + \mathsf{Pr}[\![\beta]\!]\mathsf{E}[\![X_{n-1}]\!] \\ &= \mathsf{E}[\![X_{n-1}]\!] + (n-1)\mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]\end{aligned}$$

[6]We ignore disjunctions because what seems to occur in practice when disjunctions are allowed is that the disjunction of a very interesting pattern, E, with an uninteresting pattern, F, results in an interesting pattern $E \vee F$. This occurs not because $E \vee F$ truly offers any insight into our problem domain but rather because the interestingness of E "drags up" the interestingness measure of $E \vee F$ to the point where it also becomes interesting. We choose instead to simply report E as an interesting pattern. Our decision to omit conjuctions and negation will be made clear shortly.

Therefore,

$$\mathsf{E}[\![X_n]\!] - \mathsf{E}[\![X_{n-1}]\!] = \mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!](n-1)$$

Also, $\mathsf{E}[\![X_2]\!] = \mathsf{Pr}[\![\alpha]\!] * \mathsf{Pr}[\![\beta]\!]$. From this recurrence equation, we compute $\mathsf{E}[\![\alpha \mathsf{B_K} \beta]\!]$ for the input string of length N as

$$\mathsf{E}[\![\alpha \mathsf{B} \beta]\!] = \frac{\mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]N(N-1)}{2}$$

The expected number of occurrences of patterns of other forms can be similarly computed as

$$\mathsf{E}[\![\alpha \mathsf{N} \beta]\!] = \mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!](N-1) \tag{3.1}$$

$$\mathsf{E}[\![\alpha \mathsf{B_K} \beta]\!] = \mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!](K)(N-K) + \frac{\mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!](K)(K-1)}{2}$$

$$\mathsf{E}[\![\alpha \mathsf{U} \beta]\!] = \frac{\mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]}{1-\mathsf{Pr}[\![\alpha]\!]} \sum_{i=2}^{N-1} 1 - \mathsf{Pr}[\![\alpha]\!]^i + \mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]$$

As was stated earlier, we will search for the unexpected temporal patterns in the data, where unexpectedness is defined as follows:

Definition 2 *Let P denote some temporal pattern in string S. Let $\mathsf{A}[\![P]\!]$ be the actual number of occurrences and $\mathsf{E}[\![P]\!]$ the expected number of occurrences of pattern P in S. Given some threshold T, we define a pattern P to be* unexpected *if $\frac{\mathsf{A}[\![P]\!]}{\mathsf{E}[\![P]\!]} > T$. The ratio $\frac{\mathsf{A}[\![P]\!]}{\mathsf{E}[\![P]\!]}$ is called the* Interestingness Measure (IM) *of the pattern P and will be denoted as* IM(P). [7]

[7] Another measure of interestingness is to find patterns P for which $\mathsf{A}[\![P]\!]/\mathsf{E}[\![P]\!] < T$. This

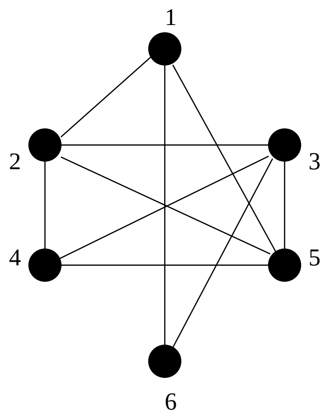


Figure 3.2: The graph $G(V, E)$ with vertices $v_1, v_2, \ldots, v_6$ and a clique C of size 4. $C = \{v_2, v_3, v_4, v_5\}$

This is a probabilistic measure of interestingness whereby a pattern is unexpected if its actual count exceeds its expected count by some proportion T. As the following theorem indicates, however, this problem is likely to be computationally difficult.

Problem (INTERESTINGNESS):

Given a string of temporal events $V = v_1, v_2, \ldots, v_r$, does there exist an interesting pattern in V of the form $X_1 \mathsf{B_k} X_2 \mathsf{B_k} \ldots \mathsf{B_k} X_m$ for an arbitrary m?

Theorem 1 *The INTERESTINGNESS problem is NP-complete.*

Proof We show that our problem is NP-hard by proving that CLIQUE $\leq_p$ IN-

problem can be treated similarly. We have chosen not to search for these patterns because they are complimentary to the ones described in Definition 1. If a pattern $\neg P$ is found to be interesting in our formulation then P will be interesting in this complimentary formulation for some new threshold. Thus in the interest of simplicity we choose to solve these complimentary problems separately and ignore negation.

TERESTINGNESS. The reduction algorithm begins with an instance of CLIQUE. Let $G = (V, E)$ be an arbitrary graph with $|V|$ vertices and $|E|$ edges. We shall construct a string of events S such that an interesting pattern of the form $e_1 \mathsf{B_k} e_2 \ldots \mathsf{B_k} e_m$ exists if and only if G has a clique of size m. The string is constructed as follows. Each vertex $v_1, v_2, \ldots, v_{|V|}$, in the graph G will become an event in our string S, i.e. our events are $e_1, e_2, \ldots, e_{|V|}$. Additionally we will make use of $(|V| + |E|)m$ "dummy" events called $d_1, d_2, \ldots, d_{(|V|+|E|)m}$, where m is the value from the CLIQUE problem. Based on each vertex $v_i \in G$ a substring will be created. The associated event e_i will be called the "generator" of this substring and the substring will be "generated" by the event. The concatenation of these substrings will be the string S. Initially, the vertices in G are arbitrarily ordered $1, 2, ...|V|$. Then for each associated event e_i, in order, we create the substring based on e_i by listing, again in sorted order, the list of vertices(actually their associated events) e_j, for which there exists an edge $(v_i, v_j) \in E$ plus the event $e_i|V|$ times. For example, the substring generated by e_2 for the graph in Figure 1 would be

$$e_1 \underbrace{e_2 e_2 \ldots e_2}_{|V|} e_3 e_4 e_5$$

since there are edges in G from v_2 to each of e_3, e_4, and e_5. Following each substring generated in this fashion we concatenate a substring of all the dummy events in sorted order. As will be seen shortly these dummy events are used to separate the substrings of events e_i and therefore no dummies are needed following the substring generated by $e_{|V|}$. Thus, for the above graph the string

$$S = \underbrace{e_1 e_1 \dots e_1}_{|V|} e_2 e_5 e_6 d_1 \dots d_{(|V|+|E|)m} e_1 \underbrace{e_2 \dots e_2}_{|V|} e_3 e_4 e_5 d_1 \dots$$
$$d_{(|V|+|E|)m} e_2 \underbrace{e_3 \dots e_3}_{|V|} e_4 e_5 e_6 d_1 \dots d_{(|V|+|E|)m} e_2 e_3 \underbrace{e_4 \dots e_4}_{|V|} e_5 d_1 \dots$$
$$d_{(|V|+|E|)m} e_1 e_2 e_3 e_4 \underbrace{e_5 \dots e_5}_{|V|} d_1 \dots d_{(|V|+|E|)m} e_1 e_3 \underbrace{e_6 \dots e_6}_{|V|}$$

The total length of S will be $2|E| + |V|^2 + (|V| - 1)(((|V| + |E|)m)$. This can be seen as follows. The substring generated by e_i will have $|V|$ occurrences of e_i plus one occurrence of each event e_j such that $(v_i, v_j) \in E$ $(deg(v_i))$. Summing over all vertices i the total length of these substrings will equal $2|E| + |V|^2$. In addition there will be a total of $|V| - 1$ occurrences of the substring $d_1 d_2 \dots d_{(|E|+|V|)m}$ with a total length of $(|V| - 1)(((|V| + |E|)m)$. The string S can clearly be constructed in polynomial time as it is polynomial in the size of the graph.

Given that our problem allows for an exogenous assignment of probabilities we will assume that all of the events are equiprobable. That is

$$\mathsf{Pr}[\![X]\!] = \frac{1}{|V| + (|V| + |E|)m}$$

for $X = e_i$ or d_j, $i \in 1 \dots |V|, j \in 1 \dots (|e| + |V|)m$. Since each dummy event occurs exactly $|V| - 1$ times and each event e_i occurs $|V|$ times in the substring it generates plus an additional $deg(|V|)$ times elsewhere, these exogenous probabilities are not consistent with the actual probabilities of the events in S as the events corresponding to vertices occur more frequently than the dummy events. It is possible to define the probabilities so that the assigned probabilities of the dummy events is consistent with their actual frequencies but this requires a somewhat more complicated construction and proof and offers little insight into the problem so we have chosen to proceed as described above.

Let BEFOREK $= |V| + |E|$.

The expected number of occurrences of a pattern

$$\begin{aligned} X_1\mathsf{B_k}X_2\mathsf{B_k}\ldots\mathsf{B_k}X_L &= (n-K(L-1))K^{L-1}\mathsf{Pr}[\![X_1]\!]PX_2\ldots\mathsf{Pr}[\![X_L]\!] \\ &+ \sum_{i=L-1}^{K(L-1)-1}\begin{pmatrix} i \\ (L-1)\end{pmatrix}\mathsf{Pr}[\![X_1]\!]\ldots\mathsf{Pr}[\![X_L]\!], \quad if K > 1 \\ &= (n-K(L-1))K^{L-1}\mathsf{Pr}[\![X_1]\!]PX_2\ldots\mathsf{Pr}[\![X_L]\!], \quad otherwise \end{aligned}$$

where $K =$ BEFOREK and $n = |S|$. This can be derived in a manner analogous to how expectations were derived in section 3. It can be seen that in the special case of $L = 2$ this formula reduces to the one derived previously for $\mathsf{E}[\![\mathsf{B_k}]\!]$.

For the case where $K = |V| + |E|, n = 2|E| + |V|^2 + (|V|-1)((|V|+|E|)m)$, and $L = m$ we will call the value of this expectation ϵ. Let the interestingness threshold

$$T = \frac{2|V|m-1}{2\epsilon}$$

The relevance of this value is that if a pattern of the form $X_1\mathsf{B_k}X_2\ldots\mathsf{B_k}X_m$ is instantiated only with events e_i(no dummies) and it occurs at least $|V|m$ times it will be deemed interesting. If it occurs $|V|m-1$ times it will not. This will be discussed in further detail shortly.

We must now show that this transformation from CLIQUE to INTERESTING-NESS is a reduction. First, suppose a CLIQUE $v_1, v_2, \ldots, v_m$ exists in G and therefore corresponding events $e_1, e_2, \ldots, e_m$ exist in S. Note that here the indexes of the vertices and events are not intended to suggest that the clique must consist of the first m vertices in the original ordering but rather are used for ease of exposition. Of course these $v_1, \ldots, v_m$(and $e_1, \ldots, e_m$) could represent any collection of m vertices(events) although we will continue to assume that they are in sorted order. By construction, the substring generated by e_1 will include

$$\underbrace{e_1 e_1 \ldots e_1}_{|V|} e_2 e_3 \ldots e_m$$

For an arbitrary i the substring generated by e_i will include [8]

$$e_1 e_2 \ldots \underbrace{e_i e_i \ldots e_i}_{|V|} e_{i+1} \ldots e_m$$

Each substring will contain $|V|$ occurrences of the pattern $e_1 \mathsf{B_k} e_2 \mathsf{B_k} e_3 \mathsf{B_k} \ldots \mathsf{B_k} e_m$ and there are m such substrings so the total number of occurrences of this pattern is $|V|m$. Thus

$$\frac{\mathsf{A}[\![e_1 \mathsf{B_k} \ldots \mathsf{B_k} e_m]\!]}{\mathsf{E}[\![e_1 \mathsf{B_k} e_2 \ldots \mathsf{B_k} e_m]\!]} = \frac{|V|m}{\epsilon} > T$$

Conversely, suppose that an interesting pattern of the form $X_1 \mathsf{B_k} X_2 \ldots \mathsf{B_k} X_m$ exists. We must show that a corresponding CLIQUE of at least size m exists in G. The following lemma is the basis for our claim.

Lemma: 1 *If an interesting pattern exists then it consists only of events e_i, containing no dummy events.*

Proof: We have already seen that if a CLIQUE of size m exists in G then an interesting pattern exists in S. Thus interesting patterns are possible. What is left to show is that if

[8] There may, of course be vertices that are not part of the clique that are connected via some edge to e_i. These vertices would also be included.

- a pattern consists only of dummy events then it cannot be interesting, and

- if a pattern consists of both dummy events and events e_i it can't be interesting

Assume we instantiate the pattern $P = X_1\mathsf{B_k}\dots\mathsf{B_k}X_m$ with j dummy events and $m-j$ events e_i where $j = 1\dots m$. Note that given our definition of BEFOREK for any pattern of this form its total length, i.e. the distance in the string S from X_1 to X_m can be at most $(|E|+|V|)m$. Therefore, if a pattern contains any dummy events these occurrences must occur only at the beginning or end of the pattern since any dummy event is part of a substring of $(|E|+|V|)m$ dummy events. That is there cannot exist a dummy event d_j in the pattern such that an event e_i occurs before d_j in the pattern *and* an event e_k occurs after it. We will assume, without loss of generality, that the j dummy events all occur at the end of the pattern. We will next count the maximum number of occurrences of patterns of this form.

Each of the $m-j$ events e_i generates a substring in S. In that substring the event e_i occurs $|V|$ times and all other events occur once. In addition, in the substring of dummy events immediately following this substring each event occurs once. Thus, there can be at most $|V|$ occurrences of the pattern P that include events from the substring generated for each e_i. There are a total of $m-k$ events e_i in the pattern and therefore a maximum of $(m-j)|V|$ occurrences of P that include these substrings. In addition, there exist $|V|-(m-j)$ substrings generated by events not in P. In each of these substrings P can occur at most once since each event in P occurs at most once in a substring that it did not generate. This can result in a maximum of $|V|-(m-j)$ additional instances of P for a total of $(m-j+1)|V|-(m-j)$ occurrences of P. This expression is maximized if $j=1$

in which case the maximum number of occurrences of $P = m|V| - m + 1$. Since

$$\frac{m|V| - m + 1}{\epsilon} < T$$

where ϵ is again the expected number of occurrences of this pattern,this pattern cannot be interesting. $\square$

We now know that any interesting pattern can consist only of events e_i. We also know that each occurrence of an interesting pattern can include only events generated by a single e_i (since BEFOREK $< (|E|+|V|)m$, the length of the dummy substrings separating event substrings generated by each event). Furthermore, we can use an argument identical to the one used in the proof of the above lemma to show that for at least $m|V|$ occurrences of a pattern to exist at least $m|V|$ of them must include the generating event from which all the events in this instance came. In other words, if an interesting pattern $e_1 \mathsf{B_k} \ldots \mathsf{B_k} e_m$ exists then there must be at least $m|V|$ instances which include the e_i that generated the substring from which all the other events came. To see this note that each time an instance of a pattern that includes a generating event occurs, $|V|$ instances will actually occur, one for each copy of the generating event in the substring it generated. Let us assume that only $(m-1)|V|$ instances of a pattern exist that include the generating event from which all other events in this instance came.[9] In all the other substrings generated by events not in the pattern there can be at most one instance of the pattern since each event occurs at most once in a substring it did not generate. There are $|V| - (m-1)$ such events so the total number of instances would only be $m|V| - m + 1$. Therefore, for a pattern to occur at least $m|V|$ times and thus to

[9]There cannot be any more than this unless there are $m|V|$ since they come in multiples of $|V|$.

be interesting there must be $m|V|$ instances that include the generator of the other events in that instance. Since each generator results in $|V|$ instances there are m generators that are part of instances. The m vertices that correspond to these m events form a clique in G. This is clearly true since for any of the e_i amongst these m generators there is an edge from itself to each of the other generators.

Finally, note that this problem is also in NP and therefore NP-complete since given a certificate(i.e. an instantiation of our pattern in this case) we can check if it is interesting by simply scanning over S. This clearly can be done in polynomial time. $\square$

Notice that we have phrased our NP-hardness problem as "Does *any* interesting pattern exist?" We could have just as easily posed the question "Do p interesting patterns exist"? Our proof can be trivially extended to accomplish this by enforcing that the dummy events always contain $p-1$ interesting patterns and that the pth interesting pattern only occur if a clique of size m exists in G. Our decision to enforce that the dummy events contain no interesting patterns and to thus pose our question as we did was rather arbitrary.

While we are trying to find interesting patterns that contain a variety of temporal operators in an arbitrary order, this theorem states that finding interesting patterns that only use the BEFORE operator is hard. Furthermore, we would like to put no restrictions on the "interesting" patterns we discover. We would simply like to find *all* patterns that are interesting. The following theorem, however, shows that it is necessary to impose some bounds on the size of the patterns that we uncover, since in the case of unrestricted patterns, the most unexpected pattern will always be the entire string.

Theorem 2 *Consider a string of temporal events $V = v_1, v_2, \ldots, v_N$ and a temporal pattern T. If the length of T (number of temporal operators in it), length(T) $< N-1$, then there exists another pattern P such that length$(P) =$ length$(T) + 1$ and $IM(P) \geq IM(T)$, where the length of a pattern is defined as the number of events in the pattern.*

Proof:

Let $\mathsf{A}[\![T]\!] = \beta$ and $\dfrac{\mathsf{A}[\![T]\!]}{\mathsf{E}[\![T]\!]} = \alpha$ and $Z = \{z_1, z_2, \ldots, z_m\}$ the set of all events.

We want to prove that $\exists\ z_i \in Z$ s.t. $\dfrac{\mathsf{A}[\![T\mathsf{N}z_i]\!]}{\mathsf{E}[\![T\mathsf{N}z_i]\!]} \geq \alpha$

Assume this is not true for $z_1, z_2, \ldots, z_{m-1}$ and show that it must be true for z_m.

By this assumption and because of (3.1)

$$\frac{\mathsf{A}[\![T\mathsf{N}z_i]\!]}{\mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_i]\!](N-1)} < \alpha \qquad\qquad \forall z_i, i = 1, 2, \ldots, m-1.$$

Therefore, $\mathsf{A}[\![T\mathsf{N}z_i]\!] < \alpha \mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_i]\!](N-1)$.

Then,

$$
\begin{aligned}
\sum_{i=1}^{m-1} \mathsf{A}[\![T \mathsf{N} z_i]\!] &< \sum_{i=1}^{m-1} \alpha \mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_i]\!](N-1) && (3.2)\\
&= \alpha \mathsf{Pr}[\![T]\!](N-1)\sum_{i=1}^{m-1} \mathsf{Pr}[\![z_i]\!] && (3.3)\\
&= \alpha \mathsf{Pr}[\![T]\!](N-1)(1-\mathsf{Pr}[\![z_m]\!]) && (3.4)\\
\text{Since, } \sum_{i=1}^{m} \mathsf{A}[\![T \mathsf{N} z_i]\!] &= \mathsf{A}[\![T]\!] = \beta, && (3.5)\\
\mathsf{A}[\![T \mathsf{N} z_m]\!] &> \beta - \alpha \mathsf{Pr}[\![T]\!](N-1)(1-\mathsf{Pr}[\![z_m]\!]) && (3.6)\\
\frac{\mathsf{A}[\![T \mathsf{N} z_m]\!]}{\mathsf{E}[\![T \mathsf{N} z_m]\!]} &> \frac{\beta - \alpha \mathsf{Pr}[\![T]\!](N-1)(1-\mathsf{Pr}[\![z_m]\!])}{\mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_m]\!](N-1)} && (3.7)\\
= \frac{\beta}{\mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_m]\!](N-1)} &- \frac{\alpha \mathsf{Pr}[\![T]\!](N-1)(1-\mathsf{Pr}[\![z_m]\!])}{\mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_m]\!](N-1)} && (3.8)\\
&= \frac{\beta}{\mathsf{Pr}[\![T]\!]\mathsf{Pr}[\![z_m]\!](N-1)} - \frac{\alpha(1-\mathsf{Pr}[\![z_m]\!])}{\mathsf{Pr}[\![z_m]\!]} && (3.9)\\
(\text{since } \frac{\beta}{\mathsf{E}[\![T]\!]} &= \frac{\beta}{\mathsf{Pr}[\![T]\!](N)} = \alpha) && (3.10)\\
&= \frac{\alpha N}{\mathsf{Pr}[\![z_m]\!](N-1)} - \frac{\alpha(1-\mathsf{Pr}[\![z_m]\!])}{\mathsf{Pr}[\![z_m]\!]} && (3.11)\\
&= \frac{(\alpha N) - (\alpha N - \alpha)(1-\mathsf{Pr}[\![z_m]\!])}{\mathsf{Pr}[\![z_m]\!](N-1)} && (3.12)\\
&= \frac{\alpha + ((N-1)(\alpha \mathsf{Pr}[\![z_m]\!]))}{\mathsf{Pr}[\![z_m]\!](N-1)} && (3.13)\\
&= \alpha + \frac{\alpha}{\mathsf{Pr}[\![z_m]\!](N-1)} && (3.14)\\
&> \alpha \qquad \square && (3.15)
\end{aligned}
$$

Intuitively, this theorem tells us that given an interesting temporal pattern, there exists a longer pattern that is more interesting. In the limit then, the most interesting pattern will always be the entire string of events, as it is the most unlikely.

In order to cope with this, we restrict the patterns that we look for to be of length less than or equal to some length limit. Of course, still the most interesting pattern we will find will be one whose length is equal to the length limit. Nevertheless, it is often the case that an interesting pattern that is not the most interesting provides valuable insight into a given domain as we will see later in discussing our experiments.

3.3 Algorithm

3.3.1 Naive Algorithm

A naive approach to discovering interesting patterns in an input sequence might proceed as follows. Sequentially scan over the input string discovering new patterns as we go. When a new pattern is discovered a record containing the pattern itself as well as a count of the number of occurrences of the pattern is appended to a list of all discovered patterns. This is repeated until all patterns up to a user-defined maximum length, have been found. More precisely, the algorithm proceeds as follows

Definition 3 *BEFOREK: A user defined constant that determines the maximum number of events that X can precede Y by, for $X \mathsf{B}_{\mathsf{K}} Y$ to hold.*

Input:

- Input String
- Event Probabilities: the exogenously determined probabilities of each atomic event.

- BEFOREK
- The threshold T for interestingness. That is the value that, if exceeded by the interestingness measure of a pattern, deems it interesting.
- Maximum allowable pattern length (MAXL).

Output:

- All discovered patterns P such that $IM(P) > T$.

Algorithm:

```
Scan the input string to determine the interestingness measure
of each event in it, and initialize list L with all these events
WHILE L is not empty DO
   Amongst all the patterns of L, choose the pattern C
     with the largest interestingness measure as the next
     candidate to be expanded.
   Expand C as follows. Scan the input string looking
     for occurrences of C.  When an instance of C is
     discovered, expand it both as a prefix and as a
     suffix. By this we mean, record all occurrences of
     (C op X) and (X op C) where op ranges over the temporal
     operators, and X ranges over all events. Finally,
     compute the interestingness of all these newly
     discovered patterns C'.
   IF Length(C') < MAXL THEN add C' to the list L.
```

```
    Remove C from L.
  END WHILE
  Output interesting patterns.
```

Note that the algorithm we just presented is tantamount to an exhaustive search and is therefore highly inefficient. We propose a more practical algorithm, that, although is not guaranteed to find *all* interesting patterns, offers speed up with minimal loss of accuracy. The idea is to expand on the approach presented in [60] of beginning with small patterns and expanding only those that offer the potential of leading to the discovery interesting, larger patterns.

3.3.2 Discovering Frequent Patterns in Sequences

When the interestingness of a pattern is measured by its frequency an efficient algorithm can be used to discover all frequent patterns. The problem statement is as follows: given a class of operators, an input sequence of events, and a frequency threshold, find all patterns that occur frequently enough.

The algorithm for solving this problem has two alternating phases: building new candidate patterns, and counting the number of occurrences of these candidates. The efficiency of the algorithm is based on two observations:

- Where there are potentially a large number of patterns that have to be evaluated, the search space can be dramatically pruned by building large patterns from smaller ones in a prescribed way. If the pattern $\alpha \mathsf{N} \beta \mathsf{N} \gamma$ is frequent, then the patterns $\alpha \mathsf{N} \beta$ and $\beta \mathsf{N} \gamma$ must also be frequent. In general, for a pattern P to be frequent so must all of its subpatterns. The algorithm for identifying frequent patterns can take advantage of this fact by only considering patterns of size n if its prefix and suffix of size $n-1$ are themselves frequent.

- All complex patterns are simply the result of recursively combining other smaller patterns. For example, in order to efficiently count the number of occurrences of the pattern $\alpha \mathsf{N} \beta \mathsf{B}_{\mathsf{K}} \delta \mathsf{B}_{\mathsf{K}} \gamma$ we simply need to have identified the number of occurrences and location of the two patterns $\alpha \mathsf{N} \beta$ and $\delta \mathsf{B}_{\mathsf{K}} \gamma$ have occurred, and have an efficient method for combining patterns via the B_{K} operator. In general, since all of our operators are binary, when combining two patterns with operator Op to create a larger pattern, in order to determine the number of occurrences of the resulting pattern we need only to have 1) determined the number and locations of Op's two operands and have an efficient method for locating patterns of the form $A\ Op\ B$.

The algorithm proceeds by initially counting the number of occurrences of length 1 patterns (the length of a pattern is simply the number events that occur in it). Following that, a candidate set for the next iteration of discovery is computed by combining pairwise all frequent length 1 patterns via each operator. In general, in the nth iteration, the combination of patterns of length $n-1$ and length 1 are added to the candidate set provided that the length $n-1$ prefix and suffix of the resulting length n pattern have already been deemed frequent in the previous iteration. Then, during the discovery phase, the number and location of occurences of the candidate length n patterns are determined tivially given the locations of their length $n-1$ prefixes and length 1 suffixes. This process proceeds until the candidate set becomes empty. Note, that the memory requirements of this algorithm are minimized because once a pattern is deemed infrequent it can never result in being the subpattern of a larger frequent pattern, and can therefore be discarded. We will see shortly that this property does not hold given our definition of interestingness .That is, a pattern can be unexpected while its component sub-patterns are not. This lack of monotonicity in our interestingness measure is most easily understood with an example.

3.3.3 Main Algorithm

Example: Let the set of events be $E = \{A, B, C\}$. Assume the probability of these events is $\mathsf{Pr}[\![A]\!] = 0.25, \mathsf{Pr}[\![B]\!] = 0.25, \text{and} \mathsf{Pr}[\![C]\!] = 0.50$. Also assume that these events are independent. Let the threshold $T = 2$. In other words, for a pattern to be interesting the value of the actual number of occurrences of the pattern divided by the expected number of occurrences of the pattern must exceed 2.0. Consider the following string of events.

```
ABABABABCCCCCCCCCCCC
```

(the length of this string $N = 20$)

Given our probabilities, $\mathsf{E}[\![A]\!] = 5$ and $\mathsf{E}[\![B]\!] = 5$. Also given the expression for computing expectations for patterns of the form $A\mathsf{N}B$.

$$\begin{aligned} \mathsf{E}[\![A\mathsf{N}B]\!] &= \mathsf{Pr}[\![A]\!]\mathsf{Pr}[\![B]\!](N-1) \\ &= (0.25)(0.25)(19) \\ &= 1.1875 \end{aligned}$$

Since $\mathsf{A}[\![A]\!] = 4$ and $\mathsf{A}[\![B]\!] = 4$, both of the events A and B are not interesting (in fact the actual number occurrences of these events was less than what was expected), but the pattern $A\mathsf{N}B$ which occurred 4 times was interesting with

$$\begin{aligned} IM(A\mathsf{N}B) &= \frac{4}{1.1875} \\ &= 3.37 \qquad \square \end{aligned}$$

This lack of monotonicity in our interestingness measure results in a significantly more complex problem especially in terms of space complexity. In the algorithm for discovering frequent patterns significant pruning of the search space can occur with each iteration. That is, when a newly discovered pattern is found to have occurred fewer times than the frequency threshold, it may be discarded as adding new events to it can*not* result in a frequent pattern. With our measure of interestingness, however, this is not the case. The addition of an event to an uninteresting pattern *can* result in the discovery of an interesting one. This inability to prune discovered patterns leads to an explosion in the amount of space required to find unexpected patterns.

A more efficient algorithm than the naive one for finding unexpected patterns involves sequential scans over the string of events discovering new patterns with each scan. A list is maintained of those patterns discovered so far, and on each subsequent iteration of the algorithm the "best" pattern is selected from this list for expansion to be the seed for the next scan.

The heart of the algorithm is how "best" patterns are chosen. We will explain it formally below (in Definition 4), but would like to give some intuition beforehand. Clearly, we would like to define "best" to mean most likely to produce an interesting pattern during expansion. By Theorem 2, we know that expanding an already interesting pattern must result in the discovery of additional interesting pattern(s). The question remains, however, amongst interesting patterns already discovered which is the best candidate for expansion, and if no interesting patterns remain unexpanded, are there any uninteresting patterns worth expanding?

Initially, the algorithm begins with a scan of the input string counting the number of occurrences (and therefore, the frequencies) of individual events. Subsequent to this, we continue to expand *best* candidates until there are no more candidates worthy of expansion. This notion will be made clear shortly.

Definition 4 *The FORM(P) of a pattern P is a logical expression with all ground terms in P replaced by variables.*

For example, if $P = \alpha \mathsf{N} \beta \mathsf{B_K} \gamma \mathsf{B_K} \delta$ then FORM$(P) = W \mathsf{N} X \mathsf{B_K} Y \mathsf{B_K} Z$.

Given the length of the input string, we can determine the number of patterns of each form in the input string. For example, given a string of length M, the number of patterns of form $X\mathsf{N}Y$ is $M-1$. The number of patterns $X\mathsf{B_K}Y$ is $(M-K)K + ((K)(K-1)/(2))$.

Definition 5 *Given a pattern P and an operator op,* Actual_Remaining(P op X) *is the number of patterns of the form $PopX$ that have yet to be expanded. This value is maintained for each operator, op and pattern P. That is, we maintain a value for $P\mathsf{N}X, P\mathsf{B_K}X, X\mathsf{B_K}P, etc\ldots$ Again, X ranges over all events.*

For example, if there are 20 occurrences of $P = \alpha\mathsf{B_K}\beta$ in the input string and 5 patterns of the form $\alpha\mathsf{B_K}\beta\mathsf{N}X$ have been discovered so far, then *Actual_Remaining_Next* $\alpha\mathsf{B_K}\beta\mathsf{N}X = 15$.

We use the following heuristic to determine which discovered pattern is the best one to expand. Given an arbitrary literal D, the *best* pattern P for expansion is the pattern for which the the value of

$$\mathsf{E}[\![\mathsf{A}[\![P \; op \; \delta]\!]/\mathsf{E}[\![P \; op \; \delta]\!]]\!]$$

is maximal for some δ.

This heuristic is simply a probabilistic statement that the pattern P that is most likely to result in the discovery of an interesting pattern is the one for which there exists a literal δ such that the expected value of the interestingness measure of the pattern generated when δ is added to P via one of the temporal operators *op*, is maximal over all discovered patterns P, literals δ, and operators *op*. It is necessary for us to use the *expected* value of the interestingness measure because, although we know the actual number of occurrences of both P and δ, we do not know the number of occurrences of $P \; op \; \delta$. How this expectation is computed follows directly from our devivations of expectations and is illustrated in the following example.

Example: If $P = \alpha \mathsf{N} \beta$ and *op* is *next*, then

$$\begin{aligned} & \mathsf{E}[\![\mathsf{A}[\![P\mathsf{N}\delta]\!]/\mathsf{E}[\![P\mathsf{N}\delta]\!]]\!] \\ = \; & (\#P_{\mathsf{N}})(\mathrm{FR}(\delta))/\mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]\mathsf{Pr}[\![\delta]\!](K-2) \end{aligned}$$

where,

K = length of input string

$\mathrm{FR}(\delta)$ = frequency of δ's that could complete the pattern $\alpha \mathsf{N} \beta \mathsf{N} X$

$\#P_{\mathsf{N}}$ = number of occurrences of pattern P yet to be expanded via the operator N

If *op* is *before*,

$$\begin{aligned}
& \mathsf{E}[\![\mathsf{A}[\![P \mathsf{B}_{\mathsf{K}} \delta]\!] / \mathsf{E}[\![P \mathsf{B}_{\mathsf{K}} \delta]\!]]\!] \\
= \; & ((\#P)(\mathrm{FR}(\delta))(BEFOREK)) / \mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]\mathsf{Pr}[\![\delta]\!](K-2)(BEFOREK) \\
= \; & ((\#P)(\mathrm{FR}(\delta))) / \mathsf{Pr}[\![\alpha]\!]\mathsf{Pr}[\![\beta]\!]\mathsf{Pr}[\![\delta]\!](K-2)
\end{aligned}$$

Similar arguments are used for any combination of the operators *before, next,and*, and *until* [10].

In general, we choose the candidate pattern, P, the suffix literal δ and the operator *op* whose combination is most likely to result in the discovery of an interesting pattern.

Throughout our algorithm, two data structures are necessary in order to efficiently compute best candidates on each subsequent iteration.

- An $((N+1) \times M)$ matrix where N is the number of distinct events, and M is the number of different pattern forms that we intend to discover. In principle, M can be very large. However, in practice we have limited the length of our patterns to approximately 5 (depending on the application), noting

[10] For *before* and *until* these definitions are slightly erroneous due to losses of patterns at the ends of the input string. These errors are negligible, however, since the length of the input string is much larger than the length of individual patterns of interest

that the infrequency of much larger patterns typically lends them statistically insignificant. With the maximum pattern length set to 5 and using our four temporal operators N, $\mathsf{B_k}$,U, and $\wedge$, the value of $m = \sum_{i=1}^{5} 4^i = 4(4^5 - 1)/(4 - 1) = 1364$, a managable number.

The structure of this matrix is as follows: each entry $[i, j]$ $i \in 1 \ldots N$, $j \in 1 \ldots M$ represents the remaining number of yet-to-be-discovered patterns of form j whose final event is i. This number is easily maintained as it is simply the total number of occurrences of the event i minus the number of already discovered patterns of form j whose final event is i. The additional $(N+1)$st row contains the total number of already discovered patterns (the sum of the values in the columns) of form j. Each column of this array is sorted such that literal α preceeds β in column j if the number of α's remaining to be added as suffixes to create patterns of form j divided by $\mathsf{Pr}[\![\alpha]\!]$ exceeds that value for β. This value will be called the *"candidacy value"* of the corresponding literal for the corresponding pattern form.[11]This matrix wil be termed the *"suffix matrix"*.

- The second data structure is an array of $M \times R$ lists where M is again number of different pattern forms that we intend to discover and R is the number of temporal operators we are using. In list j_{op}, all patterns of form j that have already been discovered are kept in sorted order by the number of occurrences

[11]Of course, sorting each column, results in row i not always referring to a single literal, but rather the ith literal in sorted order for that particular column. An indexing scheme is used to keep track of which entry represents which literal in each column. The scheme itself is straightforward and omitted for simplicity.

of each pattern yet to be expanded through the use of operator *op* divided by $\mathsf{E}[\![P]\!]$. This value will be called the corresponding pattern's *"candidacy value"* for the corresponding operator. This value is trivial to calculate since we know the total number of patterns that result of the form P *op* X. Along with each pattern we maintain the number of occurrences of the given pattern P, and the locations of P. This array will be termed the *"set of discovered patterns"*.[12]

The best combination of an element from each of these two data structures will be the candidate for the next discovery iteration. More precisely, at each iteration, assume that the first value in each list in the set of discovered patterns of whose length is less than the maximum allowed pattern length correspond to the patterns $P_1, P_2, \ldots, P_M$. Additionally, assume that the first value in each column in the suffix matrix corresponds to the literals $\alpha_1, \alpha_2, \ldots \alpha_M$. We compute the M values that result from multiplying the candidacy value for each of these P_i times the first value in the suffix matrix for the pattern form that is the result of combining a pattern P_i from the set of discovered patterns with the literal α via the operator *op* corresponding to the operator for the list from which P was taken. We choose the pattern P_i, literal α_j and operator *op* whose combination results in the largest amongst these M values. In doing this we accomplish our goal of choosing the candidate pattern, literal, and operator whose combination is most likely to result in the discovery of an interesting pattern.

[12]This discussion suggests that R copies of the list of locations of the pattern P are maintained, one for each temporal operator. In, fact only one list is kept with a pointer to that list actually stored with each occurrence of the pattern.

Once these candidates have been chosen, determining the number of occurrences of the pattern P_i op α_j can be computed via linear scans of the location lists for the pattern P_i and the literal α_j. For example, if $op = \mathsf{N}$ then we look for locations l such that P_i occurs at location l and α_j occurs at location $l+1$. If the $op = \wedge$ we look for locations where both P_i and α_j occur, etc.

Intuitively, this algorithm begins by choosing the (pattern, literal, operator) triple whose combination is most likely to result in the discovery of an interesting pattern. As the algorithm progresses if a given pattern P has not generated a lot of newly discovered patterns as a candidate for expansion it will percolate towards the top of its associated sorted list. Likewise, if a literal α has not been used as the suffix of a lot of discovered patterns it will percolate to the top of its suffix list. In this way, as patterns and literals become more likely to generate an interesting pattern via combination they will become more likely to be chosen as candidates for the next iteration.

Given these preliminary motivations, we now formally present the algorithm:

Input:

- Input Sequence of database records
- Event Probabilities
- BEFOREK: as discussed earlier we use a bounded version of the before operator. BEFOREK is a user defined variable that is equal to the maximum distance between two events X and Y for $X \mathsf{B}_{\mathsf{K}} Y$ to hold.
- Threshold T for interestingness, that is the value that if exceeded by the

interestingness measure of a pattern deems it interesting

- Value of MIN_TO_EXPAND: the minimum threshold of expected interestingness that a pattern, literal, operator triple must have in order to become the next pattern for expansion. The algorithm will terminate if no such pattern remains.
- Maximum allowable pattern length

Output:

- List of interesting patterns, their number of occurrences and the value of their interestingness measures

Algorithm:

```
Scan the input string to determine the interestingness
and locations of each event
Initialize list with the set of discovered patterns
Initialize the suffix lists
WHILE (Choose_Next_Candidate() >= MIN_TO_EXPAND
    Calculate_Pattern_Locations($P$,$\alpha$,$op$)
    Update_AlreadyDiscoveredPatterns()
    Update_SuffixList()
END WHILE
Return interesting patterns
```

The algorithm continues until there are no more patterns for which (actual remaining/expected remaining) exceeds some minimum threshold MIN_TO_EXPAND, a parameter chosen at the outset.

Scanning for each event: This is a simple linear scan of the DL events that occur in the record sequence where D is the number of database records and L is the number of fields in each record.

Initializing the set of discovered patterns: R lists need to be intialized at this stage where R is the number of temporal operators we are using. Each list represents the pattern form X where X is an arbitrary literal. One sorted list is stored for each temporal operator. The cost of this initialization is simply the cost of sorting these lists. Each list will initially be in identical sorted order. Therefore, the total cost of this initialziation is $O(NlogN)$ where N is the number of distict events in the database. Each literal α, in each list, has an initial candidacy value of $\frac{\mathsf{A}[\![\alpha]\!]}{\mathsf{Pr}[\![\alpha]\!]}$ where $\mathsf{A}[\![\alpha]\!]$ is the number of occurrences of α determined in the intial scan.

Initializing the suffix lists: R lists need to be intialized at this stage where R is the number of temporal operators we are using. Each list contains the potential suffixes for all length 2 patterns. Each of these lists will again be sorted based on their candidacy values. Initially, these values are the same as for the set of discovered patterns and, therefore, no additional sorting is necessary. The total cost of this initialization is $O(N)$.

Choose_Next_Candidate: In this function we compute the M values that result from multiplying the candidacy value for each of the patterns P_i that are at the front of the discovered pattern lists times times the first value in the suffix matrix for the pattern form that is the result of combining a pattern P_i from the set of discovered patterns with the literal α via the operator *op* corresponding to the operator for the list from which P was taken. We choose the pattern P_i, literal α_j and operator *op* whose combination results in the largest amongst these M values. The cost of this operation is $O(M)$

Calculate_Pattern_Locations: As described earlier, we can compute the locations of the pattern resulting from combining a pattern P with a literal α via the operator *op* via a linear scan of the location lists for P and α. The total number of operations required for this computation is proportional to the longer of these two location lists. This has an expected value of $\frac{DR}{N}$.

Update_Already_Discovered_Patterns: Given that we have just computed the locations of our candidate P *op* α, this update requires two steps. First, the newly discovered patterns must be inserted into the appropriate R lists. Since we need to maintain the sorted order of these lists each insertion will require $O(log(L))$ where L is the length of the list into which P is being inserted.

The second step is to update the list that P was chosen from. The number of occurrences of P yet to be expanded via the operator *op* has just been decreased by the number of occurrences of the pattern P *op* α. This will reduce its candidacy value and P, therefore, needs to be restored to its appropriate sorted position. This operation will require $O(L)$ operations where L is the length of the list that P was taken from.

Update_Suffix_List: The list corresponding to the form of pattern P *op* α now needs to be updated. The total number of patterns of this form already discovered needs to be increased by the number of occurrences of P *op* α. Additionally, the number of α's yet to be used as a suffix for a pattern of this form needs to be decreased by this same value. Finally, since the candidacy value of α will have now decreased it now needs to be put in its appropriate sorted order. This will require $O(NlogN)$ where N is the number of distinct events in our database.

A couple of observations are relevant at this stage regarding this algorithm. First, while we can evaluate the complexity (as we have done above) of each iteration of the algorithm we do not know a priori the number of patterns that will be discovered before there are no more (pattern, literal, operator) triples whose expected interestingness exceeds MIN_TO_EXPAND. We are, therefore, unable to precisely calculate the computational complexity of this entire algorithm. Second, in this algorithm we discover only a single new pattern with each iteration. We have experimented with expanding more patterns in each iteration. For example, we could simply choose a pattern P literal α, pair and include in our candidate set all patterns of the form P *op* α where *op* ranges over all temporal operators. Going even further we could only choose a pattern P and include all patterns of the form P *op* X where *op* ranges all operators and X over all literals. We have found in practice that if the number of different events in our data set is large then these techniques are too coarse. More specifically, too many patterns are discovered in each iteration and many interesting patterns go undiscovered before we begin to run into memory limitations. In the final analysis, the number of patterns worth including in each iteration results from an evaluation of the time/space usage tradeoff that results from this decision.

3.4 Experiments

In our initial evaluation of this algorithn we conducted experiments on three different problem domains. The first was a simple sequence of independent events. This data was generated synthetically. The second domain we considered were sequences of UNIX operating system calls as part of the `sendmail` program. The third was that of Web logfiles. In the last case, events were correlated.

3.4.1 Sequential independent events

We used an input string of length 1000 over 26 different events. In this case, we assumed that each event was equally likely and that the events were independent. We searched for patterns, P, for which Length(P)$\leq$ 5. Our results are presented in Table 3.2. The columns of the above table are as follows:

Algorithm - The algorithm used. The naive algorithm, presented in Section 4.1, represents essentially an exhaustive search over the input string and is guaranteed to find all interesting patterns. It is included as a benchmark by which we measure the effectiveness of the main algorithm. Percentage is equal to the value for the main algorithm divided by the value for the naive algorithm times 100 for each column respectively. The first number following each algorithm(2 or4) is the value of BEFOREK used. The second number(3,4, or 6) is the interestingness threshold.

# of Scans - The number of scans over the input sequence necessary to discover all interesting patterns found.

# of Expanded Patterns - The number of patterns discovered, interesting or oth-

Algorithm	# of Scans	# of Expanded Patterns	# of Interesting Patterns
Naive(2,3)	416	2489	290
Main(2,3)	161	919	268
Percentage	38.7%	36.9%	92.4%
Naive(4,3)	416	3105	259
Main(4,3)	163	1073	250
Percentage	39.2%	34.6%	96.5%
Naive(2,4)	416	2489	168
Main(2,4)	161	919	164
Percentage	38.7%	36.9%	97.6%
Naive(4,4)	416	3105	171
Main(4,4)	163	1073	166
Percentage	39.2%	34.6%	97.1%
Naive(2,6)	416	2489	133
Main(2,6)	161	919	130
Percentage	38.7%	36.9%	97.7%
Naive(4,6)	416	3105	129
Main(4,6)	163	1073	127
Percentage	39.2%	34.6%	98.4%

Table 3.2: Results for independent sequential data.

erwise.

# of Interesting Patterns - The number of interesting patterns found.

Based on the results presented in Table 3.2, the main algorithm did not find all interesting patterns, although it discovered most while doing less work than the naive algorithm. Also note that the main algorithm was more accurate as our threshold for interestingness increased. In other words, when our algorithm did miss interesting patterns they tended not to be the most interesting.

3.4.2 Sequences of OS System Calls

The second domain we investigated was a sequence of operating system calls made by a `sendmail` program. The events consisted of the 31 different system calls that the program made and our string consisted of 31769 sequential calls. At the time of these experiments we had no knowledge of the actual probabilities of these events. Therefore, we made an assumption that system calls are independent from each other and estimated probabilities of individual events by simply scanning the string and counting the number of actual occurrences of each event. For each event e_i we let $\mathsf{Pr}[\![e_i]\!] =$ (number of occurrences of e_i)/ (the total string length). Because of this, the interestingness of each of atomic event was by definition exactly 1. This forced us to assign a value to MIN_TO_EXPAND that exceeds 1 since otherwise the algorithm would not even begin. This resulted in more scans of the input string than were actually necessary to discover interesting patterns but nonetheless the improvement we achieved over the naive algorithm was consistent with our experiments in other domains (approximately three times). The following represent a selection of interesting patterns discovered. These were selected because of a combination of their interestingness as well as our confidence that these actually represent significant events due to the number of occurrences of them. These results were generated on a run where we allowed strings of up to length 5.

```
EVENT :((sigblock NEXT setpgrp) NEXT vtrace)
COUNT :2032
ACT/EXP :43.1628

EVENT :(((sigblock NEXT setpgrp) NEXT vtrace) NEXT vtrace)
COUNT :455
ACT/EXP :83.1628

EVENT :(((sigblock NEXT setpgrp) NEXT vtrace) BEFORE sigvec)
COUNT :355
ACT/EXP :52.1150

EVENT :(sigblock NEXT(setpgrp BEFOREK vtrace))
COUNT :2032
ACT/EXP :21.5814

EVENT :((sigblock BEFOREK setpgrp) NEXT vtrace)
COUNT :2032
ACT/EXP :21.5814
```

```
EVENT :((sigpause NEXT vtrace) NEXT lseek)

COUNT :1016

ACT/EXP :106.672


EVENT :(sigpause BEFOREK (vtrace NEXT lseek))

COUNT :1016

ACT/EXP :53.336


EVENT :(sigvec BEFOREK (sigpause NEXT (vtrace NEXT (lseek NEXT lseek))))

COUNT :29

ACT/EXP :212.349


EVENT :(sigpause BEFOREK (vtrace BEFOREK lseek))

COUNT :2032

ACT/EXP :53.336


EVENT :((vtrace NEXT lseek) NEXT lseek)

COUNT :1017

ACT/EXP :35.5112
```

In these results COUNT represents the number occurrences of the pattern EVENT and ACT/EXP represents the interestingness of this pattern. We make a few observations. First, most of the interesting patterns that occurred a reasonable number of times (the ones shown above) were mostly of length 3. There were, of course, more interesting patterns of longer length but the number of occurrences of these patterns was significantly fewer. Also notice that no interesting UNTIL patterns were discovered. This is because we never saw AAAAAAB, i.e. all the occurrences of until were of the form AB or AAB which were captured by NEXT or BEFORE and since fewer instances of NEXT and BEFORE were expected these proved more interesting.

These system calls are from the UNIX operating system. In the future we propose to assign probabilities of atomic events based on their frequencies in a period when we are confident no intrusions to the network occurred and then see if we can discover interesting patterns that correspond to intrusions.

3.4.3 Web logfiles

Each time a user accesses a Web site, the server on the Web site automatically adds entries to files called *logfiles*. These therefore summarize the activity on the Web site and contain useful information about every Web page accessed at the site. While the exact nature of the information captured depends on the Web server that the site uses, the only information we made use of was the user identity and the sequence of requests for pages made by each user. The Web site we considered was that of one of the schools at a major university. The users we considered were the two most frequent *individual* users. It is important to recognize that the Web logfiles simply tell us the hostname from which a request originated. Typically, there are a large number of users who may access a Web site from the same host, and the hostname, therefore, cannot be used to definitively identify individual users. We attempted to identify, with some confidence, frequent hostnames that did indeed represent individual users. We used two Web logfiles for our experiments. First, we considered a synthetic Web log. This included a Web site with 26 different pages and 236 total links. We used an input string of length 1000 representing 1000 hits on pages of the site. In this case events were hits on Web pages. Probabilities were, of course, not independent. The probability of a user reaching a given Web page is dependent on the page he is currently at. In order to compute a priori probabilities of each page we declared several pages to be equally likely "entrance points", to the Web site. If there were N "entrance points" then each has a $\frac{1}{N}$ chance of occurring. If P is one of these "entrance points", P has K links on it and one of these links is to page G then the probability of G occurring is $(\frac{1}{N})(\frac{1}{K})$. By conducting an exhaustive breadth-first search we were able to calculate the probabilities of each event occurring (i.e. each page being "hit"). When calculating expectations for various patterns, we used conditional probabilities. So, for example, the $\mathsf{E}[\![A\mathsf{N}B]\!]$ is no longer $\mathsf{Pr}[\![A]\!]\mathsf{Pr}[\![B]\!](K-1)$, where K is the length of the input string. It is now $\mathsf{Pr}[\![A]\!]\mathsf{Pr}[\![B|A]\!](K-1) = \mathsf{Pr}[\![A]\!](1/\#\text{of links in page } A)(K-1)$ if there is a link from A to B and 0 otherwise. Our results for this data are presented in Table 7.1. The interestingness threshold for these experiments was 3.0. Once again our algorithm was able to find most interesting patterns while examining a much smaller portion of the search space than the naive algorithm did.

Finally, we considered data from an actual Website from one of the schools of a major university. There were 4459 different pages on this site with 37954 different links between pages. We used Web log data collected over a period of nine months and selected out the two most frequent individual users of the site, both of whom accounted for more that 1400 hits and used these sequences of hits as our input string. Our experiments using this data were less enlightening than when we used

Algorithm	# of Scans	# of Expanded Patterns	# of Interesting Patterns
Naive2	634	1356	464
Main2	239	528	437
Percentage	37.7%	38.9%	94.2%
Naive4	654	1564	462
Main4	245	568	437
Percentage	37.5%	36.3%	94.6%

Table 3.3: Results for synthetic Web logfile data.

synthetic data. The main algorithm found only a handful of interesting patterns of length greater than two. In fact, when we applied the naive algorithm we found that there were few more interesting patterns to be found at all. More specifically, the main algorithm found 2 and 3 interesting patterns of length greater than two in our two input strings, respectively. The naive algorithm found 3 and 3. The primary reason for the lack of interesting patterns of greater length was that the size of the Web site dominated the size of the input string. The fact that there were 4459 pages and our input strings were only of length 1400 made the expected number of occurrences of each event very small - so small, in fact, that even a single occurrence of many events proved interesting.

Additional factors that compounded the problem are:

1. **Changing Web Structure.** Our algorithm was run on a graph that corresponds to the Web architecture at a single instant in time and thus failed to cature the information at a single instant in time

and thus failed to capture the information that encoded the evolution of the Web architecture (we captured the structure of the Web site, including the links, on a single day, and extrapolated it to 9 months of Web log data). Over this period there were some changes to the Web site. These changes create some difficulties in that the Web logfiles showed that users linked from pages to other pages where links had ceased to exist in the Web at the time of consideration. In fact, there were visits to pages in the Web log data that did not exist in the site we were using. This had the effect of forcing the expected number of occurrences of any patterns that included these pages or links to be zero and thus never considered interesting either as patterns or candidates.

2. **Multiple Sessions.** While each input string we used had a length greater than 1400 events, these Web hits spanned many sessions. In fact, the average session length was approximately 10 hits. The last hit from one session immediately preceded the first hit of the NEXT session in our input string. Normally, however, a link did not exist from the last page of the first session to the first page of the NEXT session. Therefore, once again this had the effect of forcing the expected number of occurrences of any patterns that included this sequence of pages to be zero and thus never considered interesting either as patterns or candidates.

3. **Caching.** Consider what sequence of hits appears in Web log data if a user goes to pages A, B, C, D in the following order $A \rightarrow B \rightarrow C \rightarrow$

$B \rightarrow D$. Normally, what occurs is that a request is made(and therefore logged) for page A then page B then page C then, however, when the user goes back to page B no request is made of the server because this page has been cached on the users' local machine. Finally, a request for page D will be made and logged. Therefore, this sequence of hits will appear in the Web log data as follows: $A \rightarrow B \rightarrow C \rightarrow D$. If no link exists from page C to page D then once again the expected number of occurrences of any pattern including this sequence of events will be zero. Given the wide use by Web users of the BACK button, the effect of caching is substantial.

4. **Local Files.** Finally, many pages that appeared in the Web log data did not appear in the Web site we were using because they were files kept on individuals local machines in their own directories, rather than on the Web server. These pages had the same effect as the changes made in the Web over the nine month period.

Chapter 4

Classification

Having selected appropriate predictor variables through the feature selection process described in the last chapter, we are now prepared to use these features to construct a classifier. Our goal is to keep the classifier general enough in order to be comparable to several heuristic models such as *neural networks* (by allowing robust basis functions) and *classifier systems* (by using adaptive, multiphase regression). The problem remains, however, how to best distinguish the signal $f(\mathbf{X})$ from the noise ϵ in equation 4.1. For finite training samples our definition of the true underlying function $f(\mathbf{X})$ is incomplete. Any quantity can be expressed as the sum of two other quantities. To this end, we have selected **MARS** (**M**ultivariate **A**daptive **R**egression **S**plines) developed by Jerome Friedman [41]. In choosing this model we have considered many other classification techniques and settled on this one based on its expressiveness and its performance in the presence of high dimensional data.

In order to motivate our choice of **MARS** we first include a chapter that surveys a variety of approaches to multivariate regression modelling. Our goal is to illustrate some of the difficulties associated with each method when applied in high dimensional settings, in order to motivate our choice of **MARS** as our regression technique. Friedman motivated his original development of **MARS** by contrasting it with some of the same (as well as other) modeling techniques. Recall that the obstacle we face here is that the high dimensional data results in poor coverage of the space being modeled which in turn resulsts in unacceptably high variance in the models constructed. Particular attention will be given to spline regression and decision tree classification as **MARS** can most easily be viewed as a direct extension of these approaches.

The classification problem can be viewed as an attempt to accurately approximate a response variable y that is function of many variables, given only the value of the function, perturbed by noise, at various points in the space of predictor variables as determined by the available training data. The objective is, given one or more predictor variables $x_1, \ldots, x_n$ and training data $\{y_i, x_{1i}, \ldots, x_{ni}\}_1^N$ to derive a rule for estimating response values in future observations given only the values of the predictor variables[41]

The relationship between y and the predictor variables $x_1, \ldots, x_n$ is assumed to take the form

$$y = f(x_1, \ldots, x_n) + \epsilon \tag{4.1}$$

where $(x_1, \ldots, x_n) \in D \subset R^n$ containing the data; the single-valued deterministic function f, of its n-dimensional argument, is intended to capture the predictive relationship of y on $x_1, \ldots, x_n$. The random component ϵ reflects the dependence of y on quantities other than $x_1, \ldots, x_n$. We define $\mathsf{E}[\![\epsilon \mid \mathbf{X}]\!] = 0$ for all $x_1, \ldots, x_n$, so that the assumed true underlying function f can be defined by

$$f(\mathbf{X}) \;=\; \mathsf{E}[\![y \mid \mathbf{X}]\!] \tag{4.2}$$

with the expected values taken over the population from which the training and future data are presumed to be random samples. This is somewhat of an oversimplification of the problem that we are addressing. Recall that the domains that we are interested in, the data is nonstationary. That is, it may be unreasonable to assume that the training data and the out of sample data will be drawn from identical joint probablity distributions. We ignore this issue for the time being and address it later in chapter 6 via the use of Stein shrinkage.

In this setting, the goal of the training procedure is to build a model $\hat{f}(x_1, \ldots, x_n) + \epsilon$, using available training data, that reasonably approximates $f(x_1, \ldots, x_n) + \epsilon$ over the domain D of interest.

4.1 Classification Techniques

There are five classification techniques that we consider here as motivation for our choice of **MARS** as the classification method used in our intrusion detection system. The discussion of each of these approaches begins with a description of the given technique including some implementation considerations in cases where decisions made about a specific implementation can significantly effect either the form of the model constructed or the algorithmic complexity of the model construction process.

The first, and simplest approach we consider is *global parametric modeling.* In this approach the underlying function that we are attempting to estimate is assumed to take some simple parametric form. The problem of model construction then reduces to a simple parameter estimation problem that is typically solved by minimizing the sum of square errors of the model on the training data. While this approach is very simple to implement and is quite effective if we have prior knowledge of the form of the estimated function, this is not typically the case. In the usual setting where we have no prior knowledge of the form of the estimated function, a poor choice of function for use in this technique can result in very low accuracy in the resulting model.

The second approach we consider is *spline approximation*. Spline approximation has particular interest for us since **MARS** may be viewed as a direct extension of this approach. Again, a specific parametric form of the underlying function is assumed. In spline approximation, however, we additionally divide the domain of interest into a set of nonoverlapping subregions defined by *knot points*. The location of these knot points is determined a priori and then the parameters of the chosen paramteric function are estimated seperately in each subregion. Continuity conditions are imposed in order to ensure smoothness at the region boundaries. While spline approximation offers an improved accuarcy as compared with global parametric modeling, this approach has limitations especially when dealing with high dimensional data. On the one hand, unlike global parametric modeling, this technique allows us to model local variations in the underlying function. In a high dimensional setting, however, the statistician is caught between two currents. She would like to increase the number of knot points (and therefore the number of subregions) in order to improve the models ability to reflect local variation in the underlying function. As the number of subregions increases the number of data points within each subregion decreases, resulting in a high variance in the resulting model. Additionally, spline approximation requires an a priori selection of knot point locations. The appropriate choice for these locations is typically unknown and varying these locations can have significant impact on the resulting model. As we will see in Chapter 5 the **MARS** algorithm addresses this problem by adaptively choosing knot locations based on the data.

Next we consider *kernel and nearest neighbor estimates.* This is the last of the approaches we consider where a parametric function form is assumed. Like global parametric modeling this problme reduces to one of parameter estimation. In both of these approaches, however, the parameters are functions of the independent variable, $\mathbf{x}$ and are therefore generally different at each evaluation point $\mathbf{x}$. The parameters are typically estimated by locally weighted least squares so that the dominant effect in the paratmeter estimation is given by data points close a given evaluation point. While working well in low dimensional settings this type of approach suffers reduced accuracy in high dimensional settings for similar reasons as spline approximation. Again we are caught in a tradeoff between using large, well populated regions for local parameter approximation versus small, less populated regions. The bias/variance tradeoff inherent in this decision is identical to that for spline approximation.

The fourth technique that we consider is *projection pursuit regression.* This is the first adaptive technique that we discuss where no assumtions are made about the form of the underlying function. Projection pursuit regression (PPR) can be used to estimate a smooth function of several variables from noisy and scattered data.

The procedure models the regression surface as a sum of empirically determined univariate smooth functions of linear combinations of the predictor variables in an iterative manner. It is more general than standard stepwise and stagewise regression procedures, does not require the definition of a metric in the predictor space, and lends itself to graphical interpretation. Projection pursuit regression is limited, however, in its ability to approximate functions that involve interactions between the predictor variables.

The final technique that we consider are *decision tree classifiers.* A decision tree classifies examples into a finite number of classes. Nodes in the tree are labeled with attribute names, the edges are labeled with possible values (or groups/ranges of values) for this attribute, and the leaves are labeled with the different classes (or some probability distribution over the classes). A new case is classified by following a path down the tree, by taking the edges corresponding to the values of the attributes in the object. If the leaves consist of probability distributions over the classes then the case will be classified as a member of the class it is most likely to be in given the leaf it ended up in and given the probability distribution there. While decision trees have proved valuable in multivariate function approximation, they suffers from several limitations. The first of these limitations, and perhaps the most important is that the estimated function that results from building a decision tree is discontinuous at the subregion boundaries. Decision trees are, also, poor at approximating some very simple functions such as linear functions as well as additive functions. Recursive partitioning's inability to approximate such functions is due to the fact that the functions that result from this method tend to involve functions of more than a few variables. The **MARS** algorithm can be viewed as an extension of the decision tree approach that attempts to address these limitations.

4.1.1 Global Parametric Modelling

The simplest, most widely studied, and perhaps, most widely used method for function approximation in a high dimensional setting is global parametric modeling. The principal approach has been to assume a model, throughout the domain of interest, of the form

$$\mathbf{Y} \;=\; \mathbf{X}\boldsymbol{\beta} \;+\; \boldsymbol{\epsilon} \tag{4.3}$$

where

$\mathbf{Y}$ is an $(n \times 1)$ vector of response variables,

$\mathbf{X}$ is an $(n \times p)$ matrix of known form that reflects

the parametric function of choice,

$\boldsymbol{\beta}$ is an $(p \times 1)$ vector of parameters,

$\boldsymbol{\epsilon}$ is an $(n \times 1)$ vector of errors

and where $\mathsf{E}[\![\epsilon]\!] = \mathbf{0}$, $Var(\epsilon) = \mathbf{I}\sigma^2$, so the elements of $\boldsymbol{\epsilon}$ are uncorrelated[23]. In this setting, and paramteric modelling in general, the underlying function is assumed to be a member of a parametric set of functions whereas the noise is assumed to lie outside of that set. This assumption is reasonable since the chosen parametric function typically varies smoothly with changing values of the predictor variables while the noise varies randomly. The function estimation problem then simply reduces to that of estimating the parameters β from the training data.

Since $\mathsf{E}[\![\boldsymbol{\epsilon}]\!] = \mathbf{0}$, an alternative way of writing this model is

$$\mathsf{E}[\![\mathbf{Y}]\!] = \mathbf{X}\boldsymbol{\beta} \tag{4.4}$$

The error sum of squares is then

$$\begin{aligned}
\boldsymbol{\epsilon}'\boldsymbol{\epsilon} &= (\mathbf{Y} - \mathbf{X}\boldsymbol{\beta})'(\mathbf{Y} - \mathbf{X}\boldsymbol{\beta}) && (4.5)\\
&= \mathbf{Y}'\mathbf{Y} - \boldsymbol{\beta}'\mathbf{X}'\mathbf{Y} - \mathbf{Y}'\mathbf{X}\boldsymbol{\beta} + \boldsymbol{\beta}'\mathbf{X}'\mathbf{X}\boldsymbol{\beta} && (4.6)\\
&= \mathbf{Y}'\mathbf{Y} - 2\boldsymbol{\beta}'\mathbf{X}'\mathbf{Y} + \boldsymbol{\beta}'\mathbf{X}'\mathbf{X}\boldsymbol{\beta} && (4.7)
\end{aligned}$$

The least squares estimate of $\boldsymbol{\beta}$ is the value of $\mathbf{b}$, which, when subsituted in Eq.4.5, minimizes $\boldsymbol{\epsilon}'\boldsymbol{\epsilon}$. It can be determined by differentiating Eq.4.5 with respect to $\boldsymbol{\beta}$ and setting the resulting matrix equation equal to zero, at the same time replacing $\boldsymbol{\beta}$ by $\mathbf{b}$. This provides what are known as normal equations

$$(\mathbf{X}'\mathbf{X})\mathbf{b} = \mathbf{X}'\mathbf{Y} \tag{4.8}$$

If the p normal equations are independent, $\mathbf{X}'\mathbf{X}$ are nonsingular, and its inverse exists then the solution to the normal equations can be written[23]

$$\mathbf{b} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{Y} \tag{4.9}$$

This parametric approach has limited flexibility when the form of the underlying function varies over the domain of interest. Additionally, this approach is likely to produce poor approximations when the form of the true underlying function differs from the chosen parametric one. Global parametric models tend to be computationally very simple to fit to a set of training data and have the additional virtue of requiring relatively few data points. They are also easy to interpret and rapidly computable. Finally, if the noise component ϵ is large compared to $f(\mathbf{x})$, then the error associated with model misspecification may not be the most serious error that results from modeling the given data set. In noisy settings, a simple model like global parametric is often better at mitigating the problem of overfitting than a more complex approach[41].

4.1.2 Spline Fitting

In the theory of spline functions f is approximated by several simple parametric functions each defined over a different subregion of the domain of interest. The simplest example of function approximation by splines are splines of degree 0. In this simple case, we can write

$$\hat{f}(X) = \begin{cases} c_1 & X \in [t_1, t_2] \\ c_2 & X \in [t_2, t_3] \\ \vdots & \\ c_{n-1} & X \in [t_{n-1}, t_n] \end{cases} \tag{4.10}$$

where $t_1, t_2, \ldots, t_n$ are known as knot points. In this simple degree 0 case, the value of $\hat{f}$ is a step function defined by a constant value in each domain subregion. While such low order splines results in the inability to approximate even simple linear functions without the introduction of a huge number of knot points, models of this type are widely used in the construction of decision trees as will be seen later in this chapter[23].

The most widely used splines are of degree three (i.e. a cubic function of X is assumed in each knot interval), where their second order derivatives exist and are continuous. For simplicity, we will illustrate the spline fitting procedure here through the use of degree 2 or quadratic splines. The technique for higher order splines is analogous.

Let $\{R_j\}_{j=1}^{S}$ be a set of S disjoint subregions of D such that $D = \bigcup_{j=1}^{S} R_j$. A function $\hat{f}(x)$ is a *spline of degree* k if $\hat{f}(\mathbf{x})$ is a piecewise polynomial of degree-k, such that $\hat{f}, \hat{f}', \ldots, \hat{f}^{(k-1)}$ are all continuous. In the case of quadratic splines, a simple counting process shows us the number of conditions involved in defining such a spline. Each of the R_j points are called *knots*, and are the points where the function $\hat{f}(\mathbf{x})$ changes character. If there are S such knots, then there are $S-1$ subintervals and $S-2$ interior knots. Since the spline $\hat{f}(x)$ consists of quadratic polynomials of the form $a_i x^2 + b_i x + c_i$ over each subinterval $[R_i, R_{i+1}]$, there are $3(S-1)$ coefficients. We then expect that $3(S-1)$ conditions will fully define a quadratic spline function with S knots[23].

On each end of the subinterval $[R_i, R_{i+1}]$, the quadratic spline function $\hat{f}_i$ must satisfy the interpolation conditions $\hat{f}_i(R_i) = y_i$ and $\hat{f}_i(R_{i+1}) = y_{i+1}$. Since there are $S-1$ such subintervals, this imposes $2(S-1)$ conditions. The continuity of $\hat{f}'$ at each of the interior knots gives $S-2$ more conditions. Thus, we have $2(S-1)+S-2 = 3S-4$ conditions, or one condition short of the $3S-3$ conditions required. There are a variety of ways to impose an additional condition. Perhaps the most popular is $\hat{f}'(R_0) = 0$. The equations for the interpolating quadratic splines can now be uniquely determined[23].

When qth order splines are being used, while any set of basis functions that span the space of qth order spline approximations can be used, and the corresponding coefficients of the basis functions fit to the data using ordinary least squares, the procedure of spline approximation is typically implemented through the use of

$B-$splines.

The B splines of degree 0 are defined by

$$B_i^0 = \begin{cases} 1, & \text{if } t_i \le x < t_{i+1}; \\ 0, & \text{otherwise.} \end{cases} \tag{4.11}$$

With the functions B_i^0 as a starting point, we now generate all the higher degree B splines via the following recursive definition:

$$B_i^k(x) = \left(\frac{x - t_i}{t_{i+k} - t_i}\right) B_i^{k-1}(x) + \left(\frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}}\right) B_{i+1}^{k-1}(x) \quad k \ge 1 \tag{4.12}$$

B splines are widely used due to their favorable numerical properties. When knot point locations are being selected a priori and then the basis function coefficients fit to the data using least squares, then B splines are typically an appropriate choice of basis functions. In other settings, however, such as in the **MARS** algorithm discussed in the next chapter, where knot point locations are selected adaptively, there is a significant drawback to using B splines from an implemenation point of view. In this setting, the training data is used to determine optimal knot point locations. One can observe from the functional form of the B splines in 4.12 that each time a knot point location is changed multiple basis functions also change. This observation leads to a more complicated procedure for updating B splines than would be required if each basis function were associated with a single knot point. One set of basis function that offers this advantageous feature are the power basis functions.

In the univariate case with N regions separated by $N-1$ knot points, the power basis functions are as follows:

$$1, \ \{x^i\}_1^q, \ \{(x - t_j)_+^q\}_1^{N-1} \tag{4.13}$$

where $\{t_j\}_1^{N-1}$ are the knot locations. The minimization of the SSE can be done within each domain subregion as defined by the locations of the knot points in the same fashion as was done in the context of global parametric modeling (discussed previously), while maintaining the continuity of lower order derivatives at the knot points. While these basis functions do not possess the superior numerical properties of B splines, they have the property that changing a knot location changes only one basis function, leaving the others unchanged. This advantage will be exploited in an efficient implementation of the **MARS** algorithm.

The application of this approach to high dimensional data is trivial in principle. In practice, however, one runs into the well known *"curse of dimensionality"*. As the dimension of the data increases, the number of parameters that must be approximated increases exponentially. Additionally, with high (n) dimensional data, as the number of knot points increases, the data points within each region $[t_i, t_{i+1}]$ where $t \in \Re^n$ become increasingly sparse, thus resulting in poorer and poorer approximations. Finally, spline approximation assumes an a priori knowledge of appropriate placement of knot locations, and therefore, an a priori knowledge of where the form of the underlying function f changes. In practice, the appropriate choice of knot locations is unknown. As will be seen with decision trees and **MARS**, adaptive modeling techniques attempt to identify the appropriate choice of knot locations in a data driven manner.

4.1.3 Kernel and Nearest Neighbor Estimates

Local paramteric approximations take the form

$$\hat{\mathbf{f}} = \mathbf{X}\hat{\boldsymbol{\beta}}(\mathbf{x}) \tag{4.14}$$

where $\hat{\mathbf{f}}$ is a simple parametric function. Unlike global parametric approximations, here the parameter values, $\hat{\beta}(\mathbf{x})$, are generally different at each evaluation point $\mathbf{x}$ and are obtained by locally weighted least squares by minimizing

$$\sum_{i=1}^{N} w(\mathbf{x}, \mathbf{x_i})[y_i - \mathbf{X}\hat{\boldsymbol{\beta}}(\mathbf{x})]^2 \tag{4.15}$$

The weight function $w(\mathbf{x}, \mathbf{x}')$ (of $2n$ variables) is chosen to place the dominant mass on points $\mathbf{x}'$ close to $\mathbf{x}$. The properties of the approximation are mostly determined by the choice of w and to a lesser extent by the parametric function used[68].

The difficulty with applying local parametric methods in the presence of high dimensional data lies with the choice of an appropriate weight function w for the specific problem at hand. This strongly depends on the underlying function f and is, therefore, generally unknown. The most common choice

$$w(\mathbf{x}, \mathbf{x}\prime) \quad = \quad K(|\mathbf{x} - \mathbf{x}\prime|/s(\mathbf{x})) \tag{4.16}$$

with $|\mathbf{x} - \mathbf{x}\prime|$ being a weighted distance between $\mathbf{x}$ and $\mathbf{x}\prime$, $s(\mathbf{x})$ is a scale factor and K is a kernel function of a single argument. Commonly used scale functions include constant functions $s(\mathbf{x}) = s_0$ (kernel smoothing) and $s(\mathbf{x}) = s_0/\hat{p}(\mathbf{x})$ (nearest neighbor smoothing), where $\hat{p}(\mathbf{x})$ is an estimate of the density of the local predictor points.

In low dimensions, this approximation of the weight function w of $2n$ variables by a function K of a single variable, controlled by a single parameter s_0 is generally not too serious since asymptotic conditions can be realized without requiring large sample sizes. This is not the case in high (≥ 2) dimensions. The problem with a kernel based on distances between points is that the volume of the corresponding sphere in $n-$space grows as its radius to the nth power. Therefore, to ensure that

w places adequate mass on enough data points to control the variance of $\hat{f}(\mathbf{x})$, $s(\mathbf{x})$ will have to be very large, incurring high bias.

4.1.4 Projection Pursuit Regression

Each of the modeling techniques that we have discussed so far assumes that the functional form of the underlying function f is known, reducing the problem to one of estimating a set of parameters. To the extent that the chosen model is correct, these procedures can be successful. In practice, however, model correctness is difficult to verify and an incorrect model can yield poor predicitve results. For this reason emphasis in the statistical research community has been given to nonparametric regression techniques which make few assumptions about the underlying function. In nonparametric modeling the distinction between signal and noise is based solely on the notion of smoothness; $f(\mathbf{X})$ is assumed to be that component of $\mathbf{Y}$ that varies smoothly with changing values of $\mathbf{X}$, whereas the noise is taken to be the leftover part that does not. The effectiveness of a nonparamteric regression technique is determined by how well it can gauge the local smoothness properties of $f(\mathbf{X})$ and exploit them so as to filter out most of the noise without substantively impacting the signal.

Projection pursuit uses an approximation of the form

$$\hat{f}(\mathbf{x}) = \sum_{m=1}^{M} f_m\left(\sum_{i=1}^{n} \alpha_{im} x_i\right) \tag{4.17}$$

The approximation is constructed in an iterative manner:

(1) Initialize current residuals and term counter

$$r_i \leftarrow y_i, \ i = 1, \ldots, n \tag{4.18}$$

$$M \leftarrow 0 \tag{4.19}$$

(It is assumed that the response variable is centered: $\sum y_i = 0$)

(2) Search for the next term in the model:

For a given linear combination $Z = \alpha \cdot X$, construct a smooth representation $f_\alpha(Z)$ of the current residuals as ordered in ascending value of Z. Take as a criterion of fit $I(\alpha)$ for this linear combination the fraction of the so far unexplained variance that is explained by f_α:

$$I(\alpha) = 1 - \sum_{i=1}^{n}(r_i - f_\alpha(\alpha \cdot x_i))/\sum_{i=1}^{n} r_i^2 \tag{4.20}$$

Find the coefficient vector α_{M+1} that maximizes $I(\alpha)$

$$\alpha_{M+1} = \max_{\alpha}{}^{-1} I(\alpha) \tag{4.21}$$

(3) Termination

If the criterion of fit is smaller than a user specified threshold, stop. Else update the current residuals and the term counter

$$r_i \leftarrow r_i - f_{M+1}(\alpha_{M+1} \cdot x_i), \ i = 1, \ldots, n \tag{4.22}$$

$$M \leftarrow M + 1 \tag{4.23}$$

and go to step (2).

Although simple in concept, projection pursuit regression overcomes many limitations of other nonparametric regression procedures. The sparsity limitation (the curse of dimensionality) of kernel and nearest neighbor techniques is not encountered since all estimation is performed in a univariate setting. Progression pursuit regression does not require specification of a metric in the predictor space. Projection pursuit regression is also computationally quite feasible. For increasing sample size n, dimensionality p, and number of iterations M, the computation required to construct the model has a time complexity of $O(Mpn \log n)$.

Disadvantages of projection pursuit regression are that there are some simple functions that require large M for good approximation. This results from the inability of the regression model to approximate functions that include linear relationships between the predictor variables. It is also difficult to separate the additive from the interaction effects associated with the variable dependencies.

4.2 Decision trees

A *decision tree* classifies examples into a finite number of classes. Nodes in the tree are labeled with attribute names, the edges are labeled with possible values (or groups/ranges of values) for this attribute, and the leaves are labeled with the different classes (or some probability distribution over the classes). A new case is classified by following a path down the tree, by taking the edges corresponding to the values of the attributes in the object. If the leaves consist of probability distributions over the classes then the case will be classified as a member of the class it is most likely to be in given the leaf it ended up in and given the probability distribution there [65, 66].

During the process of building decision trees several issues must be confronted.

The first is: based on what data should we build the tree? Given a database of records one option is to use all the available records to build the tree. The problem with this approach, of course, is once the tree is built there is no unbiased data on which to test it. A better alternative for estimating the reliability of any classification model is to divide the data into a training set and a test set, build the model using only the training set, and examine its performance on the unseen test cases. This is satisfactory when there is plenty of data, but if there are only a small number of records available (a few hundred for example) then several problems arise. First, in order to get a reasonably accurate fix on the accuracy of our tree, the test set must be large, so the training set will be small. Second, when the total amount of data is small, different divisions of the data into a training set and a test set can produce surprisingly large variations in error rates on unseen (test) cases. Finally, if we divide the data arbitrarily, we are implicitly assuming that the process we are modeling is stationary. This may or may not be true in reality.

A more robust estimate of accuracy on unseen cases can be obtained by *cross-validation*. In this procedure, the data is divided into N blocks so as to make each block's number of cases and class distribution as uniform as possible. N different classification models (decision trees in this case) are then built, in each of which one block is omitted from the training data, and the resulting model is tested on the cases in that omitted block. In this way, each case appears in exactly one test. Provided that N is not too small - 10 is commonly used - the average error rate over the N unseen test sets is a good predictor of the error rate of a model built from all the data [66].

There are many related statistical issues that concern themselves with ensuring statistical validity of results reported by a given classification method as well as issues involved in the comparison of different classification methods [69]. As these are not issues unique to decision trees but rather to all model building, we will not discuss them in more significant depth here, but rather focus on the issues unique to building effective decision trees.

The three issues fundamental to the construction of decision trees that we will discuss in the following sections are as follows:

1. At a given node how do we decide which attribute on which to partition the data at this point (or whether to partition it at all)?

2. Once the attribute has been chosen on which to partition the data, how do we partition it? Of course, if the attribute is discrete-valued one option would be to create a new node for each attribute value. However, this might not be desirable for attributes with many values. For continuous valued attributes this approach is not even possible and the attributes must be partitioned in some reasonable way.

3. Finally, once a tree has been built we need to determine whether and how to prune it. The goal of pruning would be to simplify and reduce the error rate of the existing tree.

After reviewing approaches to these issues, this chapter will conclude the discussion of decision trees by considering a few refinements to the basic process.

4.2.1 Attribute Selection

Given a set of training cases T and a set of classes $\{C_1, C_2, \ldots, C_k\}$ into which we are attempting to classify those training cases there are certainly many possible ways to build a decision tree. In fact, any test that divides T in a nontrivial way, so that at least two of the subsets $\{T_i\}$ are not empty, will eventually result in a partition into single-class subsets, even if many of them contain a single training case.

However, building a tree that is consistent with the training data is not the sole goal of the tree building process. In fact, one possible tree we could build would consist of a leaf for each training set. This tree would certainly result in errorless classification of the training data but would perform relatively poorly on out-of-sample data. This classification represents an extreme case of *overfitting*. Thus, our goal, in addition to minimizing error, is to build a compact tree that reveals the structure of the domain we are considering and thus has sufficient predictive power.

Since we are looking for a compact decision tree that is consistent with the training set, why not explore all possible trees and select the simplest? Unfortunately, the problem of finding the smallest decision tree consistent with a training set is NP-Complete [47]. With this in mind, decision tree construction methods are based on heuristics for selecting attributes on which to partition the data. Most of these heuristics are non-backtracking, greedy algorithms. Once a test has been selected to partition the current set of training cases, usually on the basis of maximizing some local measure of progress, the choice is not revisited and the consequences of alternative choices is not explored. This approach makes the choice of the attribute on which to partition particularly important.

In CART, Breiman [16][15] derives his goodness of split criteria from an impurity function.

Definition 6 *If there are J classes into which we are classifying cases, an impurity function is a function Θ defined on the set of all J-tuples of numbers $(p_1, \ldots, p_J)$ satisfying $p_j \geq 0$, $j = 1, \ldots, J$, $\sum_j p_j = 1$ with the properties*
(i) Θ is maximized at $(\frac{1}{j}, \ldots, \frac{1}{j})$
(ii) Θ is minimized at $(1, \ldots, 0), (0, 1, \ldots, 0), \ldots, (0, \ldots, 0, 1)$
(iii) Θ is a symmetric function of $p_1, \ldots, p_j$

Definition 7 *Given an impurity function Θ, define the impurity measure $i(t)$ of any node t as*

$$i(t) \;=\; \Theta(p(1|t), \ldots, p(J|t)) \tag{4.24}$$

Given an impurity function Θ, we would like to choose the attribute to split on at a given node that results in a maximal decrease in impurity. In the tree building methods developed by Quinlan if an attribute, A, takes multiple values then when a node is split on A a node is created for each of these values. In contrast, in Breiman's method all splits are binary. In this case, if a split s of a node t sends a proportion p_R of the data cases in t to t_R and the proportion p_L to t_L, define the decrease in impurity to be

$$\Delta i(s,t) \;=\; i(t) \;-\; p_R i(t_R) \;-\; p_L i(t_L) \tag{4.25}$$

Then we can take the goodness of split $\Theta(s,t)$ to be $\Delta i(s,t)$.

Suppose we have done some splitting and arrived at a current set of terminal nodes. The set of splits used, together with the order in which they were used, determines a binary tree T. Denote the current set of terminal nodes by $\tilde{T}$; set $I(t) = i(t)p(t)$, and define the tree impurity $I(t)$ by

$$I(t) \;=\; \sum_{t \in \tilde{T}} I(t) \;=\; \sum_{t \in \tilde{T}} i(t)p(t) \tag{4.26}$$

It is clear that selecting the splits that maximize $\Delta i(s,t)$ is equivalent to selecting those splits that minimize the overall tree impurity $I(T)$.

$$\begin{aligned} \Delta I(s,t) &= \{i(t) - p_L i(t_L) - p_R i(t_R)\}p(t) && (4.27) \\ &= \Delta i(s,t)p(t) && (4.28) \end{aligned}$$

This expression can be extended in the obvious way if we allow multivalued, as opposed to binary, splits. Since $\Delta I(s,t)$ differs from $\Delta i(s,t)$ by the factor $p(t)$, the same split s^* maximizes both expressions. Therefore, by choosing the split s at node t that minimizes the impurity of that node we are also minimizing the impurity of the entire tree. We still, of course, need to choose an impurity function consistent with these ideas. Many such functions have been suggested in the literature and we will present several of them here.

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 $\mathsf{Pr}[\![a,c]\!]$ be the probability of drawing a record uniformly from S with $A = a$,

$C = c$. Define $\mathsf{Pr}[\![a]\!], \mathsf{Pr}[\![c]\!], n(a)$, and $n(c)$ in the obvious way, and let $n = |S|$. Note, in addition, that many of the attribute selection methods utilize notions from information theory. The reader is assumed to have some basic knowledge in this area. For a good review see [25].

Information Gain

The first attribute selection method we discuss was introduce by Quinlan in his ID3 system [65]. Recall that the entropy of a discrete random variable X is the average length of the shortest description of the random variable. It, in some sense, represents the amount of information contained in X. More precisely:

Definition 8 *The* entropy H(X) *of a discrete random variable X is defined by*

$$H(X) \;=\; -\sum_{x \in X} \mathsf{Pr}[\![x]\!] \log_2 \mathsf{Pr}[\![x]\!] \tag{4.29}$$

For our purposes, assume S is the subset of the original training set currently being partitioned. Also let $C = \{C_1, \ldots, C_k\}$ be the set of classes into which records are being classified. Then define,

$$H(C) \;=\; -\sum_{j=1}^{k} \mathsf{Pr}[\![C_j]\!] \times \log_2(\mathsf{Pr}[\![C_j]\!]) \tag{4.30}$$

where $\mathsf{Pr}[\![C_j]\!]$ is the probability that a record drawn randomly from S is an element of class C_j. $H(C)$ measures the average amount of information needed to identify the class of a case in S. Or, in other words, it conveys the amount of uncertainty in S about the class membership of records in S.

Mutual Information is a measure of the amount of information that one random variable contains about another random variable. It is the reduction in the uncertainty of one random variable due to the knowledge of the other.

Definition 9 *Consider two random variables X and Y. The* mutual information *is defined as*

$$I(X;Y) \;=\; H(Y) \;-\; H(Y|X) \tag{4.31}$$

For our own purposes, we would like to partition on the attribute that conveys the most information about class membership of the records in S. More precisely, we would like to choose the attribute A that maximizes $I(A;C)$.

We calculate this by considering each available attribute, A_i, in turn and computing the conditional entropy, $H(C|A)$, the entropy of C given that we partition on attribute A_i. Assuming A_i has n possible outcomes

$$H(C|A_i) \;=\; \sum_{j=1}^{n} \frac{|S_j|}{|S|} \times H(S_j)\ , \quad \text{where } S_j = j^{th} \text{ partition of } S \tag{4.32}$$

Then, the mutual information (or information gain as termed by Quinlan) is

$$I(A_i;C) \;=\; H(C) \;-\; H(C|A_i) \tag{4.33}$$

We choose to partition on the attribute that maximizes this value.

Note that in this discussion we have assumed that all attributes take discrete values and that when partitioning on an attribute we want to create a node for *each* of its values. Neither of these assumptions are true, in general. We will discuss how to include, in this framework, both continuous valued attributes and the grouping of attribute values shortly.

Gain Ratio

Although mutual information performs well as a partition criterion, it has a serious deficiency - it has a strong bias in favor of attributes that take many values. This bias can been seen in the following example:

Example:

Consider the problem confronting a credit card company of classifying prospective card-holders based on their credit worthiness. Further, assume that the database table containing card-holder information (credit history, demographic information,etc ...) includes an ID field that uniquely identifies each prospective card-holder. Partitioning based on this attribute would lead to a large number of subsets, each containing just one record. Since all of these one-record subsets necessarily contain records of a single class, $I(A_i; C) = 0$, so the mutual information from using this attribute to partition the set of training records is maximal. From the point of view of prediction, however, such a division is useless.

The bias inherent in the mutual information criterion can be rectified by a normalization process in which the apparent mutual information attributable to tests with many values is adjusted. The new criterion resulting from this normalization is now defined [66]

Definition 10 *The GainRatio criterion is*

$$GainRatio(A_i) = \frac{H(A_i; C)}{H(A_i)} \tag{4.34}$$

where A_i is the attribute in question and S is the subset currently being partitioned.

This expresses the proportion of information generated by the split that is useful, i.e. that appears helpful for classification. In other words, we divide out the information that pertains to a record, that indicated not the class to which the record belongs, but the outcome of the attribute itself.

If the split is near-trivial, i.e. creates subsets that are very unbalanced in their size, $H(A_i)$ will be small and the GainRatio unstable. To avoid this, the GainRatio criterion selects an attribute for partitioning to maximize the ratio above, subject to the constraint that the mutual information must be large - at least as great as the average mutual information over all tests examined.

It is apparent that the prospective card-holder ID attribute, in the above example, will not be ranked highly by this criterion. If there are k classes, the numerator (mutual information) is at most $\log_2(k)$. The denominator, $H(A_i)$, on the other hand is $\log_2(n)$, where $n = |S|$. It seems reasonable to presume that the number of training cases at a node is larger than the number of classes, so the ratio would have a small value.

Splitting Criteria in CART

In CART, two splitting criteria are used. The first is the Gini criterion which utilizes the Gini index as an impurity function.The Gini index has the form

$$i(t) \;=\; \sum_{j \neq i} p(j|t)p(i|t) \tag{4.35}$$

The Gini index has the following interpretation. If we choose an object from a node t at random, we will classify it as a member of class i with probability $p(i|t)$. The probability that it is actually a member of class j is $p(j|t)$. Therefore, the probability of misclassification is exactly the Gini index.

Recall, that we will choose to split the node t with the split s that maximizes $\Delta i(s,t)p(t)$. That is, for each possible split s, we will maximize $i(t) - p_R i(t_R) - p_L i(t_L)$ where t_R and t_L represent the two new nodes that result from the split s.

The second approach used in CART is called the *Twoing Criterion*. Denote the set of classes C as $C = \{1, \dots, J\}$. At each node, separate the classes into two superclasses,

$$C_1 = \{j_1, \dots, j_n\}, C_2 = C - C_1 \tag{4.36}$$

Call the objects whose class is in C_1 class 1 objects, and all other objects class 2 objects.

For any split s of the node t, compute $\Delta i(s,t)$ where $i(t) = p(1|t)p(2|t)$. In other words use the Gini index for the two-class problem. Of course, $\Delta i(s,t)$ depends on the choice of c_1, so the notation $\Delta i(s,t,C_1)$ is used. Now find the split $s^*(C_1)$ which maximizes $\Delta i(s,t,C_1)$. Then find the superclass C_1^* which maximizes $\Delta i(s^*(C_1),t,C_1)$. The split used on the node is $s^*(C_1^*)$.

The idea is, at every node, to group the classes in such a way as to maximize the decrease in node impurity if we were to consider this a two-class problem. This approach has the advantage that it results in "strategic" splits. At each node, the classes are sorted into two groups that are in some sense most dissimilar and given these groups outputs the best split s^*. Near the top of the tree, this criterion attempts to group together large numbers of classes that are similar in some characteristic. Near the bottom of the tree it attempts to isolate single classes.

4.2.2 Partitioning Attributes

The next issue we must address is: given a selected attribute on which to partition the subset of records at a given node, how do we partition them? Until now, we have assumed that all attributes take discrete values and that a node will be created for each of these values (at least each of the values present in S). We have yet to describe how to deal with attributes that take continuous values as well as the possibility of grouping attribute values at a newly created node. In CART these methods are always used since all splits must be binary. In Quinlan's systems the grouping of categorical attributes and partitioning of continuous ones are included as an option. These will be the topics of the next two sections.

Partitioning Continuous Attributes

If an attribute A has continuous numeric values, we will partition the records by comparing values of A to some threshold Z. That is, all records whose value of $A \leq Z$ will be grouped together as will all records whose value of $A < Z$.

It may seem difficult to choose a Z that will result in an ideal split of A but Breiman [16] introduces a straightforward approach. The training cases to be partitioned, T, are first sorted on the values of the attribute A being considered. There are only a finite number of these values, so let us denote them in order as $\{v_1, \ldots, v_m\}$. Any threshold value lying between v_i and v_{i+1} will have the same effect of dividing the cases into those whose value of the attribute A lies in $\{v_1, \ldots, v_i\}$ and those whose value is $\{v_{i+1}, \ldots, v_m\}$. There are thus only $m - 1$ possible splits on A, all of which are examined. For each possible split, the splitting criterion is evaluated. For example, if we are using mutual information as our splitting criterion, we will calculate $I(A_i; C)$ for each possible split of the continuous valued attribute A_i. Assume that this maximal value sets the threshold between v_k and v_{k+1}. If $I(A_i; C)$ using this split is greater than $I(A_j; C)$, $i \neq j$, for all other attributes, then the training set will be partitioned on A_i with the threshold between

v_k and v_{k+1}.

Some systems use the midpoint $\frac{v_k+v_{k+1}}{2}$ as the threshold, while others like Quinlan's C4.5, choose the largest value of A in the entire training set that does not exceed the midpoint above. This ensures that all threshold values appearing in trees actually appear in the data.

It may seem expensive to examine all $m-1$ possible thresholds, but, when the cases have been sorted as above, this can be carried out in one pass, updating the class distributions to the left and right of the threshold on the fly.

4.2.3 Grouping Attribute Values

The two questions that must be answered regarding the grouping of attribute values are:

1. When do we choose to group attributes?, and
2. Given that the decision has been made to group attributes, how is it done?

We deal with these questions in turn.

When the criteria described so far decide to split the training set on a discrete valued attribute, they generate a separate node for each possible value of that attribute. If there are many such values, however, resulting in nodes with small subsets of training cases, this approach has two drawbacks. The first potential problem is that useful patterns in the subsets may become undetectable due to an insufficiency of data. The second type of problem that might arise is dependent on the splitting criterion being used. For example, the GainRatio criterion measures the ratio of information relevant to classification that is provided by the division to the information produced by the division itself. The denominator of this ratio grows rapidly as the number of subsets increases. The GainRatio criterion, in other words, is biased against attributes with many values. It, therefore, may make sense to group attribute values under this criterion.

There are three potential approaches to determining when and if a given attribute's values should be grouped. The first option would be to consider possible groupings of attribute's values (henceforth, called *value groups*) before the tree is built. This would involve considering groupings of many-valued attributes, choosing the best possible grouping (under the splitting criterion being used) and either adding new attributes with the reduced numbers of values or using them in place of the original attributes from which they were formed. However, the most appropriate division of an attribute's values into value groups will not remain constant but will reflect the contexts established in different parts of the tree. Thus, although this approach is computationally economical, it is flawed.

The second approach would be to determine an optimal grouping of an attribute's values after that attribute has been chosen as the one to be split. The problem with this approach is that a given attribute may never be chosen to be split *unless* its values are grouped effectively. For example, if we are using the GainRatio as our splitting criterion, since the denominator will be large without grouping attributes for an attribute with many values, the attribute may never be chosen to be split. However, there may exist a value grouping that mitigates the problem with the denominator large enough to make this an appealing attribute to split. Therefore, we would prefer an approach that considers possible value groups before determining the attribute to be split.

The final approach finds value groups of multivalued attributes each time a possible split of the training cases is being evaluated. At this time, the best groupings are determined for each multivalued discrete attribute and its splitting criterion calculated using the partition induced by the value groups. This continual re-evaluation of value groups can require a substantial increase in computation, especially for domains in which many discrete attributes have many possible values. It is, however, the most effective approach.

Determining Value Groups

If a multivalued attribute takes n different values then even if we are only considering splitting the training cases into two value groups there are $2^{n-1} - 1$ nontrivial partitions of these values. There is certainly no way to consider all of these possible groupings for large values on n - not to mention the fact that we would typically like to consider more than just binary splits.

In one special case there is a solution to this computational quagmire. Breiman et al. [16] prove that, if there are just two classes into which we are classifying the training records, it is possible to order the values so that the best partition is one of the "cuts" of this sequence - instead of $2^{n-1} - 1$ possibilities, only $n - 1$ need to be examined.

Theorem 3 *Let A be an attribute taking the values $\{a_1, \ldots, a_L\}$. Additionally, let C_1 and C_2 be the two classes, let $\mathsf{Pr}[\![C_1 | A = a_l]\!]$ denote the probability that a record is in class C_1 given that the value of $A = a_l$, and let Θ denote the splitting criterion. Then, order the $\mathsf{Pr}[\![C_1 | A = a_l]\!]$, that is,*

$$\mathsf{Pr}[\![C_1 | A = a_{l_1}]\!] \;\leq\; \mathsf{Pr}[\![C_1 | A = a_{l_2}]\!] \;\leq \ldots \leq\; \mathsf{Pr}[\![C_1 | A = a_{l_L}]\!]. \tag{4.37}$$

then one of the L subsets

$$\{a_{l_1}, \ldots, a_{l_h}\} \quad h = 1, \ldots, L \tag{4.38}$$

is optimal.[1]

In general, however, there is no nice way to find an optimal grouping. Therefore, a greedy algorithm is used to find value groups. Of course, these groups may not be optimal.

The method used in Quinlan's C4.5 is based on iterative merging of value groups. The initial value groups are just the individual values of the attribute under consideration and, at each iteration, the consequences of merging every pair of groups is considered. The process continues until just two value groups remain, or until no such merger would produce a better partition of the training cases.

The partition arising from any particular collection of value groups is evaluated by whatever selection criterion is in force. Under the GainRatio criterion, for example, merging two groups of attribute values results in fewer subsets of training cases and a corresponding reduction in the denominator. If the reduction in the numerator (mutual information) is not substantial, the final GainRatio may be increased.

In C4.5, some additional requirements are imposed such as the ones ensuring that each resulting value group have some minimal number of elements. Additionally, the final split should have at least half the mutual information of the original multivalued split.

[1] I have left out a detail of the statement of this theorem concerning functional requirements on Θ. All of the splitting criterion that have been considered satisfy these requirements.

4.2.4 Additional Issues

Decision trees are a well developed area of statistical modeling and as a result there has been much effort to improve the various stages of tree building. As can be seen by the number of splitting criteria in use there is no consensus on any "best" approach. There seem to be many domain dependent issues in decision tree building. It will not be possible to do justice to the entire area in this short space. There are a few additional issues that I would like to discuss briefly.

Multivariate Splits

Each of the divisions that results from the splitting criteria that have been discussed so far correspond to a special kind of surface in the description space, namely a hyperplane that is orthogonal to the axis of the tested attribute and parallel to all other axis. Thus, the regions produced by a decision tree using such criteria are not arbitrary, but rather, are hyper-rectangles. This represents a major limitation of decision tree classification methods. When the task at hand is such that class regions are not hyper-rectangles, the best that a decision tree can do is to approximate the regions by hyper-rectangles. Certain types of simple functions, are, however, difficult to approximate by decision trees given this limitation. For example, consider a bank deciding whether or not a prospective customer is worthy of a loan. Assume that three attributes of the customer available to the bank are, her current income, her current savings, and her current debt. The decision whether or not to give her a loan will be based not any one of these attributes, but rather a linear combination of the three. We could represent this combination in our tree by consecutively splitting the data on each of these three attributes. However, it makes more sense, in terms of minimizing the size and complexity of the resulting tree to allow splits on linear combinations of attribute values. Another type of function that is difficult to estimate with decision trees are boolean functions. In a medical diagnosis problem, for example, we may want to split the data based on whether the patient has a fever *and* a rash. Or more generally, given boolean attributes A_i, does the case have property (D_1 and D_2) or (D_3 and D_4). In CART, modifications to the basic tree building procedure are available as options to deal with these limitations.

Linear Combinations

The details of the algorithm for determining good linear combination splits is rather involved. Here I will sketch the idea. For further details see [16]. Suppose there are M_1 ordered variables (categorical variables are excluded). At every node t, take the set of coefficients $\mathbf{a} = (a_1, \ldots, a_{M_1})$ such that $\|\mathbf{a}\|^2 = \sum_m a_m^2 = 1$, and search for the best split of the form

$$\sum_m a_m x_m \leq c \tag{4.39}$$

as c ranges over all possible values. Denote this split by $s^*(\mathbf{a})$ and the corresponding decrease in impurity by $\Delta i(s^*(\mathbf{a}, t)$. That is,

$$\Delta i(s^*(\mathbf{a}^*), t) \;\; = \;\; max_{\mathbf{a}} \Delta i(s^*(\mathbf{a}), t) \tag{4.40}$$

This produces a linear split of the form

$$\sum_m a_m^* x_m \leq c^* \tag{4.41}$$

The implementation of a search algorithm for maximizing $\Delta i(s^*(\mathbf{a}), t)$ over the large set of possible values of $\mathbf{a}$ is in [16].

This process often results in a complicated tree structure. At each node one has to interpret a split based on a linear combination of all ordered variables. Some of these attributes, however, may contribute little to the effectiveness of the split and can, therefore, be pruned.

For m ranging from 1 to M_1, vary the threshold constant c and find the best split of the form

$$\sum_{\tilde{m} \neq m} a^*_{\tilde{m}} x_{\tilde{m}} \leq c_m \tag{4.42}$$

That is, find the best split using the linear combination with coefficients $\mathbf{a}^*$ but deleting x_m and optimizing on the threshold c. Denote the decrease in impurity using this split by Δ_m and

$$\Delta^* = \Delta i(s^*(a^*), t) \tag{4.43}$$

The most important single variable to the split $s^*(a^*)$ is the one whose deletion causes the greatest deterioration in performance. More specifically, it is that variable for which Δ_m is a minimum. Similarly, the least important variable is the one for which Δ_m is a maximum.

The deterioration due to deleting the most important variable is $\Delta^* - min_m \Delta_m$, and the deterioration due to deleting the least important variable is $\Delta^* - max_m \Delta_m$. Set a constant β (usually 0.1 or 0.2), and if

$$\Delta^* - max_m \Delta_m < \beta(\Delta^* - min_m \Delta_m) \tag{4.44}$$

then delete the least important variable.

This procedure is repeated on the remaining undeleted variables until no more deletion occurs. If we denote the indices of the undeleted variables by $\{m_1\}$ then the node splitting algorithm is used again to find the best split of any linear combination of these remaining variables.

Boolean Combinations

Since the class of all Boolean combinations of splits is extremely large and can lead to a confusing tree, the class of Boolean combinations considered is restricted to splits of the form

$$\{Is\ x_{m_1} \in B_1\ and \ldots and\ x_{m_h} \in B_h\} \tag{4.45}$$

When the complimentary node is considered, it also includes splits of the form

$$\{Is\ x_{m_1} \in B_1\ or \ldots or\ x_{m_h} \in B_h\} \tag{4.46}$$

The class (4.45) of Boolean splits is denoted as

$$s_{m_1} \cap s_{m_2} \cap \ldots \cap s_{m_n} \tag{4.47}$$

and interpreted as the set of all cases sent to t_L by every split in the set $\{s_{m_1}, \ldots, s_{m_n}\}$. Denote the decrease in impurity of the node t by the split as

$$\Delta i(s_{m_1} \cap s_{m_2} \cap \ldots \cap s_{m_n}, t) \tag{4.48}$$

Ideally, we would like to maximize the above value over all splits on variables $x_{m_1}, \ldots, x_{m_n}$ and then maximize over all subsets $\{m_1, \ldots, m_n\} \subset \{1, \ldots, M\}$. There is no efficient algorithm known to do this in a reasonable way so, instead, a stepwise method is used.

If a split s on an ordered variable x is of the form $\{Is\ x \leq c?\}$, let $\bar{s}$ be the split $\{Is\ x > c?\}$. If s is a split on a categorical variable x of the form $\{Is\ x \in \{b_1, \dots, b_h\}?\}$, denote by $\bar{s}$ the split $\{Is\ x \notin \{b_1, \dots, b_h\}?\}$. Additionally, let s_m^* be the best split on the variable x_m and take S^* to be the set of splits

$$S^* = \{s_1^*, \bar{s}_1^*, \dots, s_M^*, \bar{s}_M^*\} \tag{4.49}$$

The stepwise procedure proceeds as follows:

1. If $s_{m_1}^*$ is the best split in S^*, find the $s \in S^*$ that maximizes

$$max(\Delta i(s_{m_1}^* \cap s, t), \Delta i(\bar{s}_{m_1}^* \cap s, t)) \tag{4.50}$$

If that maximum value is $\Delta i(s_{m_1}^* \cap s^*, t)$, denote $s_1^* \cap s_2^* = s_{m_1}^* \cap s^*$. If the maximum value is $\Delta i(\bar{s}_{m_1}^* \cap s^*, t)$, denote $s_1^* \cap s_2^* = \bar{s}_{m_1}^* \cap s^*$.

2. Find the s in S^* that maximizes $\Delta i(s_1^* \cap s_2^* \cap s, t)$. If the maximum is achieved at $s = s^*$, denote $s_1^* \cap s_2^* \cap s_3^* = s_1^* \cap s_2^* \cap s^*$. Continue adding splits to this intersection until step 3 is satisfied.

3. Fix $\beta > 0$; if at any stage in step 2

$$\Delta i(s_1^* \cap \ldots \cap s_{n+1}^*, t) \leq (1 + \beta)\Delta i(s_1^* \cap \ldots \cap s_n^*) \tag{4.51}$$

then stop and use the split $s_1^* \cap \ldots \cap s_n^*$

Soft Thresholds

In the discussion of continuous attributes a threshold was chosen, and a binary split done, around this value. The problem with this approach is that there is no distinction made between values close to the threshold versus values far away from the threshold. If there are many values close to the threshold then small perturbations in the data can potentially produce radically different classifications. Thus, some have experimented with soft thresholds [20].

Soft thresholds is based on the following idea: rather than classifying a case as being in subset 1 (below the threshold) or in subset 2 (above or equal to the threshold) give some weighting to the probability of it being in each subset and distribute it across both subsets. For example, one could potentially add 0.6 records to subset 14 and 0.4 records to subset 2. These fractional records would then be treated as such in the determination of future splits and in the pruning phase.

How these distributions are determined is the subject of this area of research but will not be discussed here.

Missing Values

It is not unlikely that when dealing with real life data that there will be missing attribute values [66]. This might occur because the value is not relevant to a particular case, was not recorded when the data was collected, or was lost at some stage of preparing the data for the tree building process.

Typical approaches to this problem are probabilistic in nature. Similar to the approach in soft partitions, when at some node in the tree, if a case needs to be classified based on a missing attribute value the probability of it belonging to each of the children of the node is determined based on the other cases at the node. Then fractions of it may be classified in each of the subsets at the children of the node. In other words, if there is a 10% chance it belongs to subset 41 then 0.1 records will be added to subset 1, similarly for all other subsets.

There are two possible places in which a missing attribute might be encountered.

Either in trying to classify an unseen case or in a training case in the course of building the tree.

In the first case, rather than classifying the case definitely, we generate a probability distribution over the classes and choose the one that the given case is most likely to be a member of.

In the latter case we must decide how to take this missing value into account when evaluating the splitting criterion. Approaches to this issue are discussed in [21].

4.2.5 Deficiencies of Recursive Partitioning

While decision trees have proved valuable in multivariate function approximation, they suffers from several limitations. The first of these limitations, and perhaps the most important is that the estimated function that results from building a decision tree is discontinuous at the subregion boundaries. In addition to its obvious cosmetic problem, this limitation severely limits the accuracy of the approximation if the underlying function is itself continuous.

Decision trees are, also, poor at approximating some very simple functions such as linear functions, i.e. functions of the form

$$f(\mathbf{x}|\{a_j\}_0^p) \;=\; a_0 \;+\; \sum_{i=1}^{p} a_i x_i, \qquad p \leq n \tag{4.52}$$

as well as additive functions, i.e functions of the form

$$f(\mathbf{x}) \;=\; \sum_{j=1}^{n} g_j(x_j) \tag{4.53}$$

Recursive partitioning's inability to approximate such functions is due to the fact that the functions that result from this method tend to involve functions of more than a few variables. To see why this claim holds consider what happens each time a split is performed. A basis function (leaf) of lower order interaction is replaced by two new functions (leaves), each with interaction order one level higher. As this progresses, the result is a set of basis functions with high order interactions amongst the variables.

Chapter 5

Multivariate Adaptive Regression Splines

Up to this point we have taken a geometric view of decision tree classification. Our approach has been to recursively partition a set of input vectors in order to classify a single output value. While this view is very intuitive, this process can also be viewed as a stepwise regression procedure. After first justifying our selection of **MARS** we then cast decision tree classification as a stepwise regression procedure through the construction of spline basis, and finally show how some simple extensions to stepwise regression results in the **MARS** algorithm, originally due to Friedman[41].

5.1 Choosing a Classifier

Our goal in selecting a classifier is to keep it general enough so that it is comparable to the more powerful heuristic approaches such as neural networks in terms of its ability to allow robust basis functions and classifier systems that use adaptive, multiphase regression. Neural networks were rejected as a classification technique for computational reasons given the dimensionality and size of our data sets, because of the lack of interpretability of the resulting models, and because **MARS** ultimately offers similar power in its ability to represent complex nonlinear relationships.

The classifiers discussed in Chapter 4 can be broadly divided into two categories - parametric and nonparametric. Among the parametric procedures, by definition, the correct form of the underlying model must be assumed. It is often the case in reality that a practitioner will not know the true parametric form of the underlying function. Therefore, while these approaches are typically easy to implement, computationally efficient, and straighforward to evaluate and interpret, they are limited in their predictive capacity.

Given the complexity of intrusion detection data, it is unlikely that a simple parametric function would be capable of modeling the variability of the underlying function throughout all regions of the domain. For example, the features relevant to detecting an intrusion will certainly vary among different attacks. Therefore, any global parametric approach would not suffice. Spline, kernel, and nearest neighbor approximation all offer a mechanism for addressing local variation in the underlying function. They, however, all suffer from the "curse of dimensionality" and their performance is significantly degraded in high dimensional settings. Additionally, they all require some restricitive assumptions, that when wrong, again degrade their predictive capacity. In spline approximation knot locations must be chosen and in kernel and nearest neighbor approximation a weight function must be chosen.

The other two techniques discussed in Chapter 4 are both adaptive. Projection pursuit suffers from its inability to capture relationships between predictor variables that are certainly present in intrusion detection data. Regression trees offer a powerful classification technique but suffer from a lack of continuity at subregion boundaris and their inablilty to approximate some simple functions. **MARS** will be developed as an extension to the decision tree approach with some modifications that address these shortcomings.

As will be shown shortly **MARS** can be viewed as a zero order adaptive spline approximation. It is adaptive because knot locations are selected dynamically during the model building process. It is a zero order spline approximation if one views a path in the decision tree as tensor product of indicator functions. The lack of continuity of decision trees can be remedied by replacing zero order splines with linear spline functions. This, however, introduces a problem as well. The decision tree construction algorithm allows multiple splits of the same predictor variable along a path in the tree. While this is not problematic in the case of zero order splines, in the linear spline setting this introduces higher order dependencies amongst the variables in the resulting basis function. To remedy this we disallow multiple partitions of any variable along a single tree path. While this restriction, on its own, limits the power of our algorithm, the power can be restored, and the ability to approximate some simple functions that decision trees are typically unable to approximate can be achieved by allowing multiple splits of a single node during the tree construction process. **MARS** is simply the decision tree algorithm with the above modifications. Through these modifications we achieve our goal of selecting a classification technique that retains the computational tractability and the interpretability of decision trees while extending their expressive power in terms of their ability to approximate some simple functions as well as more complex nonlinear functions. Next we cast decision tress as a stepwise regression technique and then derive **MARS** as an extension to stepwise regression.

5.2 Decision Trees as Stepwise Regression

Let y represent a single univariate response variable that depends on a vector of p predictor variables $\mathbf{x}$ where $\mathbf{x} = (x_1, \ldots, x_p)$. Assume we are given N samples of y and $\mathbf{x}$, namely $\{y_i, \mathbf{x}_i\}_{i=1}^N$, and that we can describe y with the regression model, $y = f(x_1, \ldots, x_p) + \epsilon$ over some domain $D \subset \Re^p$, which contains the data. The function $f(\mathbf{x})$ reflects the true but unknown relationship between y and $\mathbf{x}$. The random additive error variable ϵ, which is assumed to have mean zero and variance σ_ϵ^2, and reflects the dependence of y on quantities other than $\mathbf{x}$. The goal in regression modeling is to formulate a function $\hat{f}(\mathbf{x})$ that is a reasonable approximation of $f(\mathbf{x})$ over the domain D.

As discussed in the last chapter a typical approach to this problem is through the use of splines. The decision tree, or recursive partitioning regression model, can be viewed similarly. Through recursive partitioning as we build a decision tree we end up with a set of nodes that form a disjoint partition of the domain of $\mathbf{x}$. After pruning the tree, which is equivalent to merging two or more of these disjoint subsets, each path from the root of the tree to a leaf is one of our interpolating spline functions. More formally, the classification function resulting from a tree can be stated as

$$\text{if } \mathbf{x} \in R_m, \text{ then } \hat{f}(\mathbf{X}) = g_m(\mathbf{X} | \{a_j\}_1^p) \tag{5.1}$$

Here $\{R_m\}_1^M$ are disjoint subregions representing the partition of D. In our tree building methodology the functions g_m are simple constant functions

$$g_m(\mathbf{x}|a_m) = a_m \tag{5.2}$$

As discussed in the last chapter one of the most significant deficiencies of recursive partitioning algorithms is their lack of continuity at subregion boundaries. In response to this, and other deficiencies Friedman [41] has developed the MARS system. The starting point is to cast the approximation (5.1) and (5.2) in the form of an expansion in a set of basis functions. As before, we have N samples of y and $\mathbf{x} = (x_1, \ldots, x_p)$, namely $\{y_i, \mathbf{x}_i\}_{i=1}^N$. Let $\{R_j\}_{j=1}^S$ be a set of S disjoint subregions of D such that $D = \bigcup_{j=1}^S R_j$. Given the subregions $\{R_j\}_{j=1}^S$, recursive partitioning estimates the unknown function $f(\mathbf{x})$ at $\mathbf{x}$ with

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{S} \hat{f}_j(\mathbf{x}) B_j(\mathbf{x}), \tag{5.3}$$

where

$$B_j(\mathbf{x}) = I[\mathbf{x} \in R_j], \tag{5.4}$$

and $I[\cdot]$ is an indicator function with value 1 if its argument is true and 0 otherwise. Each indicator function, in turn, is a product of univariate step functions

$$H[\eta] = 1, \text{if } \eta > 0; \quad 0, \text{ otherwise}; \tag{5.5}$$

that describe each subregion R_j. Thus $B_j(\mathbf{x})$ is a basis function with value 1 if $\mathbf{x}$ is a member of the R_jth subregion of D. Also, for $\mathbf{x} \in R_j$, we have $\hat{f}(\mathbf{x}) = c_j$, the sample mean of the y_i's in the case of numerical y_i's and the most likely class in the case that the y_i's are categorical, whose $\{\mathbf{x}\}_{i=1}^N \in R_j$. Given these definitions the algorithm for recursive partitioning that results in a decision tree can be restated as follows.

$B_1(\mathbf{x}) \leftarrow 1$

$For\ M = 2\ to\ M_{max}\ do:\ lof^* \leftarrow \infty$

For $m = 1$ *to* $M - 1$ **do:**

For $v = 1$ *to* n **do:**

For $t \in \{x_{vj} | B_m(\mathbf{x}_j) > 0\}$

$g \leftarrow \sum_{i \neq m} a_i B_i(\mathbf{x}) + a_m B_i(\mathbf{x}) H[+(x_v - t)] + a_M B_i(\mathbf{x}) H[-(x_v - t)]$

$lof \leftarrow min_{a_1, \ldots a_M} LOF(g)$

if $lof < lof^*$, **then** $lof^* \leftarrow lof; m^* \leftarrow m; t^* \leftarrow t$ **end if**

end for

end for

end for

$B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[-(x_{v^*} - t^*)]$

$B_{m^*}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x}) H[+(x_{v^*} - t^*)]$

end for

end algorithm

Figure 5.1: The forward recursive partitioning or decision tree algorithm

In the first line of this algorithm the initial region is set to the entire domain of $\mathbf{x}$. The outermost "for loop" amounts to iterating through the splitting procedure with M_{max} the final number of regions. The three innermost loops select an optimal basis function B_{m^*}, a variable on which to split, x_{v^*}, and a split point t^*. The quantity being minimized is the lack-of-fit, or impurity, function that results form the chosen split.

The basis functions resulting from this algorithm are of the form

$$B_m(\mathbf{x}) \;=\; \prod_{k=1}^{K_m} H[s_{km} \cdot (x_v(k,m) - t_{km})] \qquad (5.6)$$

The quantity K_m is the number of splits that give rise to B_m. The quantities s_{km} take the values ± 1 and indicate the (right/left) sense of the split. The $v(k,m)$ are the predictor variables and the t_{km} are the corresponding values of these predictor variables.

Recall, of course, that after the initial tree is built, or in the current terminology the initial set of subregions established, a backward step merges subregions to avoid overfitting by the original tree.

5.3 The MARS Algorithm

The discontinuities in the above algorithm result from the step function $H[\eta]$ and the fact that $\hat{f}_j(\mathbf{x})$ has constant value c_j. To solve this problem, Friedman replaces the step function $H[\eta]$ with linear (order-1) regression splines in the form of left $(-)$ and right $(+)$ truncated splines, i.e.,

$$T^{-}(\mathbf{x}) = [(t-x)_+]^{q=1} = (t-x)_+ \text{ and } T^{+}(\mathbf{x}) = [(x-t)_+]^{q=1} = (x-t)_+, \quad (5.7)$$

where $u_+ = u$ if $u > 0$ and 0 otherwise, the scalar x is an element of $\mathbf{x}$ and t represents a partition point in the range of x. This choice was motivated by the observation that the original algorithm also uses truncated splines. These are hidden by the fact that in the above algorithm $q = 0$. Using these univariate splines, the resulting multivariate spline basis functions take the form:

$$B_m^{(q)}(\mathbf{x}) \;=\; \prod_{k=1}^{K_m}[s_{km} \cdot (s_{v(k,m)} - t_{km})]_+^q, \tag{5.8}$$

as well as products involving truncated splines with polynomials of lower order than q. Here $s_{km} = \pm 1$.

While the introduction of first order basis splines into the recursive partitioning algorithm solves the lack of continuity problem, it introduces a problem as well. Our original algorithm allows multiple partitions of a single variable along a single path in the tree. In the $q = 0$ case this poses no problem. For $q > 0$, this approach, however, introduces higher order dependencies amongst individual variables in the resulting basis functions. In other words, what results, are not q-order splines. In order to remedy this problem, one option, of course, is to disallow multiple partitions of the same variable in a single basis function (i.e tree path). This solution is unsatisfactory as it severely limits the power of our recursive partitioning algorithm. Before discussing the solution provided by Friedman we will discuss some other limitations of the recursive partitioning algorithm as the solution to the problem of higher order interactions of a single variable remedies these as well.

The other significant problem (besides lack of continuity) that limits the effectiveness of recursive partitioning algorithms is their inability to capture low order variable interactions. This deficiency recall, is due to the fact that recursive partitioning algorithms introduce ever higher order variable interactions with each iteration. To overcome this difficulty Friedman [41] proposes that the parent region not be eliminated during the creation of subregions. In terms of decision trees, this means that multiple splits are allowed on a given node. Of course, the result of multiple splits of a single node will result in *overlapping* subregions, which never occurred in our usual approach to building trees. With the repetitive partitioning of some region R_1 by different predictor variables, the modified recursive partitioning algorithm can produce linear models. It can produce additive models by always choosing $B_1(\mathbf{x})$, the entire domain, as the parent. This modification also allows for multiple partitions of the same predictor variable from the same parent region.

This resolution also solves the problem of higher order (other than $q = 1$) interactions of a single variable in the resulting basis functions. We can now safely disallow multiple splits of the same variable in a single basis function (a single path down the tree) without affecting the power of our procedure. In the cases that multiple splits of a single variable along a single path were used in our original tree formulation we can now accomplish the same effect by repeatedly selecting the same parent for splitting on the same variable. The resulting algorithm is as follows.

The **MARS** algorithm produces a linear truncated spline model with overlapping subregions $\{R_j\}_{j=1}^{S}$ of the domain D. Each overlapping subregion of a **MARS** model is defined by the partition points of the predictor variables from an ordered sequence of linear truncated splines that form a product basis function. The **MARS** approximation of the unknown function $f(\mathbf{x})$ is

$$\hat{f}(\mathbf{x}) \;=\; a_0 \;+\; \sum_{m=1}^{M} a_m \prod_{k=1}^{K_m} [s_{km} \cdot (x_{v(k,m)} - t_{km})]_+, \tag{5.9}$$

where $\hat{f}(\mathbf{x})$ is an additive function of the product basis functions associated with the subregions $\{R_j\}_{j=1}^{S}$. Since for a given set of product basis functions, the values of the partition points are fixed, the **MARS** model is a linear model whose coefficients may be determined by least-squares regression.

$B_1(\mathbf{x}) \leftarrow 1;\ M \leftarrow 2$

Loop until $M > M_{max}:\ lof^* \leftarrow \infty$

For $m = 1\ to\ M-1\ do:$

For $v \notin \{v(k,m)|1 \leq k \leq K_m\}$

For $t \in \{x_{vj}|B_m(\mathbf{x}_j) > 0\}$

$g \leftarrow \sum_{i=1}^{M-1} a_i B_i(\mathbf{x}) + a_M B_m(\mathbf{x})[+(x_v - t)]_+ + a_{M+1} B_m(\mathbf{x})[-(x_v - t)]_+$

$lof \leftarrow min_{a_1,\ldots,a_{M+1}} LOF(g)$

$if\ lof < lof^*,\ then\ lof^* \leftarrow lof;\ m^* \leftarrow m;\ v^* \leftarrow v; t^* \leftarrow t\ end\ if$

end for

end for

end for

$B_M(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x})[+(x_{v^*} - t^*)]_+$

$B_{M+1}(\mathbf{x}) \leftarrow B_{m^*}(\mathbf{x})[-(x_{v^*} - t^*)]_+$

$M \leftarrow M + 2$

end loop

end algorithm

Figure 5.2: The forward **MARS** algorithm

5.4 Categorical Predictors

As mentioned in the descussion of projection pursuit regression, in the previous chapter, the distinction between signal and noise is based solely on the notion of smoothness; $f(\mathbf{X})$ is assumed to be that component of $\mathbf{Y}$ that varies smoothly with changing values of $\mathbf{X}$, whereas the noise is taken to be the leftover part that does not. The effectiveness of a nonparamteric regression technique is determined by how well it can gauge the local smoothness properties of $f(\mathbf{X})$ and exploit them so as to filter out most of the noise without substantively loosing the signal.

In a setting where the predictor variables are orderable there are many definitions of smoothness resulting in many techniques for nonparametric function estimation. When some or all of the predictor variables assume values for which there is no natural ordering the notion of smoothness of the dependence of y on such variables is less clear. We now develop a notion of smoothness of the dependence of an ordinal variable on unorderable categorical variables and exploit it in the context of **MARS**.

Categorical values assume a discrete set of unorderable values

$$x \in \{c_1, \ldots, c_K\} \tag{5.10}$$

Consider the case of a single variable x that is categorical and the situation where one would like to estimate $f(x) = \mathsf{E}[\![y \mid x]\!]$. The simplest and unbiased estimate is

$$\hat{f}(x = c_k) = a_k = AVG(y \mid x = c_k) \tag{5.11}$$

with the average taken over the training data. These values are the least squares estimates of the coefficients in the basis function expansion

$$\hat{f}(x) = \sum_{k=1}^{K} a_k I(x = c_k) \tag{5.12}$$

where the basis functions are indicator variables of the categorical variable taking on each of its values[40].

In developing a meaningful definition of smoothness for categorical variables it is first useful to consider what we mean by smoothness in the case of orerable variables. Smoothness is defined as a relatively low variability of the function $f(x)$ amongst values of x that all lie within a local neighborhood of x. Smoothness of the dependence of a function on a categorical variable can be analogously defined, namely low variability of $f(x)$ when its argument is restricted to particular subsets of its values. A "smooth" function, $f(x)$, of a categorical variable x is a function whose values tend to cluster about a relatively small number of different values, as x ranges over its complete set of values. This definition of smoothness depends on the variability of $f(x)$ within such clusters but not between them. A categorical variable "smoothing" procedure would attempt to discover the particular subset of x values corresponding to each of the clusters and then produce as its function estimate the mean response within each one.

Let $A_1, \ldots, A_L$ be subsets of the set of values 5.10 realized by a categorical variable x

$$A_l \subset \{c_1, \ldots, c_K\}, \qquad 1 \leq l \leq L \tag{5.13}$$

and take as the function estimate the basis function expansion

$$\hat{f}(x) = \sum_{l=1}^{L} a_l I(x \in A_l), \qquad L \leq K \tag{5.14}$$

where the coefficients $\{a_l\}_1^L$ are estimated by least squares. If $L = K$ then this is equivalent to the unbiased estimate (5.12) assuming the subsets span all values of x. For $L < K$ bias has been introduced. For a given L the goal is to choose the subsets $A_1, \ldots, A_L$ that minimize model error over the training data. The value chosen for L is the one that minimizes future prediction error as estimated through some model selection criteria[40].

This procedure can be implemented in a similar fashion as is done for ordinal predictor variables with the indicator basis functions replacing the truncated power spline functions. One considers all basis functions of the form

$$I(x \in A) \tag{5.15}$$

where A ranges over all possible subsets of the categorical variables, as candidate variables to be selected through a variable subset selection procedure. The result of this variable selection procedure will be a model of the form 5.14 with the categorical value subsets $A_1, \ldots, A_L$ and their number L, automatically estimated from the data.

The ability of spline basis functions for ordinal variables and indicator functions over value subsets for categorical variables to delineate subsets of values for their respective type of variable: indicator functions do so directly, and spline functions through the knot locations, allows us to exploit our definition of smoothness in each setting in order to distinguish signal from noise.

Consider the case where there are n predictor variables $\mathbf{X} = (x_1, \ldots, x_n)$ all of which are categorical. Proceeding as in the ordinal case, a set of basis functions can be derived by taking the tensor product over all of the variables of the univariate indicator basis functions defined on each one

$$\{I(x_j \in A_{lj})\} \quad 1 \leq j \leq n. \tag{5.16}$$

[40]

An adaptive strategy would consider all of the basis functions in this complete tensor product as candidate variables. The **MARS** algorithm that approximates this strategy would be the same as the one described for ordinal variables with the replacement of truncated power spline basis functions with indicator functions over the categorical variable subsets

$$[+(x_v - t)]_+^q \leftarrow I(x_v \in A) \tag{5.17}$$

$$[-(x_v - t)]_+^q \leftarrow I(x_v \notin A) \tag{5.18}$$

The lack of fit of the resulting model is minimized with respect to l,v, and the subset A. Here indicator functions take the place of spline functions and categorical value subsets take the place of knot locations on the respective predictor variables. The rest of the procedure and model selection are the same. The resulting model has the form

$$\hat{f}(\mathbf{x}) = a_0 + \sum_{m=1}^{M} a_m \prod_{k=1}^{K_m} I(x_{v(k,m)} \in A_{km}) \tag{5.19}$$

[40]

Next consider the case of n predictor variables, n_o of which are ordinal and n_c of which are categorical. Spline basis functions are defined for each of the ordinal variables and subset indicator functions for each of the categorical variables. The tensor product of these functions over all of the variables forms a basis in the $n = n_o + n_c$ dimensional predictor space. These basis represent candidate variables for a variable subset selection strategy.

The **MARS** algorithm for mixed ordinal and categorical variables is a straightforward generalization of that for either all ordinal or all categorical varibles. Optimization with respect to the previous basis function $B_l(\mathbf{x})$, already in the model, is done in the same manner. The type of factor multiplying it will depend on the type of variable x_v that is being considered to serve as the factor argument: spline factor for ordinal variables or subset indicator functions for categorical variables. For a spline factor optimization is performed with respect to the knot location t. For an indicator factor it is performed with respect to the corresponding subset of categorical values. The resulting joint optimization with respect to the predictor variable x_v and the parameter of its corresponding factor will give rise to the best factor of either type to multiply $B_l(\mathbf{x})$, which itself may be a mixture of spline and indicator factors. The entire optimization over l, v, and t or A produces the next pair of basis functions to include at the $(M+1)$st iteration. As in the all ordinal or all categorical case, this forward stepwise procedure for generating basis functions is continued until a relatively large number M_{max} is produced.

5.5 Computational Issues

Our presentation of the **MARS** algorithm has been presented as a series of simple extensions to recursive partitioning regression. When addressing implementation issues, however, these "simple" extensions have a significant impact on the algorithmic complexity of **MARS**. A typical implementation of recursive partitioning regression takes advantage of the nature of step functions, as well as the fact that the resulting basis functions are nonoverlapping. These features of recursive partitioning regression can dramatically reduce the computation associated with the inner two loops of the decision tree algorithm 5.2 from $O(NM^2 + M^3)$ to $O(1)$. The total computation can then be made in $O(nNM_{max})$ time, after sorting. Unfortunately the same approach does not extend to the **MARS** algorithm.

The minimization of the lack of fit criterion in the **MARS** algorithm 5.3 is a linear least squares fit of the response y on the current basis function set. There are several techniques for performing this fit. We have selected an approach based on the Cholesky decomposition algorithm to solve the normal equations

$$\mathbf{B}^T\mathbf{B}\mathbf{a} \;=\; \mathbf{B}^T\mathbf{y} \tag{5.20}$$

for the vector of basis coefficients $\mathbf{a}$, where $\mathbf{B}$ is the basis data matrix and $\mathbf{y}$ is the (length N) vector of response values. While this approach is less numerically stable than some others, such as QR decomposition, unlike QR decomposition it allows us to keep the computation linear in the number of data records N, which is the largest parameter in the problem.

If the basis functions are centered to have zero mean, then the normal equations can be written

$$\mathbf{V}\mathbf{a} \;=\; \mathbf{c} \tag{5.21}$$

where

$$\mathbf{V}_{i,j} \;=\; \sum_{k=1}^{N} B_j(\mathbf{x}_k)\Big[B_i(\mathbf{x}_k) - \bar{B}_i\Big] \tag{5.22}$$

$$c_i \;=\; \sum_{k=1}^{N} (y_k - \bar{y})B_i(\mathbf{x}_k) \tag{5.23}$$

and $\bar{B}_i$ and $\bar{y}$ correspond to the averages over the data. These equations must be resolved for every possible knot location t, for every variable v, for m basis functions already included and for each of the M iterations of the **MARS** algorithm. If these computations are conducted in a naive manner, this will require computation proportional to

$$C \sim nNM_{max}^4(\alpha N + \beta M_{max})/L \tag{5.24}$$

where α and β are constants and L is the minimal number of required (nonzero weighted) observations between each knot point [1]. This computational burden would be prohibitive except in the case of very small data sets. Although performance improvements similar to those achieved in recursive partitioning regression are impossible, significant improvement is still possible. One can make use of properties of the $q = 1$ truncated power spline basis functions in order to develop rapid updating formulae for the quantities that enter into the normal equations (5.22), as well as to take advantage of the rapid updating properties of the Cholesky decomposition.

Theorem 4 Cholesky Decomposition. *If matrix $A \in \Re^{n\times n}$ is symmetric positive definite, then there exists a lower triangular matrix $G \in \Re^{n\times n}$ with positive diagonal entries such that $A = GG^T$.*

In solving the normal equations 5.21, if $\mathbf{V}$ is symmetric positive definite then we use the *Cholesky decomposition* to compute the matrix G. In the equation $\mathbf{V} = \mathbf{GG^T}$, for $i \geq k$ we have

$$a_{ik} = \sum_{p=1}^{k} g_{ip} g_{kp}. \tag{5.25}$$

Rearranging this equation we obtain

$$g_{ik} = \left(a_{ik} - \sum_{p=1}^{k-1} g_{ip} g_{kp} \right) \Big/ g_{kk} \qquad i > k \tag{5.26}$$

$$g_{kk} = \left(a_{kk} - \sum_{p=1}^{k-1} g_{kp}^2 \right)^{1/2} \tag{5.27}$$

[1] Freidman suggest imposing this requirement in order to avoid high variance of the function estimate at local minimum of the function in the presence of noisy observations.

We then first solve $\mathbf{Gy} = \mathbf{c}$ for $\mathbf{y}$, and then $\mathbf{G^T a} = \mathbf{y}$ for $\mathbf{a}$.

During the execution of the **MARS** algorithm it may be the case at an intermediate point that the matrix $\mathbf{V}$ becomes singular. In that case pivoting becomes necessary during the course of Cholesky decomposition. In order to avoid this pivoting step for efficiency reasons we slightly modify the normal equations via a small perturbation

$$(\mathbf{V} + \epsilon\mathbf{D})\mathbf{a} = \mathbf{C} \tag{5.28}$$

where $\mathbf{D}$ is a diagonal $(M+1) \times (M+1)$ matrix containing the diagonal elements of $\mathbf{V}$. This slight perturbation ensures that the modified version of $\mathbf{V}$ is always non-singular.

The most important property of the truncated power basis is that each basis function is characterized by a single knot point. Changing a knot location changes only one basis function, leaving the remaining basis functions unchanged. Other bases lack this appealing property.

The current **MARS** model in Figure 5.3 can be rexpressed as

$$g\prime \leftarrow \sum_{i=1}^{M-1} a_i B_i(\mathbf{x}) + a_M B_m(\mathbf{x}) x_v + a_{M+1} B_m(\mathbf{x})(x_v - t)_+ \tag{5.29}$$

The innermost "**For**" loop in the **MARS** algorithm 5.3 minimizes the lack of fit criterion with respect to both the knot location t and the coefficients $a_1, \ldots, a_{M+1}$. Using $g\prime$ instead of g yields an equivalent solution. The advantage of using $g\prime$ is that only a single basis function changes as t changes[41].

The updating formulae that we use assume that we visit the eligible knot locations in decreasing order, i.e $t \leq u$,

$$\begin{aligned}
c_{M+1}(t) &= c_{M+1}(u) + \sum_{t \leq x_{vk} < u} (y_k - \bar{y}) B_{mk}(x_{vk} - t) \\
&\quad + (u-t) \sum_{x_{vk} \geq u} (y_k - \bar{y}) B_{mk} \\
V_{i,M+1}(t) &= V_{i,M+1}(u) + \sum_{t \leq x_{vk} < u} (B_{ik} - \bar{B}_i) B_{mk}(x_{vk} - t) \\
&\quad + (u-t) \sum_{x_{vk} \geq u} (B_{ik} - \bar{B}_i) B_{mk}, \quad 1 \leq i \leq M \\
V_{M+1,M+1}(t) &= V_{M+1,M+1}(u) + \sum_{t \leq x_{vk} < u} B_{mk}^2 (x_{vk} - t)^2 \\
&\quad + (u-t) \sum_{x_{vk} \geq u} B_{mk}^2 (2x_{vk} - t - u) + (s^2(u) - s^2(t))/N
\end{aligned} \tag{5.30}$$

where $s(t) = \sum_{x_{vk} \geq t} V_{mk}(x_{vk} - t)$. B_{ik} and B_{mk} are elements of the basis function data matrix, x_{vk} are elements of the original data matrix, and y_k are the data response values[41].

These updating formulae can be used to obtain the last $(M+1)$st row (and column) of the basis covariance matrix $\mathbf{V}$ and last element of the vector $\mathbf{c}$ at all eligible knot locations t with computation proportional to $(M+2)N_m$. Here N_m is the number of observations for which $B_m(\mathbf{x}) > 0$. Note that all other elements of $\mathbf{V}$ and $\mathbf{c}$ do not change as the knot location t changes. This permits the use of updating formulae for the Cholesky decomposition to reduce its computation from $O(M^3)$ to $O(M^2)$ in solving the normal equations at each eligible knot location. Therefore, the computation required for the inner "**For**" loop of the **MARS** algorithm is proposrtional to $\alpha M N_m + \beta M^2 N_m / L$. This gives an upper bound on the total computation for the **MARS** algorithm as being proportional to

$$C^* = O(nNM_{max}^3(\alpha + \beta M_{max}/L)) \tag{5.31}$$

[41]

5.5.1 Categorical Variables

The principal computational issue in an implementation of the **MARS** algorithm centers on the minimization of the lack of fit criterion jointly with respect to all expansion coefficients and the parameters associated with the two new basis functions (knot locations or categorical value subset). Optimization with respect to the other parameters (l and v) is done through repeated applications of this minimization procedure. An important concern is that the computation increase only linearly with the training sample size N since this is generally the largest parameter of the problem. The case of optimizing with respect to a knot location has been previously discussed.

For a categorical variable x_v, the optimization is done jointly with respect to the expansion coefficients and subsets of its values

$$A^* = \mathrm{a}rgmin \sum_{i=1}^{N} \left[y_i - \sum_{m=0}^{2M} a_m \tilde{B}_m(\mathbf{x}_i) - a_{2M+1} B_l(\mathbf{x}_i) I(x_{vi} \in A) \right]^2 \tag{5.32}$$

[40] Here $\{\tilde{B}_m(\mathbf{x})\}_0^{2M}$ are an orthonormalized set of basis functions that span the same space as $\{B_m(\mathbf{x})\}_0^{2M}$. For a given subset A, minimization of (5.32) with respect to the coefficients $\{a_m\}_0^{2M+1}$ requires $O(MN)$ computational time. Once this optimization has been performed for one subset, it can be computed rapidly for any other subset with computation proportional only to M. This is because the minimum for any given subset A can be computed directly from the quantities

$$\sum_{i=1}^{N} y_i B_l(\mathbf{x}_i) I(x_{vi} = c_j) \text{ and } \sum_{i=1}^{N} \tilde{B}_m(\mathbf{x}_i) B_l(\mathbf{x}_i) I(x_{vi} = c_j), \quad 0 \le m \le 2M \tag{5.33}$$

These quantities can be evaluated once and for all at the beginning. Calculation of A^* by complete enumeration over all possible subsets would therefore require computation proportional to

$$M(N + 2^{K-1}) \tag{5.34}$$

For small K this does not present a serious problem. For substantially larger K, however, the associated exponential growth reduces the viability of this approach. An alternative approach is to employ an approximate stepwise variable subset selection procedure. The approximate subset selection procedure that we employed begins by selecting the categorical variable and single element subset of that variable that most improves the accuracy of our model. An additional value is repeatedly added to that subset so long as its addition improves model accuracy. While such a stepwise procedure does not necessarily produce an optimal subset, but rather may select a subset that provides a local minima in terms of model accuracy, this approach usually produces a reasonably good one. It is, however, not necessary that an optimal subset be found for any particular basis function since a suboptimal basis function can be remedied by basis functions added to the model in subsequent iterations.

Using a stepwise strategy in (5.32) reduces the computation to $O[M(N+K^2)]$ so that the total computation associated with the **MARS** algorithm is proportional to

$$O(M_{max}^3 \left[nN + \alpha \sum_{j=1}^{n_c} K_j^2 \right]) \tag{5.35}$$

where N is the sample size, n is the total number of predictor variables, M_{max} is the maximum number of basis functions produced by the forward stepwise algorithm, $\{K_j\}_1^{n_c}$ are the number of values associated with each of the n_c categorical variables, and α is a proportionality constant. Since in the pure ordinal variable case the computational time is $O(nNM_{max}^3)$, the additional computational burden associated with the introduction of categorical variables is small except for very large values of K_j[40].

Chapter 6

Bias, Variance and Shrinkage

6.1 Introduction

In solving a classification problem in which we are attempting to approximate a function f, the approach typically goes as follows: some family, F of functions is defined and an approximation, $\hat{f}$, of f is selected as the function in F having the minimum mean squared error over some training set T. If $\mathsf{E}[\![\hat{f}(x)]\!] = f(x)$, then $\hat{f}(x)$ is said to be an *unbiased estimator* of f and a measure of the precision of this estimator is $\mathsf{E}[\![\hat{f}(x) - f(x)]\!]^2$, i.e. its variance. If, instead, $E\hat{f}(x) \neq f(x)$, then $\hat{f}$ is known as a *biased estimator* of f. A measure of its precision is still $\mathsf{E}[\![\hat{f}(x) - f(x)]\!]^2$, but now since $E\hat{f}(x) \neq f(x)$, this quantity is not the variance, but rather is known as the mean squared error[1]. We now show[70],

$$\begin{aligned}
\mathsf{E}[\![[\hat{f}(x) - f(x)]^2]\!] &= \mathsf{E}[\![[\hat{f}(x) - \mathsf{E}[\![\hat{f}(x)]\!] + \mathsf{E}[\![\hat{f}(x)]\!] - f(x)]^2]\!] && (6.1)\\
&= \mathsf{E}[\![[\hat{f}(x) - \mathsf{E}[\![\hat{f}(x)]\!]]^2]\!] + (\mathsf{E}[\![\hat{f}(x)]\!] - f(x))^2 && (6.2)\\
&+ 2(\mathsf{E}[\![\hat{f}(x)]\!] - f(x))\mathsf{E}[\![\hat{f}(x) - \mathsf{E}[\![\hat{f}(x)]\!]]\!] && (6.3)\\
&= \mathsf{E}[\![[\hat{f}(x) - \mathsf{E}[\![\hat{f}(x)]\!]]^2]\!] + (\mathsf{E}[\![\hat{f}(x)]\!] - f(x))^2 && (6.4)\\
&= var[\hat{f}(x)] + (\mathsf{E}[\![\hat{f}(x)]\!] - f(x)])^2 && (6.5)\\
&= var[\hat{f}(x)] + [Bias(t)]^2 && (6.6)\\
& && (6.7)
\end{aligned}$$

If F is small, as in the case of ordinary linear least squares, for example, and the underlying function f is rather nonlinear, the bias will be large. However, because we are only required to estimate a small set of paramters due to the limited size of F, the variance will be small. On the other hand if F is large, as is the case when we use **MARS**, the bias is typically small while the variance is large. The bias/variance tradeoff is well known in the statistical community. The introduction of bias in a model in order to reduce variance and out of sample error is the topic of this chapter. Recall that the need for such techniques is particularly acute in our domains of interest due to their nonstationarity. The fact that we expect out of sample probability distributions to differ from their in sample counterparts implies

[1]In the literature, maximum likelihood estimators such as ordinary least squares (OLS), are often termed unbiased estimators. In fact, the OLS estimator will also be biased because we can not be certain that the model being used is in fact the correct model. Therefore, while the decision to use techniques such as shrinkage is often portrayed as a choice between biased and unbiased estimators, this is in fact, not the case.

that overfitting our model to in sample data by building a low bias, high variance model will have a particularly deleterious effect on our ability to minimize out of sample error.

Our selection of **MARS** as a modeling technique, while offering us the ability to approximate complex, nonlinear functions has the adverse effect of generating a function that requires the estimation of a large number of parameters (i.e. our function has many degrees of freedom). This results in the well known "curse of dimensionality". In order for it to fit well to the in sample data on which it is trained, the model is overfit to this data and has a large variance.

6.2 Bias, Variance, and Unstable Classifiers

The need to trade increased bias for decreased variance has been recognized in the statistics community for a long time. When a classifier is built based on a set of training data, what inevitably results is a model that is extremely effective at categorizing records taken from within the training data, but is less effective at categorizing out of sample data. Because the model is built by minimizing its mean squared error on the training data, what results is an overly complex model that overfits the data, inferring more structure in the data than is justified. Many classification techniques have an additional problem in extending their effectiveness from classifying in sample to classifying out of sample data - instability. Instability results when small changes to the training data result in significant changes to the model constructed. We discussed this issue in Chapter 5 when motivating the replacement of zero order splines with first order splines in the **MARS** algorithm as compared to the decision tree algorithm. While the use of first order splines mitigates this problem to a certain extent, it is still the case that relatively small changes in the training data can result in different basis functions to be selected, and therefore, a substantively different **MARS** model to be constructed.

In this chapter we begin by summarizing approaches used to reduce out of sample error when using decision tree classifiers. These approaches generally involve some mechanism for pruning the decision tree. Subtrees are replaced by leaf nodes, reducing the size and therefore, the number of degrees of freedom of the resulting tree. Which subtrees should be replaced is typically determined by some measure of how the removal of a given subtree impacts the overall predictive accuracy of the model. While in sample prediction error will be increased, model variance is decreased as is out of sample prediction error.

Next, we discuss the pruning approach suggested by Friedman for use with **MARS**. Qualitatively, the approach is similar to that used for pruning decision trees, the one significant difference being that when pruning decision trees, basis functions are removed in pairs [2]. In **MARS**, on the other hand, because overlapping basis functions are allowed, single basis functions may be (and are) pruned on each iteration of the pruning process.

We then survey two approaches to addressing classifier instability that additionally decrease model variance. The first of these is Bagging (**b**ootstrap **agg**regat**ing**). It will be shown that if, rather than building a model based on a single training set

[2] Basis functions are removed on pairs when pruning binary decision trees. When splits resulting in multiple children at a given node are allowed then all of these children will be pruned together. The general point is that with decision trees either all or none of a nodes children must be pruned.

of size N, we were provided with K training sets of size N, built K models, and then finally took the average over these models we would end up with a model with improved stability and decreased variance as compared to a single model based on a single training set. Bagging attempts to approximate this situation, in the case where we only have a single training set, by repeatedly sampling the training set in order to generate multiple training sets. Boosting is a similar technique to Bagging, differing only in the way the sampling is done. Boosting has been shown to be better at reducing out of sample error rates as compared with Bagging. It has the compuational disadvantage, however, that the K models built must be constructed in sequence as opposed to in Bagging where the models can be constructed in parallel. These notions will be made more precise in subsequnet sections of this chapter.

Finally, we discuss in some detail, the approach we used for reducing the variance of our classifier, namely Stein Shrinkage. We will show that the maximum likelihood estimator(MLE) is inadmissible by showing that the Stein estimator has uniformly lower risk than does the MLE in high dimensional (greater than 2) settings. Stein's results have often been termed the "Stein paradox", and have been widely overlooked by statistical practitioners in the years since its introduction. Stein shrinkage implies that when estimating one paramter θ_i we consider the values fo $\theta_j, j \neq i$. For example, if fifteen people each take an IQ test, when estimating the actual IQ of one of these individuals we consider the test scores of every other individual as well. This unintuitive result has generated much scepticism in the statistical community. We will discuss these issues, the merits of Stein Shrinkage, and the specific type of shrinkage that we employed in the final sections of this chapter.

6.3 Pruning Decision Trees

The recursive partitioning method of constructing decision trees continues to subdivide the set of training cases until each subset in the partition contains cases of a single class, or until no further partitioning offers any improvement. This approach will result in an optimal classification of the training data. It, however, typically results in an overly complex tree that overfits the data, inferring more structure in the data than is justified. This problem is illustrated in the following example given by Quinlan[66]:

Example 1 Consider the situation where we have a two-class task, in which a case's class is inherently indeterminate (i.e. unrelated to its attribute values), with proportion $p \geq 0.5$ of the cases belonging to the majority class, its expected error rate is clearly $1 - p$. If, on the other hand, the classifier assigns a case to the majority class with probability p and to the other class with probability $1 - p$, its expected error rate is the sum of

- the probability that a case belonging to the majority class is assigned to the other class, $p \times (1 - p)$, and

- the probability that a case belonging to the other class is assigned to the majority class, $(1 - p) \times p$

which comes to $2 \times p \times (1-p)$. Since p is at least 0.5, this is generally greater than $1-p$, so the second, more complex, classifier will have a higher misclassification rate. Quinlan constructed an artificial dataset of this kind with ten attributes, each of which took the value 0 or 1 with equal probability. The class was also binary, yes with probability 0.25 and no with probability 0.75. One thousand randomly generated cases were split into a training set of 500 and a test set of 500. From this data, C4.5's initial tree-building routine produces a nonsensical tree of 119 nodes that has an error rate of more than 35% on the test cases. Of course, a much simpler tree with just one node, would have only a 25% error rate.

Now that we see the problem with overfitting the question remains: how do we remove those parts of the tree that do not contribute to classification accuracy on unseen cases. I will describe two approaches here. The first, called error based pruning [66] uses only the training set to prune the tree. The second, cost-complexity pruning uses a set of holdout records to determine the final structure of the tree. This approach clearly has the drawback that more records are needed to build the final tree.

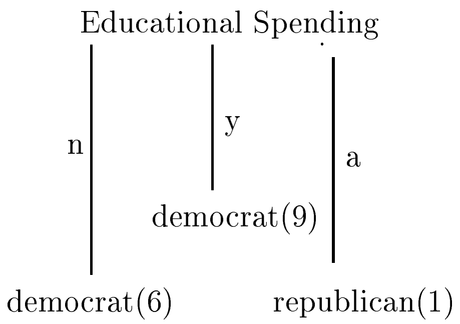
Pruning a decision tree will almost invariably cause it to misclassify more of the training cases. Consequently, the leaves of the pruned tree will not necessarily contain training cases from a single class. Instead, for each leaf there will be a class distribution specifying, for each class, the probability that a training case at the leaf belongs to that class. When considering test cases, we will classify these as being in the most likely class at their terminal leaf.

6.3.1 Error Based Pruning

Error based pruning is the method used by Quinlan in C4.5[66]. We start from the bottom of the tree and examine each non-leaf subtree. If replacement of this subtree with a leaf would lead to a lower predicted classification error rate on unseen cases we will prune the subtree in favor of a leaf. The question remains, how do we determine this error rate?

When N training cases are covered by a leaf, E of them incorrectly, the error rate for this leaf is E/N. If we regard these N cases as a sample of the entire dataset, we can ask what the probability of error would be of an arbitrary event that reaches this leaf. This probability can be determined for a given confidence level CF from the confidence limits for the binomial distribution. The upper limit of this interval will be written as $U_{CF}(E, N)$. Then, a leaf covering N training cases with a predicted error rate of $U_{CF}(E, N)$ would result in a predicted $N \times U_{CF}(E, N)$ errors on N unseen cases. We then can compare the sum of the expected number of errors in the children of a given node with the expected number of errors in the node itself if we pruned the children. If the error rate is improved then we will prune these children. It is important to note that we must consider expected errors of unseen cases as opposed to the number of errors in the existing tree because pruning will always result in more errors in the training set. The tree was built to optimally classify these cases. What we would like to do is better classify unseen cases.

Example 2 Consider the task of classifying congressman as democrats or republicans based on their voting records. Further, assume that on each issue they can vote yes(y), no(n), or abstain(a). Specifically, consider the vote taken on educational spending and the subtree resulting from partitioning the data on this attribute.



Since these nodes all contain single classes no further partitioning would occur - these nodes would be leaves of the tree. For the first leaf, $N = 6$, $E = 0$, and using a 25% confidence level $U_{25\%}(0, 6) = 0.206$, so the predicted number of errors if this leaf were used to classify 6 unseen cases is 6×0.206. For the other leaves, $U_{25\%}(0, 9) = 0.143$ and $U_{25\%}(0, 1) = 0.750$, so the number of predicted errors of this subtree is

$$6 \times 0.206 \ + \ 9 \times 0.143 \ + \ 1 \times 0.750 \ = \ 3.273 \tag{6.8}$$

If the subtree were replaced by a leaf (this would be the leaf democrat since 15/16 of the cases are democrat) N would be 16 and $E = 1$ (there would be one classification error for the one republican). Thus the expected number of errors on 16 unseen cases would be

$$16 \times \mathsf{U}_{25\%}(1, 16) \ = \ 16 \times 0.157 \ = \ 2.512 \tag{6.9}$$

Since $2.512 < 3.273$ this subtree would be pruned to a leaf and the process would continue up the tree until no further improvements are possible.

Note that this method violates statistical notions of sampling and confidence limits since the N (16 in the example above) cases being used as a "sample" are certainly not arbitrary events reaching that leaf. This heuristic has, however, produced good results.

6.3.2 Minimal Cost-Complexity Pruning

Minimal Cost-Complexity pruning is used by Breiman in CART[16].

Definition 11 *If T' is gotten from T by successively pruning off branches, the T' is called a* pruned subtree *of T and denoted by $T' < T$.*

Let T_{max} denote the unpruned tree.

The idea behind minimal cost-complexity pruning is this [16]:

Definition 12 *For any subtree $T \leq T_{max}$ define its complexity as $|\tilde{T}|$, the number of leaves in T. Let $\alpha \geq 0$ be a real number called the complexity parameter and let $R(T)$ be the classification error rate (i.e. the probability of misclassifying an arbitrary case). Then define the cost-complexity measure $R_\alpha(T)$ as*

$$R_\alpha(T) \;=\; R(T) \;+\; \alpha|\tilde{T}| \tag{6.10}$$

Thus, $R_\alpha(T)$ is a linear combination of the error rate of the tree and its complexity. If we think of α as the complexity cost per terminal node, $R_\alpha(T)$ is formed by adding to the error rate of the tree a penalty for complexity.

Now, for each value of α , the goal is to find the subtree $T(\alpha) \leq T_{max}$ which minimizes $R_\alpha(T)$. If α is small, the penalty for having a large number of terminal nodes is small and $T(\alpha)$ will be large. For instance, if T_{max} is so large that each terminal node contains only one case, then every case is classified correctly; $R(T_{max}) = 0$, so that T_{max} minimizes $R_0 T$. As α increases, the minimizing subtrees $T(\alpha)$ will have fewer terminal nodes. Finally, for α sufficiently large, the minimizing subtree $T(\alpha)$ will consist of the root node only, and the tree T_{max} will have been completely pruned.

Although α runs through a continuum of values, there are at most a finite number of subtrees of T_{max}. Because of the finiteness, what happens is that if $T(\alpha)$ is the minimizing tree for a given value of α, then it continues to be minimizing as α increases until a jump point α' is reached, and a new tree $T(\alpha')$ becomes minimizing and continues to be the minimizer until the next jump point α''. Thus, the pruning process produces a finite sequence of subtrees $T_1, T_2, \ldots$ with progressively fewer terminal nodes.

The task is now to choose one of those pruned trees (or T_{max}) as the final tree. This is done by using an additional set of test cases. Each of these cases is then run through each of the pruned trees and for each tree the misclassification error rate determined across the entire test sample. The tree with the minimal error rate will be chosen as the final tree.

This method has the disadvantage of requiring an additional set of cases to be set aside for the pruning phase. This problem can be mitigated by instead using cross validation, discussed earlier.

6.4 Pruning in MARS

As in ordinary recursive partitioning, **MARS** proceeds through two stages. First, an exhaustive set of partitions are constructed and then a backwards pruning procedure removes partitions that don't contribute to the predictive accuracy of the model. The significant difference between the pruning stage in **MARS** versus what we have seen before is that usually subregions are removed in pairs (assuming we are building a *binary* tree). In MARS, since, overlapping subregions are allowed, single subregions can be pruned at a time.

MARS uses residual-square-error in the forward and backward steps of the algorithm to evaluate model fit and compare partition points. The actual backward fit criterion used in **MARS** for final model selection is called the generalized cross-validation criterion (GCV)[41]. In general, cross validation requires many models to be built. Friedman proposes an approximation to the GCV criterion that only requires the construction of a single model. The modified generalized cross-validation criterion (GCV^*) used to evaluate a model with subregions $\{R_j\}_{j=1}^{M}$ is[41]

$$GCV^*(M) \;=\; \frac{1/N \sum_{i=1}^{N}[y_i - \hat{f}_M(\mathbf{x}_i)]^2}{[1 - C(M)^*/N]^2} \tag{6.11}$$

The numerator is the average residual-square-error and the denominator is a penalty term that reflects the model complexity.

If the values of the basis function parameters (number of factors K_m, knot locations t_{km} and signs s_{km}) associated with the MARS model were determined independently of the data response variables $(y_1, \ldots, y_N)$, then only the coefficients $(c_0, \ldots, c_M)$ are being fitted to the data. Consequently the complexity cost function is

$$C(M) \ = \ trace(\mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T) \ + \ 1, \tag{6.12}$$

where $\mathbf{B}$ is the $M \times N$ data matrix of the M (nonconstant) basis functions ($B_{ij} = B_i(\mathbf{x}_j)$).

The **MARS** procedure, however, does make use of the response values in the construction of basis functions. **MARS**, therefore, uses a modified version of the complexity cost function, $C(M)$, namely

$$\tilde{C}(M) \ = \ C(M) + d \cdot M \tag{6.13}$$

Here $C(M)$ is the number of nonconstant basis functions in the MARS model. The quantity d represents a cost for each basis function optimization and is a (smoothing) parameter of the procedure. Large values of d will result in fewer knots and, therefore, smoother functions. In practice, Friedman suggests that the best choice for values of d are $2 \leq d \leq 4$.

6.5 Bagging Predictors

Many classification techniques are unstable. Instability results when small changes to the training set result in significant changes to the model constructed. Decision trees suffer from this problem, as does **MARS** to a lesser extent. The instability of decision tress is mitigated, to a significant extent, in **MARS** as a result of the smoothing of the spline functions. In order to address the problem of instability, Breiman, has developed a technique known as bagging[14, 19] (**b**ootstrap **agg**regat**ing**). This technique, while addressing the instability problem, reduces the variance of the resulting predictive function.

In the typical setting for constructing a classification model we are presented with a training set T which consists of data $\{(y_n, \mathbf{x}_n), n = 1, \ldots N\}$ drawn randomly from a larger population (probability distribution) P, from which our out of sample data will also be drawn. Based on this training set we construct a model $\hat{f}(x)$ which approximates the true underlying function f. The average prediction error e of $\hat{f}(\mathbf{x})$ is

$$e \;=\; \mathsf{E}_\mathsf{T}\mathsf{E}_{\mathsf{Y},\mathbf{X}}[\![\mathsf{Y} - \hat{\mathsf{f}}(\mathbf{X})]\!]^2 \tag{6.14}$$

where $Y, \mathbf{X}$ are random variables taken from the distribution P and independent of T.

Imagine now, that instead of a single training set if we had K training sets each of size N and each drawn from the same distribution P. We then could construct a model $\hat{f}_i(\mathbf{x})$ based on each of these training sets and finally construct an *aggregated* model whose response variable is simply the average of the responses of each of the K models in the case of a numerical response variable, and whose response is the result of a plurality vote in the case of a categorical response variable. In this case the aggregated predictor is

$$\hat{f}_A(\mathbf{x}) \;=\; \mathsf{E}_\mathsf{T}\hat{\mathsf{f}}(\mathbf{x}) \tag{6.15}$$

and the average prediction error of the aggregated model is

$$e_A \;=\; \mathsf{E}_{\mathsf{Y},\mathbf{X}}(\mathsf{Y} - \hat{\mathsf{f}}_\mathsf{A}(\mathbf{X}))^2 \tag{6.16}$$

The important factor to note here is that in evaluating the average mean squared error we have moved the expectation over the training set T into our model construction procedure. In doing so, we can take advantage of the inequality $(\mathsf{EZ})^2 \leq \mathsf{EZ}^2$

which gives us

$$e = \mathsf{E}[\![Y]\!]^2 - 2\mathsf{E}[\![Y]\!]\hat{f}_A + \mathsf{E}_{\mathsf{Y},\mathbf{X}}\mathsf{E}_{\mathsf{T}}\hat{\mathsf{f}}^2(\mathbf{X}) \tag{6.17}$$

$$\geq \mathsf{E}[\![Y - \hat{f}_A]\!]^2 = e_A \tag{6.18}$$

Therefore, $\hat{f}_A$ has lower average prediction error than does $\hat{f}$. How much lower depends on how unequal the two sides of

$$\left(\mathsf{E}_{\mathsf{T}}[\![\hat{\mathsf{f}}(\mathbf{x})]\!]\right)^2 \leq \mathsf{E}_{\mathsf{T}}[\![\hat{\mathsf{f}}^2(\mathbf{x})]\!] \tag{6.19}$$

are. The effect of instability is clear. If $\hat{f}(\mathbf{x})$ does not change too much with replicate T the two sides will be nearly equal, and aggregation will not help. The more highly variable the $\hat{f}(\mathbf{x})$ are, the more improvement aggregation may produce. $\hat{f}_A$, however, is always an improvement over $\hat{f}$.

This analysis presumes the existance of K training sets which, of course, we don't have. In order to approximate this situation Breiman[14] has suggested the following approach. Repeated bootstrap samples $\{T^{(B)}\}$ are taken from T, and based on each of these samples a sequence of models, $\hat{f}_i(\mathbf{x})$ is constructed. If the response variable y is numerical then $\hat{f}_B$ is taken to be the average over the models constructed, and if y is categorical then the value of $\hat{f}_B$ is determined by a majority vote.

The $\{T^{(B)}\}$ form replicate data sets, each consisting of N cases, drawn at random, but with replacement, from T. Each $(y_n, \mathbf{x}_n)$ may appear several times or not at all in a particular $T^{(B)}$. The $\{T^{(B)}\}$ are replicate data sets drawn from the bootstrap distribution approximating the distribution, P, underlying T.

We return now to the discussion of error reduction as a result of this bootstrap aggregation procedure. We saw earlier the necessary error improvement that results if we have available a sequence of training sets drawn at random from the distribution P when we construct an aggregated model $\hat{f}_A$. The bagged estimate $\hat{f}_B$, however, is somewhat different in that it is constructed via a sequence of training sets drawn from the distribution P_T which concentrates mass $1/N$ at each point $(y_n, \mathbf{x}_n) \in T$. Then the quality of $\hat{f}_B$ is pushed in two directions: on the one hand, if the procedure that constructs each of the models, $\hat{f}_i(\mathbf{x})$, is unstable, then the bootstrap method can result in error reduction through aggregation. On the other hand, if the procedure is stable, then $\hat{f}_B(\mathbf{x})$ will not be as accurate for data drawn from P as $\hat{f}(\mathbf{x})$. There is a crossover point between instability and stability at which $\hat{f}_B$ stops improving on $\hat{f}$ and does worse. There is an additional limitation of bagging. For some data sets, $\hat{f}(\mathbf{x})$ is close to the limits of accuracy possible on the given data set. In that case, no amount of bagging can offer significant improvement.

6.6 Boosting Predictors

One limit to the practical application of staged model construction procedures like bagging (and boosting) is the computational requirements involved in the construction of many classification models. An appealing feature of bagging, in this context, is that the models are independent of each other and can, therefore, be constructed in parallel. The boosting technique, Adaboost[17, 18, 38, 39, 28] (adaptive boosting), discussed here, while having superior predictive capabilities as compared to bagging requires the construction of models in *sequence*. The parallelization of bagging is possible because in the construction of each subsequent model, each training example is equally likely to be selected as a member of a subsequent training set. In Adaboost, developed by Freund and Shapire, each training examples is used in the construction of subsequent models. They are weighted, however, based on the classification error of the given training example in the previous iteration of the model construction process. These notions are now made more precise.

Adaboost takes as input a training set $(\mathbf{x}_i, y_i), \ldots, (\mathbf{x}_m, y_m)$ taken from a probability distribution P. It is assumed for simplicity that each $y_i = \{-1, +1\}$. Adaboost invokes a given classification procedure in a series of rounds $t = 1, \ldots, T$. Initially, each training example is given an equal weight. In each subsequent round, training examples are reweighted so that examples that were misclassified in the previous round are given greater weight in the current round. In this manner, "hard" training examples are given ever increasing weights until a model is constructed capable of correctly classifying these examples. Finally, the aggregated model is the weighted average of the T models constructed, based on their average error rate. More precisely, the algorithm proceeds as follows[29]:

Initialize $D_1(i) = 1/m$ For $t = 1, \ldots, T$:

- Train model using distribution D_t
- Build model $\hat{f}_t : X \rightarrow \{-1, +1\}$ with error $\epsilon_t = \mathsf{Pr}[\![\hat{f}_t(x_i) \neq y_i]\!]$
- Choose $\alpha_t = \frac{1}{2} ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
- Update:

$$D_{t+1}(i) = \frac{D_t(i) exp(-\alpha_t y_i h_t(x_i))}{Z_t} \tag{6.20}$$

where Z_t is a normalization factor chosen so that D_{t+1} will be a distribution.

Output the final hypothesis:

$$H(x) = sign\Big(\sum_{t=1}^{T} \alpha_t h_t(x)\Big) \tag{6.21}$$

The algorithm maintains a distribution or set of weights over the training set. The weight of this distribution on training examples i on round t is denoted $D_t(i)$. Initially, all weights are set to be equal. On each round, the weights of incorrectly classified examples are increased so that the subsequent model is forced to focus on the hard examples in the training set.

The classification procedure's job is to construct a model $\hat{f}_t : \{-1, +1\}$ appropriate for the distribution D_t. The goodness of the constructed model is measured by its error

$$\epsilon_t = \mathsf{Pr}[\![\hat{f}_t(x_i) \neq y_i]\!] = \sum_{i:\hat{f}_t(X_i)\neq y_i} \tag{6.22}$$

Note that the error is measured with respect to the distribution D_t on which the classification procedure was trained.

Once the model $\hat{f}_t$ has been generated, Adaboost chooses a parameter α_t. Intuitively, α_t measures the weight that is assigned to $\hat{f}_t$ in the final aggregated model. Notice that $\alpha_t \geq 0$ if $\epsilon_t \leq 1/2$ and that α_t gets larger as ϵ_t gets smaller.

The distribution D_t is next updated. The effect of this rule is to increase the weight of examples misclassified by $\hat{f}_t$, and to decrease the weight of correctly classified examples. Therefore, the weight is concentrated on "hard" training examples.

The final model $\hat{f}$ is a weighted majority vote of the T models, $\hat{f}_t$ where α_t is the weight assigned to $\hat{f}_t$.

An important theoretical property of Adaboost is its ability to reduce in sample error. First, we recast the error ϵ_t of $\hat{f}_t$ as $\frac{1}{2} - \gamma_t$. Since a model that simply guesses each training example's class at random has an error rate of 1/2 (on binary problems), γ_t therefore measures how much better than random are $\hat{f}_t$'s predictions. Freund and Shapire[39, 39] prove that the fraction of mistakes on the training set of the final model $\hat{f}$ is at most

$$\begin{aligned} \prod_t [2\sqrt{\epsilon_t(1-\epsilon_t)}] &= \prod_t \sqrt{1-4\gamma_t^2} && (6.23) \\ &\leq exp(-2\sum_t \gamma_t^2) && (6.24) \end{aligned}$$

Therefore, if each model is slightly better than random so that $\gamma_t \geq \gamma$ for some $\gamma > 0$, then the training error drops exponentially fast.

For out of sample instances, Freund and Shapire show that the error, with high probability, is at most

$$\mathsf{Pr}[\![\hat{f}(x) \neq y]\!] + O\Big(\sqrt{\frac{Td}{m}}\Big) \tag{6.25}$$

where $\mathsf{Pr}[\![\cdot]\!]$ represents the empirical probability on the training sample. This bound suggests that boosting runs the risk of overfitting as T becomes large. Since the boosting procedure involves iterative stages of model construction eahc refined to better address the "hard" training examples, overfitting is a problem that one might anticipate with this approach. In fairness, Freund and Shapire have reported some excellent emerical results in terms of Adaboost's ability to reduce variance while avoiding overfitting even with large values of T.

6.7 Stein Shrinkage - Frequentist View

Having constructed one or more **MARS** models based on various training sets which we are provided with, we have now seen several approaches to trading increased bias for decreased variance in the interest of reducing overfitting, and, therefore, out of sample prediction error. We now turn to the approach that we have implemented known as *Stein Shrinkage.* Stein showed as early as 1956[75, 49] that in estimating the mean of a multivariate normal distribution with dimension $n \geq 3$, that the maximum likelihood estimator (MLE) is inadmissible and offered an alternative estimation procedure that is guaranteed to have uniformly lower mean squared error.

Stein's results have often been termed the "Stein paradox", and have been widely overlooked by statistical practitioners in the intervening years. The lack of use of these ideas is the result of several factors. First, statisticians have been widely content with the success of the MLE and have felt that the introduction of a process as complex as Stein shrinkage could not be worth the effort. More significant, however, is that fact that Stein shrinkage implies that when estimating one paramter θ_i we consider the values fo $\theta_j, j \neq i$. For example, if fifteen people each take an IQ test, when estimating the actual IQ of one of these individuals we consider the test scores of every other individual as well. This unintuitive result has generated much scepticism in the statistical community. We feel, however, that in the data mining community where high dimensional data is the norm, that these ideas deserve renewed consideration, and will provide a welcomed addition to the modeler's tool chest.

6.7.1 Motivation for Stein Shrinkage

Recall that at this point in our data mining algorithm we have constructed several **MARS** models, and estimated the associated paramters $\theta = (\theta_1, \ldots, \theta_p)$ by $\hat{\theta} =$

$(x_1, \ldots, x_p)$. If $\hat{\theta}$ is a good estimator of θ, then each of the x_i should be close to the associated θ_i. Additionally, $\sum_{i=1}^p \theta_i^2$ should be close to $\sum_{i=1}^p x_i^2$. What Stein actually observed is that $\sum_{i=1}^p \theta_i^2$ is close to $\sum_{i=1}^p x_i^2 - p$ with high probability. To address this problem, Stein suggests multiplying the MLE by a factor to ensure that not only is $\hat{\theta}$ a good estimator of θ, but also that $\sum_{i=1}^p \theta_i^2$ is close to $\sum_{i=1}^p x_i^2$ as follows

$$\hat{c} \;=\; 1 - \frac{p}{\sum_{i=1}^p x_i^2} \tag{6.26}$$

This seems reasonable because $\sum_{i=1}^p (1 - p/\sum_{i=1}^p x_i^2) x_i^2$ is close to $\sum_{i=1}^p \theta_i^2$ with a high probability.

More precisely,

Theorem 5 *Let $u(X)$ be a nonnegative function of the random variable X. If $\mathsf{E}[\![u(X)]\!]$ exists, then, for every positive constant c,*

$$\mathsf{Pr}[\![u(X) \geq c]\!] \;\leq\; \frac{\mathsf{E}[\![u(X)]\!]}{c} \tag{6.27}$$

Proof:[45]

We assume that X is continuous. This proof can be trivially extended to the discrete case by replacing integrals with sums.

Let $A = \{x : u(x) \geq c\}$ and let $f(x)$ denote the probability density function of X. Then,

$$\begin{aligned} \mathsf{E}[\![u(X)]\!] &= \int_{-\infty}^{\infty} u(x) f(x) dx & (6.28) \\ &= \int_A u(x) f(x) dx + \int_{A^*} u(x) f(x) dx & (6.29) \end{aligned}$$

Since each of these final terms is ≥ 0, we have

$$\mathsf{E}[\![u(X)]\!] \;\geq\; \int_A u(x) f(x) dx \tag{6.30}$$

Given that $x \in A$, $u(x) \geq c$, and we have

$$\mathsf{E}[\![u(X)]\!] \geq c\int_A f(x)dx \tag{6.31}$$

Since,

$$\int_A f(x)dx = \mathsf{Pr}[\![X \in A]\!] \tag{6.32}$$

$$= \mathsf{Pr}[\![u(X) \geq c]\!] \tag{6.33}$$

it follows that

$$\mathsf{E}[\![u(X)]\!] \geq c\mathsf{Pr}[\![u(X) \geq c]\!] \tag{6.34}$$

This is a general form of the well known Chebyshev inequality.

Theorem 6 Chebyshev's Inequality *Let X be a random variable with mean μ and variance σ^2. Then for every $k > 0$,*

$$\mathsf{Pr}[\![|X-\mu| \geq k\sigma]\!] \leq \frac{1}{k^2} \tag{6.35}$$

Proof:[45]

In the previous theorem let $u(X) = (X-\mu)^2$ and $c = k^2\sigma^2$. Then,

$$\mathsf{Pr}[\![(X-\mu)^2 \geq k^2\sigma^2]\!] \leq \frac{\mathsf{E}[\![(X-\mu)^2]\!]}{k^2\sigma^2} \tag{6.36}$$

Since $\mathsf{E}[\![(X-\mu)^2]\!] = \sigma^2$

$$\mathsf{Pr}[\![|X-\mu| \geq k\sigma]\!] \leq \frac{1}{k^2} \tag{6.37}$$

Corollary 1 *Let $\hat{\theta}$ be an unbiased estimator of θ. Then,*

$$\mathsf{Pr}[\![\left|\hat{\theta}-\theta\right| \geq \epsilon]\!] \leq \frac{Var(\hat{\theta})}{\epsilon^2} \tag{6.38}$$

Proof:

Let $\epsilon = k\sqrt{Var(\hat{\sigma})}$. Then this corollary follows directly from Chebyshev's inequality.

We now use these results to show that $\sum_{i=1}^{p} \theta_i^2$ and $\sum_{i=1}^{p} x_i^2 - p$ are close.

Theorem 7 *If $X_i, 1 \leq i \leq p$ are independent $N(\theta_i, 1)$ random variables then as $p \to \infty$, $\sum_{i=1}^{p} x_i^2/p$ converges in probability to $1 + \sum_{i=1}^{p} \theta_i^2/p$.*

Proof:[43]

Since $x_i \sim N(\theta_i, 1),\ 1 \leq i \leq p$,

$$\mathsf{E}\left[\left[\frac{1}{\mathsf{p}}\sum_{\mathsf{i=1}}^{\mathsf{p}} \mathsf{x}_{\mathsf{i}}^2\right]\right] = \frac{1}{p}\sum_{i=1} p\theta_i^2 + 1 \tag{6.39}$$

and

$$Var\left[\left[\frac{1}{p}\sum_{i=1}^{p} x_i^2\right]\right] = \frac{1}{p} \tag{6.40}$$

Then from Chebyshev's inequality,

$$\mathsf{Pr}\left[\left[\left|\frac{1}{\mathsf{p}}\sum_{\mathsf{i=1}}^{\mathsf{p}} \mathsf{x}_{\mathsf{i}}^2 - 1 - \frac{1}{\mathsf{p}}\sum_{\mathsf{i=1}}^{\mathsf{p}} \theta_{\mathsf{i}}^2\right| > \epsilon\right]\right] < \frac{1}{p\epsilon} \tag{6.41}$$

As $p \to \infty,\ 1/p\epsilon^2 \to 0$, and our result is proven.

An alternative motivation for Stein shrinkage based on least squares was offered by Stigler[76]. His argument proceeds as follows. The MLE is the result of performing least squares estimation of X on θ. Stigler arrives at the James-Stein estimator by doing what amounts to a Bayesian estimation, i.e. doing the inverse of the regression used to find the MLE, instead finding the least squares estimator of θ on X. Consider the situation where we are trying to find the best linear estimator of the θ_i of the form

$$\hat{\theta}_i = a + bX_i, \quad 1 \leq i \leq p \tag{6.42}$$

Then,

$$\hat{\theta}_i = \bar{\theta} + \hat{\beta}(x_i - \bar{x}) \tag{6.43}$$

where

$$\hat{\beta} = \frac{\sum_{i=1}^{p}(x_i - \bar{x})(\theta_i - \bar{\theta})}{\sum_{i=1}^{p}(x_i - \bar{x})^2} \tag{6.44}$$

We then observe that the numerator above has the same expected value as

$$\sum_{i=1}^{p}(x_i - \bar{x})^2 - (k-1) \tag{6.45}$$

Replacing this term in the expression, 6.44, for $\hat{\beta}$ we get

$$\begin{aligned} \hat{\beta} &= \frac{\sum_{i=1}^{p}(x_i - \bar{x})^2 - (p-1)}{\sum_{i=1}^{p}(x_i - \bar{x})^2} && (6.46) \\ &= 1 - \frac{(p-1)}{\sum_{i=1}^{p}(x_i - \bar{x})^2} && (6.47) \end{aligned}$$

The resulting least squares approximation is

$$\hat{\theta}_i = \bar{x} + \left[1 - \frac{(k-1)}{\sum_{i=1}^{p}(x_i - \bar{x})^2}\right](x_i - \bar{x}), \quad 1 \le i \le p \tag{6.48}$$

This result is a form of the James-Stein estimator where the estimator is centered at the mean.

6.7.2 The Inadmissibility of the MLE

Having now given two intuitive arguments why one might expect a Stein type estimator to be more accurate then the MLE, we now turn to a proof of this fact. We follow the argument of James and Stein to show that for $\theta = (\theta_1, \dots, \theta_p$, if $p \ge 3$, the MLE is inadmissible. We accomplish this by proving that the James

Stein estimator has uniformly smaller average mean squared error (risk) than does the MLE. Recall, that we assume that each θ_i has variance 1, and therefore, the MLE has risk 1 as well. We, therefore, prove the inadmissibility of the MLE by proving that the risk of the James Stein estimator is less than 1.

Theorem 8 *If $p \geq 3$ the MLE is inadmissible.*

Proof:[43]

The proof consists of computing the expected risk (R) of the James Stein estimator and showing that it is less than 1.

$$R = \frac{1}{p}\mathsf{E}[(\hat{\theta}-\theta)'(\hat{\theta}-\theta)] \tag{6.49}$$

$$= \frac{1}{p}\mathsf{E}[(\mathsf{x}-\theta)'(\mathsf{x}-\theta)] - \frac{2(\mathsf{p}-2)}{\mathsf{p}}\mathsf{E}\left[\frac{(\mathsf{x}-\theta)'\mathsf{x}}{\mathsf{x}'\mathsf{x}}\right] + \frac{(\mathsf{p}-2)^2}{\mathsf{p}}\mathsf{E}\left[\frac{1}{\mathsf{x}'\mathsf{x}}\right] \tag{6.50}$$

We now must show that $R < 1$. Observe that

$$\mathsf{E}\left[\frac{(\mathsf{x}-\theta)'\mathsf{x}}{\mathsf{x}'\mathsf{x}}\right] = \sum_{i=1}^{p}\mathsf{E}\left[\frac{(\mathsf{x_i}-\theta_\mathsf{i})\mathsf{x_i}}{\sum_{\mathsf{i}=1}^{\mathsf{p}}\mathsf{x_i^2}}\right] \tag{6.51}$$

$$= \int_{-\infty}^{\infty}\cdot\cdot\int_{-\infty}^{\infty}\frac{\sum_{i=1}^{p}(x_i-\theta_i)x_i e^{-\frac{1}{2}\sum_{i=1}^{p}(x_i-\theta_i)^2}}{\sum_{i=1}^{p}x_i^2}dx_1\ .\ .\ dx_p \tag{6.52}$$

For each summand in 6.52 rearrange the order of integration so that the first integration in evaluating the integral for the ith term is with respect to x_i. By using integration by parts we then get

$$\int_{-\infty}^{\infty}\frac{(x_i-\theta_i)x_i}{x'x}e^{-\frac{1}{2}(x_i-\theta_i)^2}dx_i = \int_{-\infty}^{\infty}(\frac{1}{x'x}-\frac{2x_i^2}{(x'x)^2})e^{-\frac{1}{2}(x_i-\theta_i)^2}dx_i \tag{6.53}$$

Integration of both sides of 6.53 with respect to the other $p-1$ variables and summing from 1 to p yields

$$\mathsf{E}\left[\frac{(\mathsf{x}-\theta)'\mathsf{x}}{\mathsf{x}'\mathsf{x}}\right] = (p-2)\mathsf{E}\left[\frac{1}{\mathsf{x}'\mathsf{x}}\right] \tag{6.54}$$

Substituting 6.54 into 6.50,

$$R = 1 - \frac{(p-2)^2}{p} \mathsf{E}\left[\frac{1}{\mathsf{x}'\mathsf{x}}\right] < 1 \tag{6.55}$$

6.8 Stein Shrinkage - Bayesian View

Bayesian Statistics is characterized by the assumption of a prior distribution for the parameters about which inferences are being made. Both the prior information and the data are obtained by sampling together with Bayes theorem are used to make inferences about the parameters of interest. When the prior distribution is unknown its parameters are often replaced by sample estimates. This technique is called empirical Bayes.

Bayes estimators are obtained as functions of prior parameters, population parameters and sample estimates. The posterior distribution is obtained by using Bayes theorem to combine the sampling distribution with the prior distribution to arrive at a posterior distribution. The Bayes estimator is optimal because it has the smallest mean squared error averaging over the prior distribution. For the squared error loss function the Bayes estimator is the mean of the posterior distribution.

When the prior distribution is unknown, first the Bayes estimator is obtained as if the prior were known. The unknown priors are then replaced by functions of sample estimates. The approximate Bayes estimator obtained by this technique is known as the empirical Bayes estimator.

6.8.1 The Bias Variance Tradeoff Revisited

Recall that in our current setting we have constructed a classification model of the form

$$B_m^{(q)}(\mathbf{x}) \;=\; \prod_{k=1}^{K_m}[s_{km}\cdot(s_{v(k,m)} - t_{km})]_+^q, \tag{6.56}$$

by replacing the univariate step function $H[\eta]$ of the decision tree algorithms with linear (order-1) regression splines in the form of left ($-$) and right ($+$) truncated splines, i.e.,

$$T^-(\mathbf{x}) = [(t-x)_+]^{q=1} = (t-x)_+ \text{ and } T^+(\mathbf{x}) = [(x-t)_+]^{q=1} = (x-t)_+, \tag{6.57}$$

where $u_+ = u$ if $u > 0$ and 0 otherwise, the scalar x is an element of $\mathbf{x}$ and t represents a partition point in the range of x.

Thus far we have considered the bias/variance issues in the context of regression rather than classification. While the arguments from the regression context extend rather naturally, given that the **MARS** model that we have constructed is a classification model, and that the problems we are interested in solving are classification problems, we now review the bias/variance tradeoff from that point of view.

In classification, the output variable $y \in \{1,\ldots,J\}$ is a class label. The training set T is of the form $T = \{(y_n, \mathbf{x}_n),\ n = 1,\ldots,N\}$ where the y_n are class labels. Given T, we have constructed a **MARS** classifier $M(\mathbf{x}, T)$ for predicting future class values of y. We assume that the training data consists of a set of independent records drawn from a distribution $P = (Y, \mathbf{X})$. The misclassification error is defined to be

$$ME(M(,T)) \;=\; \mathsf{E_P}[\![\mathsf{M}(\mathbf{X},\mathsf{T}) \neq \mathsf{Y}]\!] \tag{6.58}$$

and denote $ME(M)$ the expectation of $ME(M(,T))$ over T. Let

$$\begin{aligned}
\mathsf{Pr}[\![j|\mathbf{x}]\!] &= \mathsf{Pr}[\![Y = j|\mathbf{X} = \mathbf{x}]\!] && (6.59)\\
\mathsf{Pr}[\![\mathbf{dx}]\!] &= \mathsf{Pr}[\![\mathbf{X} \in \mathbf{dx}]\!] && (6.60)\\
\pi_j(\mathbf{x}) & \quad \text{is the prior distribution of the class } j \text{ over } \mathbf{x} && (6.61)
\end{aligned}$$

Then the Bayes classifier

$$C^* = argmax_j \mathsf{Pr}[\![j|\mathbf{x}]\!] \tag{6.62}$$

has a misclassification rate

$$\begin{aligned}
ME(C^*) &= 1 - \int max_j(\mathsf{Pr}[\![j|\mathbf{x}]\!])\mathsf{Pr}[\![\mathbf{dx}]\!] && (6.63)\\
&= 1 - \int max_j(\mathsf{Pr}[\![j|\mathbf{x}]\!]\pi_j(\mathbf{x}))\mathbf{dx} && (6.64)
\end{aligned}$$

This error rate can be broken into its component bias and variance. For each basis function (partition) in the **MARS** model

$$\begin{aligned}
ME(M_{B_l}) &= 1 - \sum_l max_j \mathsf{Pr}[\![x \in B_l, x \in j]\!] && (6.65)\\
&+ \sum_l I_{(y_l \neq y_l^*)} |\mathsf{Pr}[\![x \in B_l, x \in j]\!] - \mathsf{Pr}[\![x \in B_l, x \notin j]\!]| && (6.66)
\end{aligned}$$

where B_L is a given partition or basis function. The first term on the right hand side of the equality is the error due to bias, while the second term is the error due to variance. We can then observe that the bias for a given partition is

$$B(L) \sim \frac{C}{\sqrt[N]{L}} \tag{6.67}$$

and the variance for a given partition is

$$V(L) \sim \sqrt{\frac{L}{D}} \tag{6.68}$$

where C is a constant, N is the dimensionality of $\mathbf{x}$, L is the number of partitions in the **MARS** model, and D is the number of training data records. This shows very clearly the tradeoff that exists between bias and variance. As the model becomes more complex and its number of degrees of freedom is increased through the introduction of additional partitions with each iteration of the **MARS** algorithm, the bias decreases while the variance increases. We can reduce the variance (and in turn the out of sample error) by trading decreased variance for increased bias through a reduction in the number of degrees of freedom.

6.9 Stein Shrinkage as an Empirical Bayes Estimator

Having established the theoretical foundation of Stein shrinkage we now turn to our application of this apporach. In our setting, once the knot points (spline functions) of the **MARS** model have been established, the problem reduces to a linear regression problem. We, therefore, now look to apply Stein shrinkage in this context. Given a set of training data and a model,

$$y \;=\; \sum \theta_i B_i + \epsilon_i \tag{6.69}$$

we wish to estimate the parameters, θ_i. We assume priors on the θ_i,

$$\theta_i \;\sim\; N(0, \tau^2) \tag{6.70}$$

and let $\hat{\theta}_i$ be the unbiased estimator of θ

$$\hat{\theta}_i \;\sim\; N(\theta_i, \sigma^2) \tag{6.71}$$

We assume here that the $\hat{\theta}_i$ are independently distributed, and that the prior distribution is multivariate normal and the coordinates of the N dimensional vector

θ are independent. Therefore, the conditional distribution of $\hat{\theta}_i$ on θ_i is

$$f(\hat{\theta}_i|\theta_i) = \frac{1}{\sqrt{2\pi\tau}} e^{-\frac{1}{2}(\hat{\theta}_i-\theta_i)^2\tau^2}, \; 1 \le i \le N \tag{6.72}$$

and the prior distribution of the θ_i is

$$\pi(\theta_i) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{\theta_i^2}{2\tau^2}}, \; 1 \le i \le N \tag{6.73}$$

We now use Bayes theorem

$$f(\theta_i|\hat{\theta}_i) = \frac{f(\hat{\theta}_i|\theta_i)\pi(\theta_i)}{\int f(\hat{\theta}_i|\theta_i)\pi(\theta_i)d\theta_i} \tag{6.74}$$

in order to compute the posterior distribution of $\theta_i|\hat{\theta}_i$. We simply substitute 6.72 and 6.73 into 6.74, and obtain

$$f(\theta_i|\hat{\theta}_i) = \frac{1}{\sqrt{\frac{2\pi\tau^2}{\tau^2+1}}} e^{-\frac{(\theta_i - \frac{\tau^2\hat{\theta}_i}{\tau^2+1})^2}{\frac{2\tau^2}{\tau^2+1}}} \tag{6.75}$$

The Bayes estimator, $\hat{\theta}$ is the mean of the posterior distribution and can be expressed as

$$\hat{\theta}_i = \frac{\tau^2}{\tau^2+1}\hat{\theta}_i \tag{6.76}$$

$$= (1 - \frac{1}{\tau^2+1})\hat{\theta}_i \;\; 1 \le i \le N \tag{6.77}$$

This expression assumes that we know the variance of θ which is typically untrue. In these cases we simply estimate the variance from the data. We observe that

$$\frac{\hat{\theta}'\hat{\theta}}{\tau^2+1} \sim \chi^2(N) \tag{6.78}$$

and use this fact to compute the unbiased estimator of $\frac{1}{\tau^2+1}$,

$$\frac{1}{\tau^2+1} \sim \frac{(N-2)\sigma^2}{\hat{\theta}'\hat{\theta}} \tag{6.79}$$

resulting in the Stein estimator[30, 31, 32, 33]

$$\hat{\theta}_i = (1 - \frac{(N-2)\sigma^2}{\hat{\theta}'\hat{\theta}})\hat{\theta}_i, \quad 1 \leq i \leq N \tag{6.80}$$

Therefore, the Stein estimator may be viewed as an empirical Bayes estimator.

In assuming a prior mean of 0 for each of the θ_i the resulting Stein estimator shrinks each paramter towards 0. An important feature of the Stein estimator is that it performs significantly better than the MLE in only a relatively small subregion of the entire domain. For the Stein estimator to offer maximal improvement one must, therefore, construct an estimator designed to do well in this subregion. We accomplish this by shrinking the Stein estimator towards this desired region. For any θ_i that are substantially outside this region the Stein estimator will collapse towards $\hat{\theta}_i$ and, therefore, provide little advantage over the MLE. Stein noted this problem and suggested the following remedy which we have employed, namely to shrink towards the population mean.

Let $\Gamma_i = |\theta_i|$ and $\Gamma_1 < \Gamma_2 < \cdots < \Gamma_N$ and the truncated Stein estimator is then

$$\theta_i^{ts} = \left(1 - \frac{(L-2)\sigma^2 min\{1, \Gamma_L/|\theta_i|\}}{\sum_1^N \theta_j^2 \wedge \Gamma_L^2}\right)_+ \theta_i \tag{6.81}$$

This estimator provides a reasonable solution to the extreme θ_i problem, as is evident from the observation that $\sum_1^N \theta_j^2 \wedge \Gamma_L^2$ is fairly small even if $(N-L)$ of the θ_i deviate significantly from the mean. Dey and Berger[26] suggest that an appropriate choice for L is

$$L = [3 + 0.7(N-3)] \tag{6.82}$$

and this is the value we have used.

Chapter 7

Intrusion Detection and the 1999 DARPA Evaluation

As the number of networked computers grows and the amount of sensitive information available on them grows as well there is an increasing need to ensure the security of these systems. The security of computer networks is not a new issue. We have dealt with the need for security for a long time with such measures as passwords and encryption. These will always provide an important initial line of defense. However, given a clever and malicious individual these defenses can often be circumvented. Intrusion detection is therefore needed as another way to protect computer systems. While the construction of network intrusion detection systems has been an active research area in goverment, academia, and industry, the comparison of various approaches to this problem has been difficult. Intrusion detection systems are often built for specific networks with their own topologies using certain protocols and running certain programs. Additionally, the disemination of data about specific computer networks and activity on those networks has been limited. Network administrators are reluctant to make information about their networks widely available for fear that hackers will be able to identify and exploit weaknesses in them. To that end the Defense Department's Advanced Research Project Agency (DARPA) has initiated annual evaluations of intrusion detection systems. They have simulated a network based on actual activity on an Air Force base and made this data available to the network intrusion detection community. They have, additionally, simulated a wide variety of attack scenarios on this simulated network.

In this chapter we begin by reviewing the network intrusion detection problem and various approaches to detecting intrusions. We then describe the 1999 DARPA evalution and our exeriences while participating in it.

7.1 Approaches to Inrusion Detection

An intrusion can be defined as "any set of actions that attempt to compromise the integrity, confidentiality or availability of a resource". Researchers in the field of intrusion detection attempt to identify intrusions once the initial lines of defense have been breached and an unwarranted user has accessed the network. There are two fundamental approaches to this problem. The first, often called *misuse detection* attempts to categorize patterns of known intrusions as a means of identifying intrusions. This approach amounts to encoding these known patterns in some form and then solving the pattern matching problem of identifying these patterns in actual network use. This approach, of course, suffers from its inability to detect new types of intrusions. As networks evolve and sophisticated hackers continually attempt to identify and exploit new weaknesses in these networks, this approach will clearly not solve the problem of network intrusion detection in any comprehensive way.

The second approach looks at the other side of the coin. Rather than identifying patterns common in intrusions, it attempts to identify patterns in *normal* usage and identify deviations from these normal patterns. This approach is often called *deviation detection*. We have taken a hybrid approach, modeling both normal and abnormal behaviour within a computer network.

Although most computers in sensitive applications collect audit trails, these audit trails were traditionally established for performance measurement and offered little help in detecting intruders. Difficulties include the large quantity of audit information that is too detailed, voluminous, and often meaningless to a human reviewer. Also, such audit trails may omit information that is relevant to detecting intrusions. Nevertheless, they do provide information such as who ran what programs and when, what files were accessed, and how much memory and disk space were used, which is potentially useful for detecting intrusion attempts. To make audit trails useful for security purposes, automated tools are needed to analyze the audit data to assist in the detection of suspicious events.

There are several different levels on which an intrusion detection system may operate. Some systems like NIDES, analyze audit trails at the user level, developing statistical profiles of user behavior. As NIDES observes the behavior of each monitored user, it keeps statistics for each user, for each of several dozen intrusion detection measures. These statistics form a user's historical profile. The profiles are periodically updated based on observed user behavior. The hypothesis of such a system is that the short term behavior of a user, while they are engaged in some malicious act, will deviate sufficiently from their long-term normal behavior. NIDES identifies such statistical deviations as potential intrusions or attempted intrusions.

There are obvious difficulties in attempting to detect intrusions solely on the basis of departures from observed norms for individual users. Although some users may have well-established patterns of behavior, logging on and off at close to the same times every day and having characteristic levels and types of activity, others may display much more erratic behavior. For the latter type of user, almost anything is "normal", and a masquerader might easily go undetected. Moreover, the approach is vulnerable to defeat by an insider who knows that his behavior is being compared with his previously established behavior pattern and who slowly modifies their behavior over time, until they have established a new profile within which they can mount an attack. Finally, this approach will certainly be of no help in detecting malicious behavior of users for whom no profile exists such as users that entered a network via anonymous FTP or some similar service.

Alternatively, Forrest et al. [44] suggest that behavior at the level of privileged processes can be monitored. Privileged processes are those that have privileges over and above those granted normal users. Thus, we will be monitoring patterns exhibited by programs rather than people. There are several advantages to this approach:

- Vulnerabilities in privileged processes are most dangerous because exploitation of those vulnerabilities can give an intruder privileged user status.

- Privileged processes constitute a natural boundary for a computer, especially processes that listen to a particular port. The points of access into the system are controlled by privileged processes.

- The range of behaviors of privileged processes is limited compared to the range of behaviors of users. Privileged processes usually perform a specific, limited function, whereas users can carry out a wide variety of actions.

- The behavior of privileged processes is relatively stable over time, especially compared to user behavior. Not only do users perform a wider variety of actions, but the actions performed may change considerably over time, whereas the actions (or at least the functions) of privileged process usually do not vary much with time.

In their Intrusion Detection System (IDS) Forrest et al. define normal patterns in terms of OS system calls made by privileged processes. The algorithm they use to build a profile of normal behavior is extremely simple. They scan traces of system calls made by a particular privileged process, and build up a database of all unique sequences of a given length k, that occurred during the trace. Each program of interest would have its own database, which would be specific to a particular architecture, software version and configuration, local administrative policies, and usage patterns. Once a stable database is constructed for a given program, the database can be used to monitor the ongoing behavior of the processes executing that program.

Once this database of normal behavior has been constructed, the same method that was used generate the database is used to check new traces of behavior. All overlapping sequences of length k in the new trace are considered in order to determine of they are represented in the normal database. Sequences that do not occur in the normal database are considered to be mismatches. In order to determine the strength of an anomalous signal the difference between a mismatch and existing normal signals is measured. The similarity between two sequences can be computed using a matching rule. The matching rule used by Forrest is based on Hamming distance, i.e. the difference between two sequences i and j is indicated by the Hamming distance $d(i, j)$ between them. For each new sequence i, they determine the minimal Hamming distance $d_{min}(i)$ between it and the set of normal sequences:

$$d_{min}(i) \;=\; \min\{d(i,j) \text{ for all normal sequences } j\} \tag{7.1}$$

The d_{min} value represents the strength of the anomalous signal. Note that this measure is not dependent on trace length.

To detect an intrusion, at least one of the sequences generated by the intrusion must be classified as anomalous. In terms of Hamming distances they require that at least one of the sequences generated by the intrusion have $d_{min} > 0$, preferably $d_{min} >> 0$; the higher the d_{min} the more likely it is that the sequence was actually generated by an intrusion. In practice, they report the maximum signal in the trace, i.e. they compute the signal of the anomaly, S_A, as:

$$S_A \;=\; \max\{d_{min}(i) \text{ for all new sequences } i\} \tag{7.2}$$

Furthermore, in order to be able to compare S_A values for different values of k they normalize S_A over the sequence length k, i.e.:

$$\hat{S}_A \;=\; \frac{S_A}{k} \tag{7.3}$$

If the value of $\hat{S}_A$ is greater than some user defined threshold C then the trace is flagged as anomalous. The value of C will determine the number of false positives that are flagged. The larger C is the less likely a false positive will be flagged. Of course, if C is too large some real intrusions may be missed.

A similar approach is taken by Lee and Stolfo [54]. Rather than looking at simple sequences, however, they use RIPPER, a rule learning program, and apply it to training data. They formulate the learning task as follows:

- Each record has n positional attributes, $p_1, p_2, \ldots, p_n$, one for each of the system calls in a sequence of length n; plus a class label, "normal" or "abnormal".
- The training data is composed of normal sequences taken from 80% of the normal traces, plus the abnormal sequences from 4 traces that include intrusions.
- The testing data includes both normal and abnormal traces not used in the training data.

RIPPER outputs a set of if-then rules for the "minority" classes, and a default "true" rule for the remaining class. The following are examples of rules generated by RIPPER on the system call data:

$normal : -p_2 = 104, p_7 = 112$
Meaning: if p_2 is system call 104($vtimes$) and p_7 *is* 112 ($vtrace$) then the sequence is "normal"

$normal : -p_2 = 104, p_7 = 104$
Meaning: if p_2 is system call 104($vtimes$) and p_7 *is* 104($vtimes$) then the sequence is "normal"

$\vdots$

$abnormal : -true$

Meaning: if none of the above rules hold, the sequence is "abnormal"

The RIPPER rules can be used to predict whether a sequence is "abnormal" or "normal". Whether or not an abnormal sequence represents an intrusion depends, of course, on the accuracy of the rules. They again use a user defined threshold, C, so that if the number of abnormal sequences in a region of the trace exceeds C that region is deemed an intrusion. This differs from the approach used by Forrest in that here the specific locations of intrusions within a trace are identified. Since most of the sequences in a malicious trace will be normal with only brief deviations during the attempted attack they claim that this approach will be more sensitive to noise. A brief, attempted intrusion may be caught by this system since there will be a deviation from normal activity in some region of the trace. The same abnormal activity may not be caught by Forrest's IDS since the number of abnormal sequences as well as their Hamming distances from normal sequences may not be enough to exceed the minimum threshold.

The effectiveness of these approaches is difficult to evaluate due the lack of good, *real* data to test them on. First of all, the normal data used in both of their experiments was generated synthetically. That is, the privileged processes of interest were run in various modes over relatively long periods of time until the number of system call sequences stabilized. For an IDS that is deployed to protect a functioning system, it is not at all clear that this is the best way to generate normal sequences. The real normal behavior of a given process on a particular machine may be quite different from the synthetic normal. Additionally, in their experiments they run their algorithm with three traces of known intrusions. They then report the values of $\hat{S}_A$ for each of these runs with these values ranging from 0.2 to 0.7. They suggest that given these results, in a real environment, by setting the threshold C to some value in this range, their system will be effective at detecting intruders. While this may be true, it is certainly not definitive. How it would perform on a host of other intrusions is unclear. Also, it is not clear that these values of $\hat{S}_A$ suggest large enough deviations from normal to indicate intrusions. It is hard to evaluate whether these deviations based on synthetic data and run on a handful of known intrusion traces are representative of intrusions in general.

The system used by Lee and Stolfo is even more difficult to evaluate. Their system attempts to not only identify traces that include intrusions but also to pinpoint the regions in the traces in which the intrusions were attempted. In the data they used, however, while they knew the identity of intrusions they had no knowledge of the locations of these intrusions. It is therefore, impossible to evaluate whether or not they correctly identified intrusions.

7.2 The 1999 DARPA Network Intrusion Evaluation

The lack of quality and consistent training data is systemic in the intrusion detection community. Network administrators have been reluctant to make the necessary information available for obvious reasons. Recently, however, DARPA has begun sponsoring network intrusion evaluations prepared and managed by MIT Lincloln Labs in which a standardized set of audit data was collected on a simulated computer network. This data included both background netwrok traffic as well as a wide variety of attack scenarios.

7.2.1 The Simulation Network

The simulation network provided by DARPA consisted of two Ethernet network segments connected to each other through a router. One of these segments is "outside" the Air Force base LAN and the other is inside the LAN. The inside network is connected to one interface of the router and consists of all the computers that are part of the simulated local domain. The computers that model the "outside world" are connected to the external interface of the router.

The simulation network included eleven computers. The outside of the network contained a traffic generator for both background traffic and attack simuations, a web server, a sniffer, and two machines used for non-automated attack generation. The inside of the network consisted of a background traffic generator, a sniffer, a Solaris 2.5 victim, a SunOS 4.1.4 victim, a Linux 4.2 victim, a Linux 5.0 victim, and a Windows NT victim. Although one computer inside the LAN and on outside generated all of the background traffic in the simulation, a modification to these computer's operating systems allowed them to act as hundreds of "virtual" machines. The same modification was made to the outside web server so that it would be capable of simulating thousands of web servers.

7.2.2 Simulated Attack Scenarios

The approximately fifty attack scenarios implemented in the DARPA evaluation fell into four broad categories: denial of service, remote to user, user to root, and probing attacks [51].

- **Denial of Service:** A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of denial of service attacks. Some, such as a *mailbomb, neptune, or smurf* attacks abuse a legitimate system feature. Others like *teardrop or ping of death* create malformed packets that confuse the TCP/IP stack of the machine that is attempting to reconstruct the packet. Yet others such as *apache2, back*, and *syslogd* take advantage of bugs in a particular network daemon.

- **Remote to User:** A remote to user attack occurs when an attacker who has the ability to send packets to a machine over a network, but who does not have priveleges on that destination machine, exploits some vulnerability to gain local access as a user of that machine. Some of these attacks, such as *imap, named*, and *sendmail* exploit buffer overflows in network server software. Others such as *dictionary, ftp-write, guest*, and *xsnoop* attempt to exploit weak or misconfigured system security policies. Yet others such as *xlock* simply try to trick an authorized user into providing her password to a screensaver that is actually a trojan horse.

- **User to Root:** User to root attacks begin with a user who legitimately or otherwise has access to a normal user account on the network and then exploits some system vulnerability to gain root access to the system. User to root attacks are characterized by the fact that careful programming could eliminate all of the expolited vulnerabilities. These vulnerabilities are, however, present in every major operating system in release today. There are several types of user to root attacks. The most common is a buffer overflow attack. These occur when a program copies too much data into a static buffer without first checking to ensure that the data will fit. Through manipulation of this data overflow, an attacker can cause arbitrary commands to be executed by the operating system. Another class of user to root attack exploit programs that make assumptions about the environment in which they will be run. An example of this type of attack is *loadmodule*. The loadmodule program is used by the xnews window system under SunOS 4.1.x to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes the environment, unauthorized users can gain root access to the local machine. Other user to root attacks take advantage of programs that are not careful about the way in which they manage temporary files. Finally, some user to root vulnerabilities exist because of an exploitable race condition in the actions of a single program or multiple programs running simultaneously.

- In recent years, a growing number of programs have been released that are capable of automatically scanning a computer network to gather information or to find known vulnerabilities. These network probes are quite useful to an intruder who is planning to stage a future attack. At attacker with a map of the machines on a computer network as well as the services available on each machine can use this information to identify system weaknesses. Some of these scanning tools such as *satan, saint* and *mscan* enable even unskilled attackers to very quickly check hundreds or even thousands of machines on a network for known vulnerabilities.

In addition to varying the targets and types of attacks, stealth was also introduced into the attack scenarios in order to hide these attacks from an intrusion detection system. There are several ways that an attacker can accomplish this. Skilled attackers may attempt to cover their tracks by editing system logs or resetting the modification date on files they replaced or modified. Attackers can also distribute the stages of an attack over a long period of time in order to come in "under the radar" of an intrusion detection system looking for specific attack signatures.

7.2.3 The DARPA Data

The data provided by Lincoln Labs consisted of approximately 7 gigabytes of audit data collected by running the simulated Air Force network for twelve weeks. There were two types of audit data provided. First was *tcpdump* list files which included a record for each TCP/IP packet sent over the network. The number of records ranged between approximately 50 thousand and 2 million records per day over the twelve weeks depending on the level of network activity on the given day. These list files were sufficient for the detection of network based attacks and represented the training data that we focused on. The other type of audit data provided was BSM audit data from a single UNIX Solaris host internal to the network. This data was necessary for detecting attacks local to a specific machine (user to root attacks for example). Limited by time and the prepocessing necessary on all of the audit data, we did not mine BSM audit files and were, therefore, unable to detect these local attacks.

Each tcpdump list file was approximately 100 bytes long and consisted of the following attributes:

1. Start_Date (e.g. 01/27/1998)
2. Start_Time (e.g. 05:04:43)
3. Duration (e.g. 00:00:23)
4. Service (e.g. ftp)
5. Src_Bytes (e.g. 500)
6. Dest_Bytes (e.g. 500)
7. Src_Port (e.g. 1755)
8. Dest_Port (e.g. 21)
9. Src_IP (e.g. 192.168.0.20)
10. Dest_IP (e.g. 192.168.1.30)
11. Attack_Score (e.g. 2.56)
12. Attack_Type (e.g. ping-sweep)

In the training data all of these fields were filled in. In the test data the last two fields were filled in by the data mining system (actually Attack Type was optional). Systems are evaluate based on their ability to assign scores that are monotonically related to the posterior probability of an attack occurring in a session.

We used Bro, a program developed by Paxson[63] as a tool for reconstructing TCP/IP packets. We additionally took advantage of work done by Lee and Stolfo[55] in the 1998 DARPA evaluation in which they had extended Bro to handle ICMP packets as well as to augment the records with some additional features to be discussed shortly. Unlike *denial of service* and *probe* attacks, the *remote to local* and *user to root* attacks are not visible when simply looking at the packet records. These attacks typically involve a single TCP/IP connection and in order to detect them one must consider the actual contents of the packets. The additional features available as a result of the work by Lee and Stolfo, provided us with data specific information necessary for detecting these attack types.

7.2.4 Running the Algorithm

The problem of network intrusion detection involves analyzing computer network activity and attempting to identify the existence of intrusive or malicious behavior. In building a model of both normal and malicious activity we cannot assume that our data is stationary. It was known that attacks present in the training data would, in some cases, be modified when present in the test data. Additionally, some attacks that were not present at all in the training data were introduced in the test data. Therefore, we must account for these changes in building our model. Or more specifically, we must allow for these changes when using a model which was built based on the training data when trying to identify intrusions in the test data. Furthermore, each record in our data represented a single network connection. It was necessary that we analyze relationships between these records since intrusions often do not occur in a single network connection but rather are the result of a series of actions taken over many connections.

Our goal is to classify each augmented record as either 1) normal behavior, or 2) as part of a network intrusion. It is necessary to recognize that while many network activities may be benign in and of themselves, as part of a certain sequence of events they may represent malicious behavior. Using failed logins again as an example a single failed login may simply be the result of a user mistyping her password. A protracted sequence of failed logins, however, is probably the result of an intruder attempting to guess user passwords. Therefore, before we can begin to classify audit records as being normal or malicious we must first augment these records with contextual information regarding activities that preceded the given record. Once these features have been selected a classification model is built based on the training data. Finally, we employ shrinkage that addresses the issues of overfitting and non-stationarity, as well as offering us a mechanism for combining the classifiers we have built. The implementation of **MARS** and shrinkage are straightforward applications of the ideas discussed in previous chapters. The implementation of the initial pattern discovery phase, however, deserves some domain specific discussion.

Feature Extraction

As mentioned previously, the Bro program was used to reconstruct TCP/IP packets from the raw binary connection data provided by Lincoln Labs. These records were augmented with additional features that were generated as a result of the modifications made by Lee and Stolfo[55] to the Bro program. As discussed previously, these additional features are necessary for detecting *remote to local* and *user to root* attacks since attacks of these types typically occur in a single TCP/IP connection and in order to detect them one must consider the actual contents of the packets.

These additional features are as follows[55]:

- **hot** - The number of "hot" indicators which include things like access to system directories, creation and execution of programs, and others.
- **failed_logins** - The number of failed login attempts.
- **logged_in** - A bit indicating whether or not the login was successful.
- **compromised** - The number of compromised conditions.
- **root_shell** - A bit indicating whether or not a root shell was obtained.
- **su** - A bit indicating whether or not an "su root" command was attempted.
- **file_creations** - The number of files that were created.
- **shells** - The Number of shell prompts.
- **access_files** - The number of attempts that were made to create, edit or delete access control files.
- **outbound_commands** - The number of outbound commands in a FTP session.
- **hot_login** - A bit indicating whether or not the login belonged to a superuser on the system.
- **guest_login** - A bit indicating whether or not the login belonged to a guest of the system.

One of the nice features of our modeling approach is its ability to leverage successful techniques used by other practitioners. Our feature selection algorithm allows us to augment data records with as many primitive features as we like and then to allow the pattern discovery algorithm to determine which of these, or which combinations of these are actually most useful as **MARS** predictor variables. The final stage of our algorithm, shrinkage, has similar properties. Given a model constructed over a training set drawn from the same population as our own training set, whether a **MARS** model or some other model, we can combine the resulting predictor function with our own via shrinkage. This is done by considering the parameters of these other models as additional data points in approximating via shrinkage the appropriate "shrunk" paramter values. As we just saw we leveraged the work done by Lee and Stolfo[55] in identifying a set of connection data content features that are necessary for detecting *remote to local* and *user to root* attacks. They additionally observed that the following derived features are useful in identifying other types of intrusions. We have, therefore, taken their lead and incorporated them as well. These additional features are as follows:

- **count** - Given that the current record represents a connection to a specific IP address, A, this feature is a count of the number of connections to A in the two seconds preceeding the current connection.

- **syn_error%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same IP address as the current record, this feature represents the percentage of records that have a "syn" error.

- **rej_error%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same IP address as the current record, this feature represents the percentage of records that have a "rej" error.

- **same_service%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same IP address as the current record, this feature represents the percentage of records that connect to the same service as the current connection.

- **diff_service%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same IP address as the current record, this feature represents the percentage of records that connect to a different service than the current connection.

- **service_count** - Given that the current record represents a connection to a specific service S, this feature is a count of the number of connections in the two seconds preceeding the current connection that connect to the same service.

- **service_syn_error%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same service as the current record, this feature represents the percentage of records that have a "syn" error.

- **service_rej_error%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same service as the current record, this feature represents the percentage of records that have a "rej" error.

- **service_diff_ip%** - Amongst those connections in the two seconds preceeding the current connection which are connections to the same service as the current record, this feature represents the percentage of records that are connections to different IP addresses.

Before conducting the pattern discovery phase of our algorithm two additional modifications to the data were made. The first of these modifications involved augmenting each record with two additional fields. These fields are the subnet of both the source and destination IP addresses. For example, if the source IP address was 152.168.168.72 and the destination IP address was 192.168.153.89 then the source subnet field (SrcSubnet) and the destination subnet field (DstSubnet) would be 152.168.168 and 192.168.153 respectively. These fields were helpful in detecting various network probe attacks that sweep through all of the IP addresses on the target subnet. In these attacks one would observe a sequence of (typically ICMP) packets sent one after another to each of the IP addresses on a given subnet. Since the destination IP addresses in each of these successive connection records are not identical, without adding these additional features the relevent relationship between these records would not be identifiable.

The second modification that we made to the data involved how we represented the values of numerical attributes. Rather than using the feature's value as initially determined, we typically replaced this actual value with the number of standard deviations it was away from the mean amongst all records that used the same service. For example, we would compute the mean and variance of the values in the *syn_error%* field amongst all records that use the telnet service. We would then, in each telnet session, replace the *syn_error%* value with number of standard deviations it was from the computed mean. More precisely a value of 0 meant that it was within one standard deviation of the mean, a value of 1 meant that it was at least 1 standard deviation above the mean, a value of 2 meant that it was at least 2 standard deviations above the mean, and a value of 3 meant that it was at least 3 standard deviations above the mean. Similar meaning was given to the values -1,-2, and -3 with respect to values at least 1 standard deviation below the mean.

This modification was done for two reasons. The first was motivated by the limitation of our pattern discovery algorithm due to the fact that we are using a predicate rather than propositional temporal logic. Due to this limitation and without this modification, if our pattern discovery algorithm encountered a sequence of records with unusually large yet *unequal* values in their respective *syn_error%* fields, no pattern would be identified despite this (potentially) relevent sequence of connections. This is, of course, because our algorithm is limited to measuring equality with no notion of two values being "close". The relevent observation that we would have liked to capture in a discovered pattern would be that a sequence of records occurred, all with large values in their *syn_error%* fields. This notion of largeness(smallness) was captured by this modification. The reason that we had to calculate the mean and variance of numerical attributes only amongst records that used the same service is the inherent difference in the value one would expect of these fields between different services. Consider, for example, the *Src_Bytes* field which represents the number of bytes sent from the source to the destination in the current connection. If the current service is FTP then, since a file is being sent, one would expect this value to be frequently large. On the other hand, if the current service is telnet one would be surprised by a large value in this field since all that is being sent is a connection request, a username, and password. By computing the mean and variance of numerical attributes only amongst records that used the same service we were able to distinguish this difference.

The second reason for this modification was computational. In the next stage of our algorithm, the construction of the **MARS** model, when introducing a new basis function based on a predictor variable (field) V, the number of knot points that we must consider is determined by the number of distinct values of that variable present in our data set. By making this modification we have substantially reduced this number and, consequently, improved the efficiency of the **MARS** algorithm.

Having now developed all of the necessary primitive predictor variables we are now ready to run our pattern discovery algorithm. Recall that the training data consists of twelve weeks of simulated network activity, with one file provided for each day. Pattern discovery proceeded in three phases.

First, for each attack type, we identified those patterns most highly corrolated with an occurrence of the given attack type. For each attack type in turn, we discovered patterns in each data file in which the attack occurred. This was done by specifying an interestingness threshold T and identifying patterns whose interestingness measure exceeded T, i.e. $S1 = \{P | I(P) > T\}$ where P is a discovered pattern and $I(P)$ is P's interestingness measure [1]. In creating $S1$ the interestingness measure, I, of a pattern, P, was the ratio of the number of occurrences of P during the course of the current attack of interest to the total number of occurrences

[1] The value of T was typically around 3 as this value was empirically determined to be effective for the purpose of identifying enough interesting patterns. For some attack types, however, this value was adjusted when either too many (for practical purposes) or too few patterns were discovered using this threshold.

of P in the current data file.

The second set of patterns that we discovered were those that are least highly corrolated with an occurrence of any attack. We again considered each data file in turn and then, given a threshold, T (again a value of 3 was used) we discovered all patterns $S2 = \{P|I(P) < 1/T\}$ where, again, P is a discovered pattern and $I(P)$ is P's interestingness measure. These patterns represented patterns most highly representitive of normal behavior on the network. In creating $S2$ the interestingness measure, I, of a pattern, P, was the ratio of the number of occurrences of P during the course of *any* attack to the total number of occurrences of P in the current data file.

The final set of patterns that we included as predictor variables in the construction of the **MARS** model were intented to assist our intrusion detection system in indentifying novel attacks that did not appear in the training data. Intuitively, we needed to be able to identify deviations from normal behavior that were, nevertheless, not present in any attack we had already seen. To accomplish this we included the patterns $S3 = \{P|I(P)\neg P \in S2 \wedge P \notin S1\}$.

Having computed these sets $S1, S2$, and $S3$ for each data file we then had 180 sets of patterns, 3 for each of the 60 days of training data we were provided with. In order to select which patterns to use as predictor variables, we first had to merge these sets into three aggregated sets $S1, S2$, and $S3$. In addition to merging the 60 $S1$, 60 $S2$, and 60 $S3$ sets we also had to recompute the interesetingness measure of each discovered pattern. Since many of the discovered patterns did not occur at all or did not occur frequently enough to meet the threshold requirement in each of the data files, this required a linear scan through each file in order to count the total number of occurrences of each pattern as well as the number of occurrence of each pattern during the course of an attack, or in the case of the $S1$ patterns, during the course of a specific type of attack.

More precisely, for each pattern, P, present in any of the 60 $S1$ sets we counted the total number of occurrences of P as well as the number of occurrences of P during the course of the attack type with which P was associated. For each pattern, P, present in any of the 60 $S2$ sets we counted the total number of occurrences of P as well as the number of occurrences of P during the course of any attack. Patterns were stored as binary trees with temporal operators at the internal nodes and primitive features at the leaves. The locations of patterns in a given data file could then be identified by first counting the number of occurrences of and storing the locations of the various values taken by each of the primitive variables. The number of occurrences of a pattern could then be determined by recursively identifying the locations of their subpatterns. If the length of a given data file is D, the length of a pattern P is $|P|$ and the maximum number of occurrences of any primitive in P is M then it requires $O(DM|P|)$ computation to determine all of the locations of P in the given data file. Having determined all of the locations of P a linear scan of the data file is neessary to determine which of these occurrences are part of normal behavior on the network and which are part of intrusions and what type of intrusions they are part of.

Before the aggregation could occur, however, a postprocessing normalization of the discovered patterns was necessary. Following the aggregation, a postprocessing generalization was also required.

In order to motivate the need for the normalization procedure, consider the set of patterns, $S1$, that are found when discovering patterns present in the Ipsweep attack. An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines. An Ipsweep can be identified by many Ping packets destined for every possible machine on a network, all coming from the same source. An interesting Ipsweep pattern that our algorithm discovered was $\{\{service = SMTP\} \wedge \{SrcIP = IP_A\} \wedge \{DstSubnet = Subnet_A\} \mathsf{N} \{service = SMTP\} \wedge \{SrcIP = IP_A\} \wedge \{DstSubnet = Subnet_A\}\}$ where IP_A and $Subnet_A$ are specific IP addresses and subnets respectively. While this was the form that this pattern took in one data file, in another file it appeared as $\{\{service = SMTP\} \wedge \{SrcIP = IP_B\} \wedge \{DstSubnet = Subnet_B\} \mathsf{N} \{service = SMTP\} \wedge \{SrcIP = IP_B\} \wedge \{DstSubnet = Subnet_B\}\}$. These patterns are, of course, the same. They, however, appear to be different for the purpose of recalculating the interestingness of this pattern. More significantly, if, in the test data, this same attack is launched from a different $SrcIP$ we would be unable recognize the occurrence of this pattern because it would not exactly match any already discovered pattern. Our normalization procedure addresses this problem. The pattern discovered in the first file would be rewritten as $\{\{service = SMTP\} \wedge \{SrcIP = IP_1\} \wedge \{DstSubnet = Subnet_1\} \mathsf{N} \{service = SMTP\} \wedge \{SrcIP = IP_1\} \wedge \{DstSubnet = Subnet_1\}\}$. This representation means that the $SrcIP$, IP_1 was the first IP address encountered in an interesting pattern in the current file, with similar meaning given to $DstSubnet_1$. Since, in the course of an Ipsweep attack only a single $SrcIP$ and a single $DstSubnet$ will be involved we would not expect to see $SrcIP_i$ or $DstSubnet_j$ where $i, j > 1$. In other attacks, however, these might occur. Returning to our Ipsweep attack example, the interesting pattern found in the second file, namely, $\{\{service = SMTP\} \wedge \{SrcIP = IP_B\} \wedge \{DstSubnet = Subnet_B\} \mathsf{N} \{service = SMTP\} \wedge \{SrcIP = IP_B\} \wedge \{DstSubnet = Subnet_B\}\}$ would also be rewritten as $\{\{service = SMTP\} \wedge \{SrcIP = IP_1\} \wedge \{DstSubnet = Subnet_1\} \mathsf{N} \{service = SMTP\} \wedge \{SrcIP = IP_1\} \wedge \{DstSubnet = Subnet_1\}\}$, again since only a single $SrcIP$ and a single $DstSubnet$ will be present in the course of an Ipsweep attack. We have now normalized this pattern and will be able to identify its occurrence in all of the training sets as well as in the test sets. While we have an example of this normalization procedure in the context of attack patterns the same issues occurred and the same remedies were imposed on the normal patterns as well.

After having aggregated all of the discovered patterns into the sets $S1, S2$, and $S3$ we were still often left with more patterns than could be practically included in building our **MARS** model [2]. We were, therefore, limited to including only the most interesting patterns present in each attack in the set $S1$ and the ones least highly correlated with an attack from the set $S2$. The patterns chosen from $S3$ were implicitly determined by the definition of $S3$ and our selections from $S2$. The exact number of patterns included for each set varied. The interestingness measures of the patterns themselves typically suggested an appropriate point at which to stop including additional patterns.

There were many occasions where there were multiple patterns that were simply repetitions of each other or generalizations of each other all of which had large(small) interestingness measures. We attempted to appropriately select a sin-

[2]Recall that the algorithmic complexity of the **MARS** algorithm is proportional to the number of predictor variables. For computational reasons it was therefore important to limit this number.

gle pattern from each of these groups so that the patterns that we included in the **MARS** construction phase represented different features of the associated attack (or of normal behavior).

The simplest and most prevelent occurrence of repetitive patterns in the course of an attack were ones of the form $\{A\mathsf{N}B\mathsf{N}C\}$, $\{A\mathsf{N}B\mathsf{B_k}C\}$, $\{A\mathsf{B_k}B\mathsf{N}C\}$, and $\{A\mathsf{B_k}B\mathsf{B_k}C\}$. Since whenever a pattern over the operator N occurred, by definition, a pattern over the operator BK_k also occurred, we often found that when sequences of particularly relevent events were present in the data, patterns of all of these forms would be found interesting. Of course, in this case the patterns over the N operator typically had higher interestingness measures than those over the $\mathsf{B_k}$. Nevertheless, it was often the case that all of these patterns' interestingness measures exceeded T. In these cases we would include only the pattern $\{A\mathsf{B_k}B\mathsf{B_k}C\}$ in the construction of the **MARS** model. While this decision may be unintuitive since the pattern $\{A\mathsf{N}B\mathsf{N}C\}$ has a higher interestingness measure than the pattern $\{A\mathsf{B_k}B\mathsf{B_k}C\}$, we made the decision to include the latter rather than the former pattern because it is more *general*. That is, it was clear that whenever the pattern $\{A\mathsf{N}B\mathsf{N}C\}$ occurred, the pattern $\{A\mathsf{B_k}B\mathsf{B_k}C\}$ would also occur. The reverse is, however, not the case. In the interest of maximizing the chance of detecting an attack through the identification of an occurrence of one of these patterns we chose to include the most general one. A similar argument holds in selecting among reptitive patterns from the set $S2$. Again we always chose the most general pattern for inclusion in the set of **MARS** predictor variables.

Another example of the need for this generalization postprocessing is in the context of an ftp-write attack. The ftp-write attack is a remote to local attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and all of its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these permissions are misappropriated and any of these directories are not write protected, an intruder will be able to add files and eventually gain access to the system. These attacks can be identified by monitoring anonymous ftp sessions and ensuring that no files are created in the ftp root directory. Two patterns that were discovered during incidences of this attack were $\{\{root\_shell = 1\} \wedge \{guest\_login = 1\} \wedge \{file\_creations = 1\}\}$ and $\{\{root\_shell = 1\} \wedge \{guest\_login = 1\} \wedge \{file\_creations = 2\}\}$. Recall that a value of 1(2) in the $file\_creations$ field means that the number of files created exceed the mean number of files created, among all ftp sessions, by at least 1(2) standard deviations. Since whenever the pattern $\{\{root\_shell = 1\} \wedge \{guest\_login = 1\} \wedge \{file\_creations = 2\}\}$ occurred, the pattern $\{\{root\_shell = 1\} \wedge \{guest\_login = 1\} \wedge \{file\_creations = 1\}\}$ necessarily also occurred, we included only the latter pattern in the construction of the **MARS** model. In general, when two interesting patterns, P_1 and P_2, both occur during the execution of a specific attack type, where P_1 is a necessary consequence of (a generalization of) P_2 we include only the pattern P_1 in the construction of the **MARS** model. Having now completed the discovery of interesting patterns we are now ready to use these patterns along with all of the primitive record features in order to construct a **MARS** model based on each training set.

Building the MARS Model

Recall that building the **MARS** model involves solving the following normal equations

$$\mathbf{Va} = \mathbf{c} \tag{7.4}$$

where

$$\mathbf{V}_{i,j} = \sum_{k=1}^{N} B_j(\mathbf{x}_k)\Big[B_i(\mathbf{x}_k) - \bar{B}_i\Big] \tag{7.5}$$

$$c_i = \sum_{k=1}^{N} (y_k - \bar{y}) B_i(\mathbf{x}_k) \tag{7.6}$$

and $\bar{B}_i$ and $\bar{y}$ correspond to the averages over the data.

The solution to the normal equations was discussed in Chapter 5 where we used the *Cholesky decomposition* of V to compute the matrix G where $V = GG^T$. We then first solve $Gy = c$ for y, and then $G^T a = y$ for a. What remains to be done is the construction of the basis data matrix B (the averages $\hat{B}$ and $\hat{y}$ are trivially computed. Entry $B_{i,j}$ of the basis data matrix represents the value of the ith predictor variable in the jth data record. The precitor variables fall into two categories - the primitive features, and the temporal patterns. The values of the primitive predictor variables in each record is either part of the initial data record as returned by the Bro program or is generated via one of the postproccesses we run on those initial records. The value of $B_{i,j}$ where i is a temporal pattern is either 0 or 1 depending on whether or not the associated temporal pattern i is present in the record j. The location of a pattern, P, could be determined in exact analogy to the way we identified the locations of patterns when merging the $S1$ and $S2$ sets that resulted from our pattern discovery on each of the training files. When a pattern, P, where P is the ith predictor variable was found to occur in record j, $B_{i,j}$ was set to 1, otherwise it was set to 0.

A **MARS** model was constructed for each of the 60 data files that we had available to us. Given a connection record (along with all of the predictor variables) each model returned a value between 0 and 1 reflecting the probability that the given record was part of an intrusion. Each of the models constructed included a different set of basis functions since each model was built based on a different data file. For each set of tensor products of basis functions we then estimated the variable θ_i based on the records in each of the 59 data files other than the one that was originally used for the selection of the basis functions of model i. In this way, for each of the 60 sets of tensor products of basis functions, we had 60 approximations of the value of the multivariate variable $\theta_i,\ 1 \leq i \leq 60$.

Shrinking the Predictors

At this point, for each of 60 different sets of tensor products of basis functions, we have constructed 60 regression models. For each of the θ_i we therefore have 60 observations that are assumed to be independent and drawn from a multivariate normal distribution. As discussed in Chapter 6 we then used the following shrinkage estimator as our final approximation of each of the θ_i. Let $\Gamma_i = |\theta_i|$ and $\Gamma_1 < \Gamma_2 < \cdots < \Gamma_N$. The truncated Stein estimator is then

$$\theta_i^{ts} = \Big(1 - \frac{(L-2)\sigma^2 min\{1, \Gamma_L/|\theta_i|\}}{\sum_1^N \theta_j^2 \wedge \Gamma_L^2}\Big)_+ \theta_i \qquad (7.7)$$

We now have reduced 3600 classification models (60 for each of 60 tensor products of basis function sets) into 60 models. Each of these models returns a value between 0 and 1 reflecting the probability that a given record is part of an intrusion. We return the average of these 60 response variables as our final prediction.

Results

We report the results of our model, constructed based on the twelve weeks of training data provided to us by MIT Lincoln Labs. Lincoln Labs also provided us with two weeks of unlabeled (we were not told where attacks occurred). We were told after we had returned the results of our detection system where attacks and which attack types were present in the test data. In total there were roughly fifty attacks present about three quarters of which had appeared in the training data and about one quarter of which were novel attacks. Since we built our detection system based only on TCP/IP data, we were aware a priori that there would be a small subset of attack scenarios, namely those that were carried out locally on a single machine, that we would be unable to detect. Our detection rates are, therefore, reported only with respect to those attack types that we expected we could identify.

There are two important points to be noted about the scoring system used to evaluate our detection system. In the previous year's (1998) intrusion detection evaluation Lincoln Labs asked participants to score each test record with some number where the larger the score, the more likely the given record was part of an attack. Participants were then graded based on the corrolation of their scoring procedure with the actual presence of attacks. Using the 1998 evaluation as a starting point, we built our intrusion detection system to be capable of the same, record by record, scoring. In the evaluation in which we participated, however, we were instead asked to report the day and time at which an attack occurred and the duration of the attack. Our system had no precise way of identifying the start of an attack or its duration. This problem was particularly acute when attacks overlapped. We had no mechanism for recognizing the fact that there were two seperate attacks present in a sequence of highly scored (close to 1) records. It was also unclear what score threshold we should use for reporting the occurrence of an attack. If a test record received a score of 0.6 does this record deserve to be reported, what about a score of 0.8? Given no precise method for determining this threshold we used 0.75 as our threshold for reporting. Of course, using a smaller threshold would have resulted in the detection of more attacks but would have been accompanied by a higher false positive rate. Conversely, using a larger threshold would have resulted in the detection of fewer attacks but a smaller false positive rate. Given our limitation in being able to identify the start and duration of a specific attack, we rescored our intrusion detection system ourselves after Lincoln Labs distributed the entire attack schedule (which included starting times, durations, and types for each attack) using the following procedure. If our system gave *any* record that was part of an attack a score of greater than 0.75 we deemed that attack successfully detected. If our system gave a score of greater than 0.75 to a record that was not part of an attack we deemed that record a false positive. It is the results of this rescoring that we report here.

The second point to be noted involves the determination of the assigning of a specific attack type to an attack that has been detected. Our intrusion detection system provides no precise way of determining attack types. We, therefore, used the following heuristic to determine attack types. Once a record was deemed to be part of an attack we determined which tensor product of basis functions contributed most significantly to its high score. Amongst the basis functions in this tensor product, we looked for basis functions whose predictor variable, V, was a member of $S1$. We then reported the attack type to be that attack type that caused V to be included in $S1$ in the first place. If none of the predictor variables in the basis function tensor product were part of $S1$ we then used the second or third, etc. most influential tensor basis product until we found one that included a basis function from $S1$. If no such basis functions were found then the record's attack type was deemed unidentifiable. If there were multiple basis functions from $S1$ in the tensor product that was used for determining attack type, and these basis functions had been included in $S1$ due to their membership in *different* attack types, then the specific attack type for the given record was likewise deemed unidentifiable. When this occurred, however, it was typically the case that the different attack types that were suggested by the different basis functions were all members of the same class of attacks, i.e. *denial of service*, *probe*, *remote to local*, or *user to root*.

Novel attack types that were present in the test data but not in the training data, while potentially detectable, were, of course, not identifiable. Among attacks that we detected in the test data that had also been present in the training data, we were able to correctly label 61% of them with their specific attack types. 89% were labeled with an attack type in the correct class of attacks (but the wrong specific attack), or had been labeled as unidentifiable because of our inability to distinguish between multiple attack types that were part of the same, and the correct, attack class. Our detection rates for both novel attacks and attacks that were present in both the training and test data as well as false alarm rates are shown in Table 7.2.4. The detection rates are shown for three different score thresholds. These rates indicate the percentage of attacks detected by our system when an attack was identified by records whose score exceeded the associated score threshold. Of course as the score threshold increases both the detection rate and the false positive rate decrease.

Score Threshold	% of Old Attacks Detected	% of Novel Attacks Detected	% of False Positives
0.5	0.74	0.56	0.31
0.7	0.59	0.43	0.24
0.9	0.44	0.28	0.18

Table 7.1: Results from the DARPA Intrusion Detection Evaluation.

We began our involvment in the intrusion detection evaluation with little experience in this field. We were, of course, aided by our ability to leverage the work done by others in identifying appropriate features for inclusion when building our predictive models and hoped that this along with our data driven approach would allow us to build an effective intrusion detection system. We were gratified by our success in the course of this evaluation and feel that our results justify the merits of our approach. There were, however, several domain specific lessons that we learned, and modifications that we would make to improve our system in the future. We will mention a few of them here.

First, and foremost, would be the inclusion of additional, available audit data so that we would be able to detect the entire variety of attack types. Second, would be the introduction of a more formal method for selecting a score threshold for reporting the occurrence of an attack. This decision will probably always be impacted by a network administrator's threshold for false positives. The more a network administrator is willing to accept false positives the higher guarantee we can offer that we have detected all intrusions. The third improvement to our system would be the introduction of a more formal mechanism for determining attack types. The final significant modification that we would like to introduce regards the way in which we generated the set $S2$, i.e. the way in which we model patterns for normal behavior. Our system was more successful at detecting *probe* and *user to root* attacks than it was at detecting *denial of service* or *remote to local* attacks. This was because behavior during attack types that are part of the former two attack classes are much less variable than those in the latter two classes. *Probe* attacks all involve scanning a large number of hosts on a computer network. *User to root* attacks all involve a user gaining unauthorized root access to the system. The other two attack types, on the other hand involve the exploitation of vulnerabilities in a wide variety of system services. Therefore, unless we have modelled normal behavior for use of each system service we may be unable to identify deviations from this normal behavior. Since, in the training data, certain system services were utilized much more than others we found, in retrospect, that the patterns in $S2$ that modeled normal behavior did not include patterns involving some of the less frequently used services. To remedy this problem we would, in the future, develop the set $S2$ by discovering patterns on a service by service basis, therefore ensuring that patterns over the entire variety of normal system behavior are included in the construction or our intrusion detection model.

Chapter 8

Conclusions and Future Directions

We have developed a nonlinear procedure for modeling high dimensional, temporal, nonstationary data. Before summarizing our work in the context of network intrusion detection we will first discuss the potential application of our work to another equally timely and relevent area of research - cancer classification. While, on the surface, these areas seem completely unrelated we feel that there exist strong statistical similarities that we can exploit in applying our techniques in this important research area.

8.1 Cancer Classification

While medical professionals have become increasingly adept at identifying carcinogens as well as developing treatments for various classes of cancer the large variance in effectiveness of cancer treatment suggests a need for improved cancer classification. Consider, for example, the case of acute leukemias. It has been well established that acute leukemias can be categorized into those arising from lymphoid precursors (acute lymphoblastic leukemia, ALL) versus those arising from myeloid precursors (acute myeloid leukemias, AML). While this distinction has been well known for several decades and despite the fact that treatment efficacy of acute leukemias is highly correlated with proper classification, no single test yet exists to establish proper diagnosis. Rather cancer classification has been traditionally based on morphological appearance of the tumor. This approach has several clear limitations. First, it is dependant on the interpretation of a tumor's morphology by a hematopathologist. Therefore, two cancers with substantively different clinical courses and different responses to therapy may go undifferentiated because of their similar appearance under a microscope. Second, and more alarming, there are certainly many cancer classifications that are still unknown. As distinguished from cancer treatment, when infectious diseases are diagnosed and treated a sample of the infectious bacteria is first extracted and grown in vitro. Many different treatment options are then tested on this in vitro sample and the most effective one is then used in vivo on the patient. Doctors have observed a high correlation between the effectiveness of in vitro and in vivo treatment for infectious disease. In contrast, no similar correlation exists in the treatment of cancer. For many reasons many treatments that work well in vitro often do not translate into effective in vivo treatments, an example being blood flow to the tumor. Therefore, no pre-treatment tests are available for cancer patients in determining an appropriate treatment protocol. Instead, doctors rely on simple statistics in choosing treatment protocols often resulting in poor survival rates among cancer patients. If one considers, for example, the most prevalent of the solid tumor cancers namely lung, breast, and colon cancer the average remission rates range from roughly 30% - 40%. This data begs the question, what differentiates patients whose cancer is cured from those whose cancer is not? Or more appropriately, what distinguishes the cancers of those who are cured versus those that are not? We hypothesize that there exist unknown cancer classes. More specifically, within the group of all patients with breast cancer the reason only 30% are cured is that the existing chemotherapy treatments are only appropriate to the specific class of cancer that those 30% have, while the remaining 70% are suffering with an unknown and, therefore, untreated cancer class. By better classifying cancers it would be possible for medical researchers to devise more appropriate treatment plans.

The goal of cancer classification is, therefore, multifaceted. First, among known classes one would like to devise tests for the identification of these classes, thereby, taking this job out of the hands of hematopathologists. Second, one would like to discover novel classes so that researchers might be able to find new cancer treatments as well as target existing ones better.

Cancer is a disease in which cell growth and division are unregulated. Without regulation, the cells divide ceaselessly, piling on top of each other to form tumors. In all tumors something has gone wrong with the systems that control cell division. Decades of research have now firmly established that this loss of control is due to abnormal gene expression. This research has shown that cancer may be caused by mutant cellular oncogenes, by the inappropriate expression of normal cellular oncogenes that have been relocated in the genome by a chromosome rearrangement, or by the loss or inactivation of genes that suppress tumor formation.

Given that cancer results from changes in the DNA of healthy cells we propose an approach to cancer classification based on gene expression. We will address both the cancer classification problem as well the class discovery problem by identifying deviations in gene expression between healthy and cancerous cells. We will evaluate the quality of our approach to cancer classification by considering RNA samples from both healthy individuals as well as samples from patients from multiple known cancer classes as identified by their histopathological appearance. We attempt to accurately and consistently validate the diagnosis made by hematopathologists on genetic grounds. This is achieved by training our system (as described shortly) on RNA samples that are properly labeled by their cancer class (or as healthy). By discovering genetic differences among cancer classes we then build a predictive model of theses classes that can then be tested via cross validation and through testing on out of sample data. We then turn to the more challenging task of class discovery. Here we train our system on the same RNA samples. This time, however, these samples are unlabeled. We attempt to discover the classes associated with each sample without a priori knowledge of this information. Additionally, we attempt to discover novel classes within these samples. Our success at doing this will be measured by considering treatment effectiveness among cancer patients within these novel classes. If a newly discovered class is truly novel then patients within this class, given the same treatment protocol, should expect similar clinical results. We will discover these novel classes without a priori knowledge of patient treatment efficacy.

8.2 Conclusion

Broadly speaking, there are two kinds of patterns discovered by data mining algorithms: predictive and descriptive. Predictive algorithms are built to solve a specific problem of predicting one or more attributes in a database based on the rest. Predictive patterns are not always attempts at predicting 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 attributes. By contrast, the point of descriptive algorithms is simply to present interesting patterns that a domain expert might not already know. In other words, with descriptive patterns there are no right answers only interesting ones. Because of this, descriptive 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. Our work, the work described in this thesis, has focused on addressing problems in both of these areas, applied to a specific class of data namely, non-stationary, temporal, high dimensional data.

We addressed the nonstationarity issue by trading increased bias for decreased variance in the construction of our classification model. In some sense, we recognized a priori that, in the presence of nonstaionary data, any model that we constructed based on a set of training data would have limited predictive capabilities on out of sample (test) data. With this in mind we were willing to use the biased estimator that resulted from the application of truncated Stein shrinkage in order to reduce the variance in our model and improve its out of sample predictive power.

The most important nonstationary feature in the intrusion detection domain resulted from the introduction in the test data of attack types that had not been present in the training data. In addition to our use of shrinkage we, also addressed this type of nonstationarity in the way that we selected temporal patterns for inclusion in the construction of our **MARS** model. Recall that there were three sets of patterns that we included as features in the **MARS** model, $S1 = \{P|I(P) > T\}$, $S2 = \{P|I(P) < 1/T\}$, and $S3 = \{P|I(P)\neg P \in S2 \wedge P \notin S1\}$ where P is a discovered pattern and $I(P)$ is P's interestingness measure. We included this final set, $S3$, in recoginition of the nonstationary nature of attacks. It provided us with a set of features that, while not particularly relevent to improving the in sample predictive power of our algorithm, would improve the power of our model on out of sample data.

Our use of temporal logic in describing patterns of network behavior is novel (as best we know) in the intrusion detection community. As intrusion detection systems are increasingly deployed on sensitive computer networks, hackers will increasingly look for ways to improve the "stealthiness" of their attacks in order to avoid detection. Most intrusion detection systems involve considering patterns of behavior during some fixed time window. One obvious approach for hackers is, therefore, to spread the individual stages of an attack over a period of time that exceeds this window. Temporal logic offers a way to express temporal relationships withoutbeing tied to a specific clock. In this way intrusion detection systems can be made capable of recognizing patterns of behavior over arbitrarily long time intervals.

After having developed our feature set through the discovery of patterns over a linear time, propositional, temporal logic we then used these features, and others as the predictor variables in the construction of a **MARS** model. We included a survey of several widely used approaches to multivariate function approximation and then motivated our selection of **MARS** as a consequence of its robust, nonlinear predictive power, its computational tractability, and the interpretability of the resulting model.

One's ability to build predictive models of data is fundamentally dependent on two factors - the complexity of the underlying (true) model, and the quality of the available training data. Once a problem domain has been selected the complexity of the underlying model becomes an issue exogenous to the modeling process and therefore an issue over which the modeler has little impact except for ensuring that the techniques being used are robust enough to handle the domain's inherent complexities. The quality and size of the training data, while typically limited by real world constraints, must be addressed in a more proactive manner. As mentioned previously, nonstationarity is a qualitative feature of data that we address. Another important issue is the size of training data. The size of the training data is intimately related to the dimensionality of that data. In most complex domains the number of independent variables and thus the number of degrees of freedom is very large. Despite the availability of thousands or even millions of training records this results in poor coverage of the space being modeled which in turn results in high variance in the models constructed. Virtually all techniques for dealing with small sample statistics involve some final pruning phase where increased model bias is traded for decreased variance through a reduction in the number of degrees of freedom. We have surveyed many of these alternatives. We then show that the widely used maximum likelihood estimator is inadmissible by showing that the Stein estimator has uniformly lower risk in estimating the mean of a multivariate normal distribution whose dimension is greater than two. We use truncated shrinkage as our final estimator due to its favorable properties in estimating parameters far from the sample mean.

We then describe our experience in unifying all of these ideas in building an intrusion detection system for participation in the 1999 DARPA intrusion detection evaluation. As part of this evaluation we were provided with twelve weeks (~ 7 gigabytes) of training data and two weeks of test data. Our results are summarized in Chapter 7.

Bibliography

[1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. In *In Proc. of the conference on foundations of data organizations and algorithms (FODO)*, October 1993.

[2] R. Agrawal, T. Imielinski, and A. Swami. Database mining - a performance perspective. *Ieee Trans. On Knowledge And Data Engineering*, 5:914–925, 1993.

[3] R. Agrawal, T. Imielinsky, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD Conference*, pages 207–216, 1993.

[4] R. Agrawal, K-I Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *In Proc. of the 21st VLDB conference.*, 1995.

[5] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, chapter 12. AAAI Press, 1996.

[6] R. Agrawal, G. Psaila, E. Wimmers, and M. Zait. Querying shapes of histories. In *In Proc. of the 21st VLDB conference.*, 1995.

[7] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 487–499, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[8] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. of the International Conference on Data Engineering.*, March 1995.

[9] W.A. Ainsworth. *Speech recognition by machine.* P. Peregrinus Ltd., London, 1998.

[10] G. Berger and A. Tuzhilin. Discovering unexpected patterns in temporal data using temporal logic. In O. Etzion, S. Jajodia, and S. Sripada, editors, *Temporal Databases - Research and Practice.* Springer Verlag, 1998.

[11] D. Berndt. AX: Searching for database regularities using concept networks. In *Proceedings of the WITS Conference.*, 1995.

[12] D. J. Berndt and J. Clifford. Finding patterns in time series: A dynamic programming approach. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining.* AAAI Press/ The MIT Press, 1996.

[13] C. Bettini, X.S. Wang, and S. Jajodia. Testing complex temporal relation-

ships involving multiple granularities and its application to data mining. In *Proceedings of PODS Symposium*, 1996.

[14] L. Breiman. Bagging predictors. Technical Report 421, University of California Berkeley, September 1994.

[15] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. Classification and regression trees. Technical report, Wadsworth International, Monterey, CA, 1984.

[16] L. Breiman, J. H. Friedman, R.A. Olshen, and C.J. Stone. *Classification and Regression Trees*. Wadsworth Publishers, 1984.

[17] Leo Breiman. Bias, variance, and arcing classifiers. Technical Report 460, University of California at Berkeley, Statistics Department, April 1996.

[18] Leo Breiman. Arcing the edge. Technical Report 486, University of California at Berkeley, Statistics Department, June 1997.

[19] Leo Breiman. Using adaptive bagging to debias regressions. Technical Report 547, University of California at Berkeley, Statistics Department, February 1999.

[20] C. Carter and J. Catlett. Assessing credit card applications using machine learning. *IEEE Expert*, 2(3):71–79, 1987.

[21] B. Cestnik, I. Kononenko, and I. Bratko. ASSISTANT 86: a knowledge-elicitation tool for sophisticated users. In I. Bratko and N. Lavrač, editors, *Progress in Machine Learning*, pages 31–45, Wilmslow, 1987. Sigma Press.

[22] P. Cheeseman and J. Stutz. Bayesian classification (AutoClass): theory and results. In U. M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and R. Uthurusamy, editors, *Knowledge Discovery in Data Bases II*, chapter 6, pages 153–180. AAAI Press / The MIT Press, Menlo Park, CA, 1995.

[23] W. Cheney and D. Kincaid. *Numerical Mathematics and Computing*. Brooks/Cole Publishing Company, Pacific Grove, California, 1985.

[24] J. Clifford, V. Dhar, and A. Tuzhilin. Knowledge discovery from databases: The NYU project. Technical Report IS-95-12, Stern School of Business, New York University, December 1995.

[25] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.

[26] D.K. Dey and J. Berger. On truncation of shrinkage estimators in simultaneous estimation of normal means. *Journal of the American Statistical Association*, 78(384):865–869, December 1983.

[27] V. Dhar and A. Tuzhilin. Abstract-driven pattern discovery in databases. *IEEE Transactions on Knowledge and Data Engineering*, 5(6), 1993.

[28] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, pages 1–22, 1999.

[29] Harris Drucker. Improving regressors using boosting techniques. In Jr. D.H. Fischer, editor, *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 107–115. Morgan Kaufman, 1997.

[30] B. Efron and C. Morris. Limiting the risk of bayes and empirical bayes estimators - part i: The bayes case. *Journal of the American Statistical Association*, 66(336):807–815, December 1971.

[31] B. Efron and C. Morris. Limiting the risk of bayes and empirical bayes estimators - part ii: The empirical bayes case. *Journal of the American Statistical Association*, 67(337):130–139, March 1972.

[32] B. Efron and C. Morris. Stein's estimation rule and its competitors - an empirical bayes approach. *Journal of the American Statistical Association*, 68(341):117–130, March 1973.

[33] B. Efron and C. Morris. Data analysis using stein's estimator and its generalizations. *Journal of the American Statistical Association*, 70(350):311–313, June 1975.

[34] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *In Proceedings of the SIGMOD conference.*, 1994.

[35] U. M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. From data mining to knowledge discovery: An overview. In *Advances in Knowledge Discovery and Data Mining*, chapter 1. AAAI Press, 1996.

[36] U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* MII Press, Mento Park, 1996.

[37] W.J. Frawley, G. Piatetsky-Shapiro, and C.J. Matheus. Knowledge discovery in databases: an overview. In G. Piatetsky-Shapiro and W.J. Frawley, editors, *Knowledge Discovery in Databases*. AAAI / MIT Press, 1991.

[38] Y. Freund and R. Schapire. A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 15(5):771–780, 1999.

[39] Y. Freund and R. Shapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997.

[40] Jerome H. Friedman. Estimating functions of mixed ordinal and categorical variables using adaptive splines. In S. Morgenthaler, E. Ronchetti, and W.A. Stahel, editors, *New Directions in Statistical Data Analysis and Robustness*, pages 73–113. Birkhauser-Verlag, 1993.

[41] J.H. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–141, 1991.

[42] D.Q. Goldin and P.C. Kanellakis. On similarity queries for time-series data: constraint specification and implementation. In *In Proc. of the 1st Int'l Conference on the Principles and Practice of Constraint Programming. LNCS 976*, September 1995.

[43] Marvin H.J. Gruber. *Improving Efficiency by Shrinkage - The James-Stein and Ridge Regression Estimators.* Marcel Dekker, Inc., New York, New York, 1998.

[44] S. Hofmeyr, S. Forrest, and A. Somayaji. Lightweight intrusion detection for networked operating systems.

[45] R.V. Hogg and A.T. Craig. *Introdiction to Mathematical Statistics*. Prentice Hall, Upper Saddle River, New Jersey, 1995.

[46] M. Holsheimer and A. P. J. M. Siebes. Data mining: the search for knowledge in databases. In *346*, page 78. Centrum voor Wiskunde en Informatica (CWI), ISSN 0169-118X, January 31 1994. CS-R9406.

[47] R. Hyafil and R.L. Rivest. Constructing optimal binary trees is np-complete. *Information Processing Letters*, 5:15–17, 1976.

[48] T. Imielinski. A database perspective on knowledge discovery. Invited Talk at KDD'95, August 1995.

[49] W. James and C. Stein. Estimation with quadratic loss. In *Proceedings of the Fourth Berkeley Symposium on Mathematics and Statistics*, pages 361–379. University of California Press, 1961.

[50] G. H. John. *Enhancements to the Data Mining Process*. PhD thesis, Stanford University, 1997.

[51] K. Kendall. A database of computer attacks for the evaluation of intrusion detection systems. Master's thesis, Massachusetts Institute of Technology, 1999.

[52] B. Kero, L. Russell, S. Tsur, and W.M. Shen. An overview of database mining techniques. In *Proceedings of the Post-Conference Workshop on Knowledge Discovery in Deductive and Object-Oriented Databases*, Singapore, 1995.

[53] P. Laird. Identifying and using patterns in sequential data. In *Algorithmic Learning Theory, 4th International Workshop*, Berlin, 1993.

[54] W. Lee, S. Stolfo, and P. Chan. Learning patterns from unix process execution traces for intrusion detection. In *AAAI Workshop: AI Approaches to Fraud Detection and Risk Management*, July 1997.

[55] Wenke Lee. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems.* PhD thesis, Columbia University, 1999.

[56] J.B. Little and L. Rhodes. *Understanding Wall Street.* Liberty Publishing Company, Cockeysville, Maryland, 1978.

[57] H. Mannila. Data mining: Machine learning, statistics, and databases. In P. (Per) Svensson and J. C. (James Cornelius) French, editors, *Proceedings: Eighth International Conference on Scientific and Statistical Database Systems, June 18–20, 1996, Stockholm, Sweden*, pages 2–11, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1996. IEEE Computer Society Press.

[58] H. Mannila. Methods and problems in data mining. In Foto N. Afrati and Phokion Kolaitis, editors, *Database Theory—ICDT'97, 6th International Conference*, volume 1186 of *Lecture Notes in Computer Science*, pages 41–55, Delphi, Greece, 8–10 January 1997. Springer.

[59] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, August 1996.

[60] H. Mannila, H. Toivonen, and A. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal,Canada, August 1995.

[61] H. Mannila, H. Toivonen, and A. I. Verkamo. Efficient algorithms for discovering association rules. In U. M. Fayyad and R. Uthurusamy, editors, *AAAI Workshop on Knowledge Discovery in Databases (KDD-94)*, pages 181–192, Seattle, Washington, July 1994. AAAI Press.

[62] B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, Portland, Oregon, August 1996.

[63] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*, 1998.

[64] H.V. Poor. *An Introduction to signal detection and estimation.* Springer-Verlag, New York, 1988.

[65] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[66] J.R. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann, 1993.

[67] L.R. Rabiner and S.E. Levinson. *Isolated and connected word recognition: Theory and selected applications. In Readings in speech recognition.* Morgan Kaufmann Publishers, San Mateo, CA., 1990.

[68] Thomas P. Ryan. *Modern Regression Techniques*. John Wiley and Sons, Inc., New York, New York, 1997.

[69] S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3), 1997.

[70] A. Sen and M. Srivastava. *Regression Analysis - Theory, Methods, and Applications*. Springer-Verlag, New York, New York, 1990.

[71] P. Seshadri, M. Livny, and R. R. Design and implementation of sequence database system. In *Proceedings of ACM SIGMOD Conference*, 1996.

[72] A. Silberschatz and A. Tuzhilin. On subjective measures of interestingness in knowledge discovery. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, Montreal, Canada, August 1995.

[73] A. Silberschatz and A. Tuzhilin. What makes patterns interesting in knowledge discovery systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(6), December 1996.

[74] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the International Conference on Extending Database Technology*, 1996.

[75] Charles Stein. Inadmissability of the usual estimator for the mean of a multivariate normal distribution. In *Proceedings of the Third Berkeley Symposium on Mathematical Statistics and Probability*, pages 197–206. University of California Press, 1956.

[76] S.M. Stigler. The 1988 neyman memorial lecture: A galtonian perspective on shrinkage estimators. 5(1):147–155, 1990.

[77] J. van Leeuwen. *Handbook of Theoretical Computer Science: Volume B Formal Models and Semantics*. The MIT Press/Elsevier, MA, 1990.

[78] J. T.-L. Wang, G.-W. Chirn, T. G. Marr, B. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *Proceedings of ACM SIGMOD Conference on Management of Data*, 1994.

[79] L. Zadeh. The role of fuzzy logic in the management of uncertainty in expert systems. In *Fuzzy Sets and Systems, vol. 11*, pages 199–227. 1983.