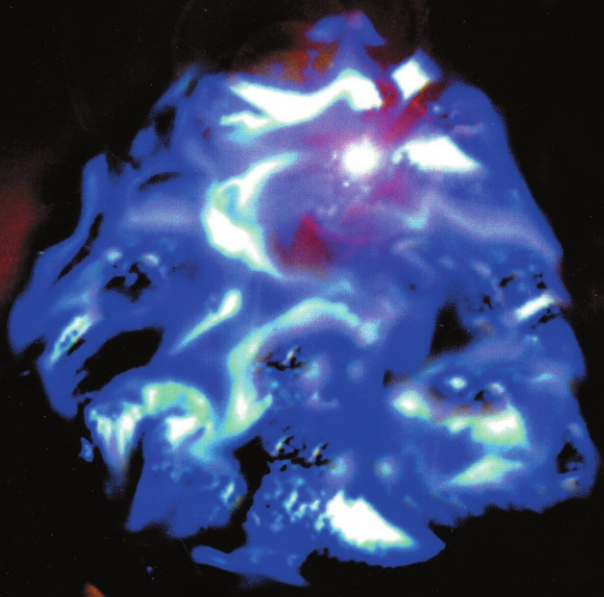
FEATURE SELECTION

FOR KNOWLEDGE DISCOVERY AND DATA MINING

HUAN LIU HIROSHI MOTODA



SPRINGER SCIENCE+BUSINESS MEDIA, LLC

FEATURE SELECTION FOR KNOWLEDGE DISCOVERY AND DATA MINING

THE KLUWER INTERNATIONAL SERIES IN ENGINEERING AND COMPUTER SCIENCE

Library of Congress Cataloging-in-Publication Data

Liu, Huan.
Feature selection for knowledge discovery and data mining / by Huan Liu and Hiroshi Motoda.
p. cm. -- (Kluwer international series in engineering and computer science ; 454)
Includes bibliographical references and index.
ISBN 978-1-4613-7604-0 ISBN 978-1-4615-5689-3 (eBook)
DOI 10.1007/978-1-4615-5689-3
1. Database management. 2. Data mining. I. Motoda, Hiroshi. II. Title. III. Series : Kluwer international series in engineering and computer science ; SECS 454.
QA76.9.D3L595 1998
006.3--dc21 98-25204
CIP

Originally published by Kluwer Academic Publishers, New York in 1998. Second Printing 2000.
Softcover reprint of the hardcover 1st edition 1998

Printed on acid-free paper.

FEATURE SELECTION FOR KNOWLEDGE DISCOVERY AND DATA MINING

by

Huan Liu
National University of Singapore
SINGAPORE

and

Hiroshi Motoda
Osaka University
Osaka, JAPAN

SPRINGER SCIENCE+BUSINESS MEDIA, LLC

Contents

List of Figures

List of Tables

Preface

As computer power grows and data collection technologies advance, a plethora of data is generated in almost every field where computers are used. The computer generated data should be analyzed by computers; without the aid of computing technologies, it is certain that huge amounts of data collected will not ever be examined, let alone be used to our advantages. Even with today's advanced computer technologies (e.g., machine learning and data mining systems), discovering knowledge from data can still be fiendishly hard due to the characteristics of the computer generated data. Taking its simplest form, raw data are represented in feature-values. The size of a dataset can be measured in two dimensions, number of features (N) and number of instances (P). Both N and P can be enormously large. This enormity may cause serious problems to many data mining systems.

Feature selection is one of the long existing methods that deal with these problems. Its objective is to select a minimal subset of features according to some reasonable criteria so that the original task can be achieved equally well, if not better. By choosing a minimal subset of features, irrelevant and redundant features are removed according to the criterion. When N is reduced, the data space shrinks and in a sense, the data set is now a better representative of the whole data population. If necessary, the reduction of N can also give rise to the reduction of P by eliminating duplicates. Simpler data can lead to more concise results and their better comprehensibility. Because of these advantages, feature selection has been the focus of interest for quite some time. Much work has been done from 70's to the present. With the creation of huge databases and the consequent requirements for good data mining programs, new problems arise and novel approaches to feature selection are in high demand. This is a perfect time to look back and see what have been done, and to look forward to the challenges ahead.

This book offers an overview of the various methods developed since 70's, provides a general framework in order to examine many methods and categorize them, employs simple examples to show the essence of representative feature selection methods, compares them using data sets with combinations of intrinsic properties according to the objective of feature selection, suggests guidelines how to use different methods under various circumstances, and points out some new challenges. This book consists of seven chapters, two appendices, and bibliographies after each chapter. Chapter 1 is about the background knowledge on data explosion, knowledge discovery from raw data, machine learning, and feature selection. Many fields related to these topics will be mentioned such as pattern recognition, statistics, visualization, database management systems, and so on. Chapter 2 describes perspectives of feature selection and explains how these different perspectives can be unified. Representative perspectives are pattern recognition, statistics, visualization, and machine learning. Chapter 3 derives a general framework after studying perspectives of feature selection. Chapter 4 starts with aspects of feature selection and illustrates representative feature selection methods to facilitate the reader in constructing his own feature selection method. Chapter 5 introduces the ways in which the methods should be evaluated, based on which empirical studies are performed. Experimental results are organized and presented in answering questions about the methods. It studies different characteristics of the data, and offers guidelines of applying feature selection methods under varied circumstances. Chapter 6 covers related topics such as feature extraction, feature construction, and feature discretization. Chapter 7 elaborates on the underlying philosophy throughout the book - *less is more* and concludes the book with prospects of feature selection in data mining. In the two appendices, we list the on-line sources for machine learning, data mining and knowledge discovery and provide descriptions of data sets and software used in the book. The soft copies of data sets and programs with instructions can be accessed from `http://www.iscs.nus.edu.sg/~liuh/Fsbook`.

This book can be considered as the first book for those, who start working on knowledge discovery but have not been exposed to feature selection, to understand the essence of feature selection and its various concepts. The book can be used by researchers in machine learning, data mining, and knowledge discovery as a toolbox in which they can find relevant tools that help in solving large real-world problems. The book can also be served as a reference book for those who are taking up tomorrow's challenges and conducting the research about feature selection.

To our families for their love,
support and understanding.

Acknowledgments

We are grateful to the editorial staff of Kluwer Academic Publishers, especially Scott Delman, Sharon Fletcher, and Robert Holland for their patience, interest, and helpfulness in bringing this project to a successful conclusion.

We have been fortunate to be assisted and encouraged by our colleagues and research associates who have contributed to this project in many ways. We are deeply indebted to them for their careful reading, thorough checking, and constructive suggestions. Although there is no space to thank them all, our special thanks to Manoranjan Dash, Farhad Hussain, Jian Shu, and Jun Yao for their invaluable assistance and extraordinary patience in this seemingly endless project, Kiansing Ng for helping design and complete the annotations in Appendices and maintaining the well constructed web sites, Rudy Setiono, Takashi Washio and Tadashi Horiuchi for fruitful collaboration without which the book would have been incomplete, Kaitat Foong and Kiansing Ng for their innovative cover design for this book.

Our families have been a constant source of encouragement throughout this project. Huan's greatest debt to his parents Baoyang Liu and Lihua Chen, and his family: Lan, Feitong, and Feige. Hiroshi's deepest gratitude to his family: Kay, Lena, Yuka and Eri.

1 DATA PROCESSING AND KNOWLEDGE DISCOVERY IN DATABASES

With advanced computer technologies and their omnipresent usage, data accumulates in a speed unmatchable by the human's capacity of data processing. To meet this growing challenge, the community of knowledge discovery from databases emerged not long ago. The key issue studied by the community is, in a layman's term, to make use of the data to our advantage. Or, why should we collect so much of it in the first place? In order to make the raw data useful, we need to represent it, process it, extract knowledge from it, present and understand knowledge for various applications. Through the first chapter, we provide the computational model of our study and the representation of data, and introduce the field of knowledge discovery from databases that evolves from many fields such as classification and clustering in statistics, pattern recognition, neural networks, machine learning, databases, exploratory data analysis, on-line analytical processing, optimization, high-performance and parallel computing, knowledge modeling, and data visualization. The ever advanced data processing technology and the increasing demand of taking advantages of data stored form a new challenge for data mining; one of the solutions to this new challenge is *feature selection* - the core of this study.

1.1 INDUCTIVE LEARNING FROM OBSERVATION

The study of feature selection finds its practical needs in machine learning in which a learning algorithm constructs a description of a function from a set of input/output instances through the interaction with the world. The learning model we adopt in this book is *inductive learning from observation*. We use a learning task for classification to explain the idea of induction from observation[1]. The observation is summarized in a data set which consists of input/output instances in terms of feature-values plus one feature describing classes. The learning task is to find a function (or classifier here) that generalizes from the data so that the classifier is able to predict the classes for new instances. Detailed examples of representative classifiers are to be discussed in the next chapter.

The problem of feature selection has long been studied in pattern recognition (Lampinen et al., 1998), in which the general model can be described as: learning some functions from patterns (also in feature-values, the same as instances) that are observations about the world so that new patterns can be recognized by the learned functions. In general, machine learning is more concerned about the non-continuous features, and pattern recognition more about continuous ones. As a matter of fact, the two get closer than ever in the context of knowledge discovery from databases. If we can handle both types of features, either model can be chosen, but some preprocessing may be required: discretization (Catlett, 1991) or introduction of dummy features (variables) (Mendenhall and Sincich, 1995). A general model of pattern recognition consists of three basic elements: observing element, learning element, and performance element (Watanabe, 1985). A thread connecting these elements is *features*.

1.1.1 Features

The choice of representation for the desired function is probably the most important issue facing the designer of a learning algorithm. Besides influencing the nature of the learning algorithm, it can determine whether the problem is solvable at all, or how powerful the language is in describing the problem for a learning algorithm to work on it. Many forms of representation are available, such as first-order logic, weighted networks. We choose the representation of features in our discussion because they are:

- *primitive* - they are the basic units for defining a problem, a domain, or a world to be observed, and do not require much effort from human experts to design them;
- *convenient* - they are easy to define, implement, and use;

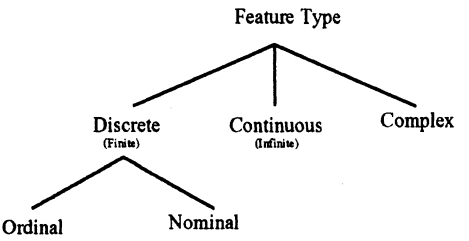


Figure 1.1. The hierarchy of feature types.

- *independent* - the use of features makes data acquisition decoupled from learning or classification;
- *widely used* - they are used in pattern recognition, machine learning, statistics, as well as databases; and
- yet still reasonably *general* - so that they are sufficiently powerful for many applications in knowledge discovery and data mining.

Features are also called attributes, properties, or characteristics. A collection of features with their values forms a flat data file that describes an application in which each line describes an instance (or pattern, case, example, record, tuple). Each feature can have discrete or continuous values, or be complex. The continuous features may have values from the domain of the real numbers which implies that the number of possible values is infinite. The discrete feature are variables with only a limited number of values. Discrete features can be further divided into ordinal (such as the winning places in a tournament) and nominal (such as names of colors). Values of a complex feature can be a structure as in C programming language, or a complex number ($x+iy$, for instance). The feature types are depicted in Figure 1.1 in a hierarchy. We deal only with discrete and continuous features in this book.

For ordinal features, their values can be ordered in just one direction or both directions. The distinction lies in whether the basic origin of the feature values is fixed by definition or it is left to our discretion. For example, in the example of winning places, we can have two schemes in representing the winning places. One scheme is that the first place is assigned value 1, the second place value 2, and the third place, value 3. The other scheme is that we represent the first place using 3, the second place 2, and the third place 1. As long as there is an order among the three values, either scheme will do. But some ordinal features do not have this flexibility, i.e., the way in which values are coded can be of only

one direction, such as *age*. It is unnatural to code *age* as the remaining years from death. Nominal features have no order among their values. The various values (green, blue, red, white, etc.) of nominal feature *color* have no ordinal relations among them. Another special type of features is binary features.

Are continuous and discrete features interchangeable? A simple answer is "yes", a long answer requires a section. In short, continuous features can be discretized or quantized into discrete ones; the only way to change nominal discrete features to continuous ones is via *binarization*. One way of binarizing a nominal feature is to turn it into a sparse binary vector. For a feature of three colors (red, green, and blue), we can code the feature into three binary features such as (1 0 0) is read, (0 1 0) is green, and (0 0 1) is blue. Now each new feature can be "considered" continuous, although it takes only two values, 0 or 1. In Chapter 6, we will discuss discretization and issues of feature discretization related to feature selection.

An example of feature-based data

Hair	Height	Weight	Lotion	Result
blonde	average	light	no	sunburned
blonde	tall	average	yes	none
brown	short	average	yes	none
blonde	short	average	no	sunburned
red	average	heavy	no	sunburned
brown	tall	heavy	no	none
brown	average	heavy	no	none
blonde	short	light	yes	none

Table 1.1. An example of feature-based data.

We use a dataset in (Winston, 1992) to explain what are features, different types of features, classes, and instances. The slightly modified dataset can be seen in Table 1.1. There are a total of five features, one of which is of class *Result*. The class feature has two values: sunburned or none. Each row in the table is an instance. All features are of discrete values. However, features Height and Weight can be readings of measurements. In that case, the two features are of continuous values. The discrete values of features *Height* and *Weight* can be the result of manual discretization of the measurements. For example, we can define an adult's height as *short* if the measurement is less than or equal to 1.5 meters, as *average* if it is greater than 1.5 meters and less than or equal to 1.8 meters, and as *tall* if it is greater than 1.8 meters.

1.1.2 Feature-based classification

We can now define instances based on a set of features. An instance is a pair $(\mathbf{x}, f(\mathbf{x}))$, where $\mathbf{x}$ is the input and is defined by an N-dimensional feature vector where features could be discrete or continuous, $f(\mathbf{x})$ outputs one of the predefined classes (categories). We name this particular feature as the class feature which is usually discrete if not specified. The example data shown in Table 1.1 consists of 8 instances with 5 columns (4 features plus one class feature).

Feature-based classification can be of forms of lazy or eager learning (Aha, 1997). Lazy learning algorithms are often called case-based, instance-based, or memory-based. One example is k-nearest neighbor algorithms. Lazy learning is so named because this type of algorithms does not generalize from the data when the data is available; instead, it delays the learning until it is necessary - when it needs to classify an unseen instance. The classification model of eager learning, however, induces from the data a concept description (be it a rule set, decision tree, or neural network) which is usually much more compressed compared with the original data. Nevertheless, feature selection methods discussed in this book can be helpful for both types of learning, although we mainly draw examples from eager learning (Aha, 1998). For example, k-nearest neighbor (kNN) (Atkeson et al., 1997) is highly susceptible to curse of dimensionality, especially when many features are irrelevant to the performance task (Friedman, 1997). A data set with too many features will cause the dimensionality problem (Weiss and Kulikowski, 1991), i.e., if many features are redundant and/or noisy, the distance measures can be particularly misleading.

The process of classification is completed in three major steps: Step 1, learning a classifier (a concept description) from the data; Step 2, classifying the data using the classifier; and Step 3, measuring the classifier's performance. In other words, the task of inducing the classifier is to learn a function $h(\mathbf{x})$ given a collection of instances of f, where h approximates f. In order to truly measure the performance of a classifier, the data is normally divided into two parts, a training set and a test set, which are used in the first two steps respectively. The performance of a classifier can be measured by learning speed (how fast a classifier is learned), predictive accuracy (how close the two functions f and h are[2]), and complexity of learned results (concepts). More details about these measurements will be discussed in Chapter 5. We briefly bring up these issues here to facilitate our discussion of feature selection.

Many aspects affect the performance of inducing a classifier. For data driven induction, the two prominent ones are (1) data, and (2) learning algorithms. With respect to the data, we have problems as follows:

- too much data so that a computer learning system cannot handle the data, and consequently, nothing can be learned;
- too little data so that a learning algorithm cannot learn anything meaningful, or there are too many hypotheses to be formed; and
- noisy data that distracts a learning algorithm in its learning of function h.

The choice of a learning algorithm also makes a difference in learned results (rules, decision trees, networks, etc.) as well as in predictive accuracy, speed of learning, and comprehensibility. In feature selection, we inevitably need to consider both classifiers and data, as we want to overcome the obstacles caused by data as well as to learn effectively with the limitations of classifiers.

Coming with the rapid growth of computer applications are the menacing profiles of ever-growing mountains of data. Although information is a key asset, raw data *is not* information. As a matter of fact, there exists a big gap between raw data and information, and tools are needed to bridge the gap. The confluence of several factors, such as humongous amounts of growing data, information hidden in data, and traditional tools available for use are responsible for the emerging field of knowledge discovery and data mining which answers the question of "how to transform raw data to information effectively".

1.2 KNOWLEDGE DISCOVERY AND DATA MINING

Knowledge Discovery in Databases (KDD) is concerned with extracting useful information from databases (Fayyad et al., 1996). Data mining is a set of techniques used in an automated approach to exhaustively explore and bring to the surface complex relationships in very large datasets. The adopted approaches are discovery-based in which pattern-matching and other algorithms can look at numerous multidimensional data relations concurrently, highlighting those that are dominant or exceptional (Moxon, 1996).

Data mining is usually viewed as a single step in the KDD process. The various steps in the process include: a) data warehousing, b) target data selection, c) cleaning, d) projection and reduction, e) model selection, f) data mining, g) evaluation and interpretation, and h) consolidation of the discovered information (see (Fayyad et al., 1996) for more detailed descriptions). Briefly, data warehousing allows data collected from multiple sources in different formats to be used for a single application; target data selection creates a dataset; cleaning removes noises and/or outliers; projection and reduction transform the data for proper use according to the application; model selection adopts a proper algorithm; data mining searches for patterns of interest and usefulness; evaluation and interpretation validate and explain the results; and consolidation makes

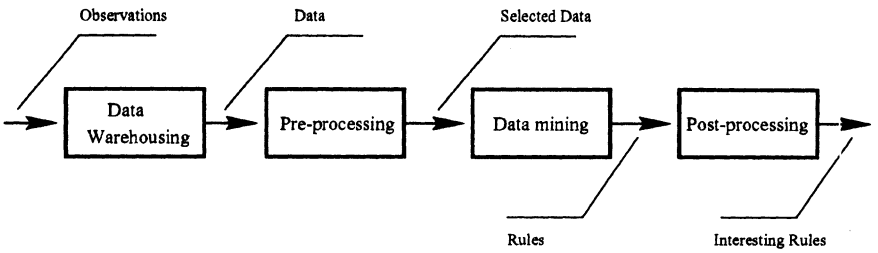


Figure 1.2. A general model of knowledge discovery and data mining (KDD).

the discovered information into a form that can be checked and reused. Considering these steps at the level of feature selection, they can be summarized into four basic steps: 1) data warehousing - observations from different sources are recorded as data; 2) pre-processing (b, c, and d) - data are selected for better use, given an application; 3) data mining (e and f) - a proper mining algorithm is chosen and rules (concepts) are learned, here rules are a generic term for what is mined and discovered; and 4) post-processing (g and h) - rules are sifted or grouped for their better use and understanding, since it is often that the number of rules is large, and some rules are more "interesting" than others (Piatetsky-Shapiro, 1991, Kamber and Shinghal, 1996, Silberschatz and Tuzhilin, 1996, Liu and Hsu, 1996). This four-step process is seen in Figure 1.2.

1.2.1 Data mining - a new challenge

Many existing approaches are available for data mining such as association, sequence analysis, numerical and conceptual clustering, classification, and estimation. Association approaches try to find association rules between features. A typical example is of classic market-basket analysis: one transaction is an instance as in the data example, the goal is to find which features are often associated with some other features among large numbers of transactions. The information can be used to adjust inventories, rearrange shelf layouts, or target promotional activities. Clustering[3] approaches address segmentation problems. Data with a large number of features are segmented into clusters of small datasets with intrinsic characteristics. Clustering is often one of the first steps in data mining. Clusters are formed to focus the effort of further study. In contrast to classification problems, clustering problems do not have a particular class feature. Classification approaches may be the most commonly applied data mining techniques, used widely in fraud detection and credit-risk

assessment. Estimation approaches are a variation of classification that handles pre-scored data and generates a credit-worthiness score, in other words, its class feature is continuous.

The promise of data mining is attractive for decision makers to make sense out of large volumes of data. The promise of KDD that data mining can go through the data and identify the key relationships relevant to the business is being pushed as a panacea for all data analysis woes. Yet this image is far from reality (Moxon, 1996).

Data mining tools have typically evolved from the research efforts in statistics, pattern recognition, databases, and machine learning. Tools were built for various purposes and handling data with different representations. Hence, at the confluence of data mining, many gaps exist between the new situations and the existing solutions. Outstanding examples are representation gap and tools gap. The representation gap reflects the different ways in which data are described (at various levels, e.g., a chair can be a piece of furniture, at a general level or a study chair, at a specific level), represented (such as flat files, relational databases, first order logic, lists, schemas) and collected (e.g., readings from instruments or inputs from a person). The tools gap summarizes the problems with these tools that have to interface with each other (before each tool performed satisfactorily in its specialized field) and handle data with sizes varying immensely.

It is, therefore, necessary to find ways to bridge these gaps. We either build new and more general tools or find means to adapt the data to the existing tools. Efforts along both lines should be encouraged. Experience tells us that the development of new tools takes time, new applications (problems) arise all the time, and new gaps appear when old gaps are bridged. This is a never-ending process. It shows the necessity of having various tools for different gaps. The subject of this book, feature selection methods, is about tools that select appropriate data subsets and transform data, for performance and consistency reasons, so as to bridge the representation gap.

The eventual goal in developing this set of tools is that with the assistance of this tool set, we can get the most out of the data using the available data mining techniques. Particular difficulties with data mining techniques are found in data dimensionality, amounts of data, quality of data, and data projection and visualization. Dimensionality is a problem for classification using K-nearest neighbor methods (Friedman, 1997)[4]. Large dimensionality becomes prominent in data for data mining. Since people seek comprehensive and reusable knowledge, it is convenient to define a large number of features for a domain to be investigated, in order to make sure that every possible aspect is covered. It is, therefore, increasingly likely to introduce irrelevant and/or redundant features that could mislead a data mining algorithm (a commonly seen example

is that a classifier overfits the data in all dimensions.) By choosing relevant features, the dimensionality is reduced.

When N is fixed, amounts of data can cause problems to data mining as well, be it too much or too little. Data can never be too much if all are relevant. It is always true that the more, the better. So what's the problem with excessive data? One possibility is that data is collected for general purposes, and for a particular data mining task, the irrelevant data is simply too much and obscures the mining. Preprocessing such as feature selection can refocus the data on the task by removing irrelevant features. Another possibility is due to the computer hardware restriction, i.e., each time, a computer memory can only take so much of the data, any extra data will be too much for it to process[5]. Excess in data is a relative concept in this sense. If a computer can only process part of the data, the mining cannot be done as effectively as on the whole data. Removing irrelevant features helps reduce the number of distinctive instances.

In other cases, we may face the lack of data problem: the available data is only a tiny portion of the data that is needed for a data mining algorithm to learn something meaningful; or the number of features is very large so that, relatively speaking, much more instances are needed. When we have too little data or too many features, feature selection can help a data mining algorithm zero in on relevant features so that the problem with huge amounts of data can be mitigated. An often heard saying is "garbage in garbage out"; it is specially so in computer processing. In the context of data mining, the quality of data is one of the key issues. With the presence of noise and missing values in data, the quality of data deteriorates. More often than not, this problem can be alleviated by selecting relevant features. Data visualization and projection enables us to visualize high dimensional data, to better understand the underlying structure, and to explore the intrinsic dimensionality of multivariate data. Feature selection and its relevant techniques can also help in this matter by choosing the most relevant features.

1.3 FEATURE SELECTION AND ITS ROLES IN KDD

As we have seen, feature selection is one important means to attack problems with various aspects of data, and to enable existing tools to apply, otherwise not possible. In order to better use this useful tool, we need to study feature selection in detail, what it is, and what we can do with it. From a pragmatic stand, we have to answer what is the result of feature selection and why we need feature selection. For the former, the effect is to have a reduced subset of features; for the latter, the purposes vary: we want (1) to improve performance (speed of learning, predictive accuracy, or simplicity of rules); (2) to visualize

the data for model selection; and/or (3) to reduce dimensionality and remove noise. Combining both considerations, we define feature selection as follows:

Definition 1 Feature selection *is a process that chooses an optimal subset of features according to a certain criterion.*

The *criterion* specifies the details of measuring feature subsets. The choice of a criterion can be influenced by the purposes of feature selection. An optimal subset can be a minimal subset; other things being equal, it can be a subset that gives the best estimation of predictive accuracy. In some cases, if the number of features is given (as in data visualization and projection, this number is usually 2 or 3), we need to find a subset with the specified number that satisfies the criterion best. In Chapter 2, we will review various criteria that evaluate features and discuss their relationships between each other. Intuitively, choosing a subset of M features from a full set of N features is equivalent to a reduction in the hypothesis space[6]. This would make it easier for a learning algorithm to learn with the available data. For example, if we want to learn a binary function from a binary domain, reducing the number of features from N to M means the reduction of data by a factor of $O(2^{N-M})$.

The application of feature selection helps all the three phases of knowledge discovery: pre-processing, mining, and post-processing. As a matter of fact, every phase is equally important for successful knowledge discovery, or data mining in particular. The first phase ensures that the data fed to the second phase is of good quality and of proper size; the second phase makes sure that data mining can be performed effectively and efficiently; and the third phase sorts out the mined results into reliable and comprehensible knowledge being relevant to what a user needs. The three closely connected phases constitute a coherent process of knowledge discovery. In the following, we briefly introduce some issues concerning each phase.

- *Pre-processing Phase* The goal of this phase is to get some idea about the data and prepare the data for the next phase.

  The issues this phase handles are:

  - *Too many features*
    A large number of features can make available instances to become relatively insufficient for mining. In practice, the number of features can be as many as some hundreds. If we have only a few hundreds of instances, dimensionality reduction is necessary in order for any reliable rule to be mined or to be of any practical use.
  - *Data overload*
    This is a data oversizing problem in the dimension of number of instances. The data is simply too much either for a mining algorithm

to process or for a computer to read it all in one go. Dimensionality reduction is necessary for data mining to be possible.

- *Data quality*
  This issue is not only about noisy data or contaminated data, but also about irrelevant data and redundant data. Recall that data is usually collected not solely for data mining purposes or for a specialized application. Dealing with relevant data alone can be far more effective and efficient. Using only the relevant data can also help in the previous two issues.

- *Pre-model selection*
  One way to get some idea about the data is via data visualization or some simple statistics (e.g., means and deviations (Michie et al., 1994)). Some preliminary or rough understanding of the data can greatly assist in selecting proper data mining algorithms, especially so if we want to use statistical data analysis tools.

- *Mining Phase* Various results can be produced in different formats in this phase and many data mining algorithms can be at our disposal. It is not always feasible to run all data mining algorithms for a particular application. Hence, we should consider the following issues:

  - *Model selection*
    We look for the best suitable algorithm for data mining. With the aid of pre-model selection and the understanding of the domain, we may be able to find some candidate algorithms that are more suitable for the application at hand than the rest.

  - *Accelerating* and *Focusing*
    Other things being equal, a faster mining algorithm is preferred. Nevertheless, for a complex application, if we let loose a mining algorithm (blind mining), no matter how fast it is, it may lead to nowhere, or to the best, take unbearably long time to discover something that may be useless. Focusing is about avoiding blind mining but concentrating on the aspects relevant to the application. If we can focus in the right direction, a fast algorithm can be even faster and more effective.

  - *Intermediate result displaying*
    One way of improving the quality of mining is via expert intervention, or interactive mining. If a mining algorithm allows, intermediate results should be timely presented to experts to assist their next-step decisions. Choosing the most relevant to present would enhance the quality of experts' decisions, and in turn, improve the performance of data mining.

- *Post-processing Phase* More often than not, the results from the Mining Phase can still be too many and disorganized. Some knowledge may be well known and too much of it could make oblivious some newly discovered knowledge. Furthermore, we need some solid evidence to convince ourselves that the discovered is really nuggets not some nuisance. In this sense, feature selection can help in post-processing.

  - *Evaluation*
    We scientifically evaluate the mined results to make sure that they are indeed statistically reliable and significant. Feature selection reduces data dimensionality which makes mining not only faster, easier, but also more accurate, and more reliable.
  - *Consolidation*
    It is often the case that too many rules are induced. We need to sort them out and categorize them so that more interesting or important ones can stand out and would not be buried in too many unsorted rules. The problem of feature selection is changed to that of rule selection.
  - *Comprehensibility*
    Finding rules is wonderful, finding comprehensible ones is great. The knowledge extracted from data is for reuse, fine-tunning, and expansion of existing ones. Its comprehensibility determines if it can be understood, passed on, and applied. Feature selection reduces the data dimensionality, and thus very likely improves the comprehensibility of rules (in simple terms, each rule is described by fewer number of features).

1.4 SUMMARY

Feature selection can help in all the three phases of knowledge discovery: it selects features relevant to a particular application, removes irrelevant and/or redundant ones, improves the data quality, makes mining algorithms work faster on larger sized data, and enhances the comprehensibility of mined results. In conclusion, feature selection methods are useful in the three phases of data mining. They are needed now and will be needed in the future. This is because there will always be tools and representation gaps as new problems appear and new fields emerge. Some problems that were not dealt with in statistics are now handled by machine learning, and data mining emerges as a field that considers problems that were not issues for statistics and machine learning. At the time of developing new and more powerful tools, we may also want to bridge the gaps caused by the limitations of existing tools. Feature selection can be considered as a set of special tools bridging the gaps and boosting

the performance of existing tools. Considering the two key components of knowledge discovery: mining algorithms and data sets, tools can be created for both. Feature selection is about the tools of manipulating data to suit the mining algorithms, but without changing the nature of the data.

We have not yet reached the stage where robust off-the-self tools can easily be deployed in data mining applications. Most of the existing tools still need quite a bit of expertise in their deployment and fine-tuning until good analysis results are achieved. While it is in high demand to build the right set of tools and systems for data mining, it is also important to develop tools that can close the gaps between systems and applications. Feature selection is one of such effective means to boost data mining approaches, to help close the gaps in tools and representations, and to enable more techniques in our data mining endeavors.

In the following three chapters, we will elaborate on feature selection from its perspectives, aspects, to methods. Although data mining approaches are not limited to classification as we mentioned above, we will begin with feature selection for classification and expand it to feature selection for unsupervised data, and other related issues such as feature discretization, extraction and construction.

References

Aha, D. (1997). Editorial. *Artificial Intelligence Review*, 11.

Aha, D. W. (1998). *Feature Weighting for Lazy Learning Algorithms*, pages 13–32. In (Liu and Motoda, 1998).

Atkeson, C., Moore, A., and Schaal, S. (1997). Locally weighted learning. *Artificial Intelligence Review*, 11.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *European Working Session on Learning.*

Dasarathy, B. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques.* IEEE Computer Society Press.

Dietterich, T. (1997). Machine learning research: Four current directions. *AI Magazine*, pages 97–136.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery: An overview. In Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 495–515. AAAI Press / The MIT Press.

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

Kamber, M. and Shinghal, R. (1996). Evaluating the interestingness of characteristic rules. In *Proceedings of the Second International Conference on Data Mining (KDD-96)*, pages 263–266. AAAI Press.

Lampinen, J., Laaksonen, J., and Oja, E. (1998). Pattern recognition. In Leondes, C., editor, *Image Processing and Pattern Recognition*, volume 5 of *Neural Network Systems Techniques and Applications*, pages 1 – 59. Academic Press.

Liu, B. and Hsu, W. (1996). Post-analysis of learned rules. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 828–834. AAAI Press / The MIT Press.

Liu, H. and Motoda, H., editors (1998a). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Liu, H. and Motoda, H. (1998b). Feature transformation and subset selection. *IEEE Intelligent Systems*, 13(2):26 –28.

Mendenhall, W. and Sincich, T. (1995). *Statistics for Engineering and The Sciences*. Prentice Hall International, 4th edition.

Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in Artificial Intelligence.

Mitch, T. (1997). *Machine Learning*. McGraw-Hill.

Moxon, B. (1996). Defining data mining. *DBMS ONLINE*, DBMS Data Warehouse Supplement.

Piatetsky-Shapiro, G. (1991). Discovery, analysis, and presentation of strong rules. In Piatetsky-Shapiro, G. and Frawley, W. J., editors, *Knowledge Discovery in Databases*, pages 229–248. AAAI / The MIT Press.

Silberschatz, A. and Tuzhilin, A. (1996). What makes patterns interesting in knowledge discovery systems. *IEEE Trans. on Knoweldge and Data Engineering*, 8(6):970–974.

Watanabe, S. (1985). *Pattern Recognition: Human and Mechanical*. Wiley Interscience.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems That Learn*. Morgan Kaufmann Publishers, San Mateo, California.

Winston, P. (1992). *Artificial Intelligence, Third Eddition*. Addison Wesley.

Notes

1. There are many types of learning tasks. Some examples are conceptual clustering, reinforcement learning, macro operator learning (speedup learning), among many.

2. If using f to predict, we expect 100% accuracy, then any error rate due to h indicates how far h is away from f.

3. A reader may wonder why sequence analysis is not mentioned. This is because sequence analysis uses datasets different from those as shown in Table 1.1. The issue there is more about determining the window size for the temporal data than selecting features.

4. An important aspect contributing to the successful resistance of classification error to the curse-of-dimensionality is the choice of a good value for the number of nearest neighbors K (Friedman, 1997).

5. No matter how advanced a computer is, this physical limitation always exists because for some applications, data collection is massive, non-stopping, and endless.

6. The hypothesis space contains all the possible functions that can be learned from the data.

2 PERSPECTIVES OF FEATURE SELECTION

From here on, we study feature selection for classification. By choosing this type of feature selection, we can focus on many common perspectives of feature selection, obtain a deep understanding of basic issues of feature selection, appreciate many different methods of feature selection, and later in the book move on to topics related to feature selection. The problem of feature selection can be examined in many perspectives. The four major ones are (1) how to search for the "best" features? (2) what should be used to determine best features, or what are the criteria for evaluation? (3) how should new features be generated for selection, adding or deleting one feature to the existing subset or changing a subset of features? (That is, feature generation is conducted sequentially or in parallel.) and (4) how applications determine feature selection? Applications have different requirements in terms of computational time, results, etc. For instance, the focus of machine learning (Dietterich, 1997) differs from that of data mining (Fayyad et al., 1996).

Firstly, feature selection is considered as a problem of searching for an optimal subset which can usually be dealt with by some lattice traversal algorithms. If subsets of features are properly generated, trees instead of lattices are searched. Secondly, evaluation criteria are surveyed and their characteris-

tics are analyzed. Different types of measures have their unique applications. Thirdly, feature selection is reviewed in the way that features are evaluated, mainly, univariate and multivariate evaluation. In the course of feature selection, univariate evaluation considers adding the best feature among the unchosen features to the set of chosen features or deleting the least important features from the set of chosen features, and multivariate evaluation considers a subset of features instead of a single feature in searching for the best subset. Lastly, feature selection is studied based on how it is involved in classification (Breiman et al., 1984, Quinlan, 1993). A feature selection component can be used as a preprocessor for a classifier or a performance booster for a classifier. In other words, it can be viewed as a filter or a booster for a classifier. Whenever appropriate, we will draw examples from fields such as statistics, neural networks, machine learning, and pattern recognition to elaborate on these perspectives. In a way, we prescribe some general solutions to the problems of feature selection for classification, although more technical details are discussed in the next chapter.

2.1 FEATURE SELECTION FOR CLASSIFICATION

The basic problem of pattern classification is concerned with the classification of a given object to one of d known classes. An object is represented by a pattern of features which, presumably, contains sufficient information to distinguish among the classes (Ben-Bassat, 1982). A pattern is an "attribute-value" or "feature-value" representation with a specified class. A data set (usually called a training set) consists of patterns with known classes. If we like, the data set can sometimes be rewritten to class-conditional probabilities of features. The task is to induce a classifier that can predict an unseen pattern's class. The goal is to have a classifier that can predict accurately and at the same time, its structure be as simple as possible.

For a given problem, the number of potential features can be quite large. This is because:

- Data collected are not solely for one particular task such as data mining. For example, it is often that data are collected for a general task like bookkeeping, or required by a law. On one hand, the data can be used for different purposes since it is not specially designed to serve one task; on the other hand, for a particular application, it is quite likely that many *redundant* or *irrelevant* features are present.

- Sometimes even if we know features are designed for a particular task, relevant features are often unknown. This is due to the nature of investigation. We conduct experiments and collect data because we want to know more

about the domain and the problem, and we usually do not have a precise idea about the features (or dimensions describing the domain and problem) needed. Therefore we have to introduce candidate features as many as we can think of even if some may be remotely relevant; consequently, some of them are inevitably redundant or irrelevant. We can only know which features are relevant after we study the data collected.

- A task may require data from different sources. If data from each source is moderately large, the join of them could be huge. Only if we know the relevant features may we be able to attack the problem. Again we usually don't know beforehand which features are relevant.

Irrelevant or redundant features may have negative effects on classification algorithms: (1) Having more features usually means the need of more instances since we need to ensure the statistical variability between patterns from different classes. Consider a binary domain, a complete set with five-feature data contains 32 instances, while it has 1024 instances with ten features. It usually takes a classification algorithm longer time to learn with more data. (2) Redundant or irrelevant features/data may mislead learning algorithms or cause them to overfit the data. Hence, the obtained classifier in general is less accurate than the one learned from the relevant data. (3) In addition, with the presence of redundant or irrelevant data, it is more likely that the classifier obtained is more complex. This will unnecessarily hinder our understanding of the learned results. Following the rationale behind Occam's razor (Blumer et al., 1990, Russell and Norvig, 1995), a complex classifier tends to be less accurate comparing to a simpler classifier.

Since we usually cannot determine which features are relevant before data collection and redundant/irrelevant features have negative effects on classification algorithms, we are faced with the feature selection problem. Basically, we want to choose features that are relevant to our application in order to achieve maximum performance with the minimum measurement effort. Feature selection results in:

1. *less data* so that the classification algorithm can learn faster;
2. *higher accuracy* so that the classifier can generalize better from data;
3. *simpler results* so that they are easier to understand; and
4. *fewer features* so that in the next round of data collection, savings can be made by removing redundant or irrelevant features, if possible.

In the following, we will examine feature selection from these perspectives: search, evaluation, generation, and selection model.

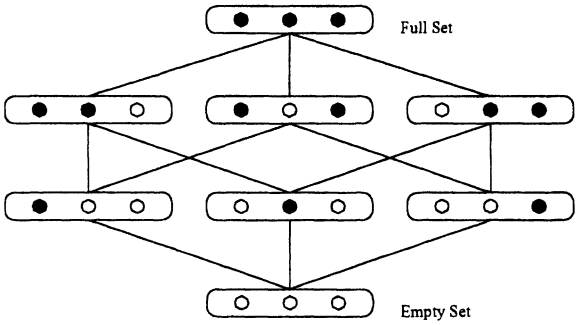


Figure 2.1. Feature selection as a search problem: a lattice and three features.

2.2 A SEARCH PROBLEM

Feature selection can be viewed as a search problem, with each state in the search space specifying a subset of the possible features. Let's look at a data set with three features (A_1, A_2, A_3). There is a corresponding binary array, an element of being 1 as the feature indexed in the array is selected. Hence, (1, 1, 1) means all three features are selected and (1, 0, 0) says only A_1 is chosen. There should be a total of 2^N subsets where N is the number of features of a data set. In the case of three features, there are 8 states (subsets), as depicted in Figure 2.1. At the two ends lie two extreme subsets: one is full with all three features selected and the other is empty without any feature. An optimal subset in a more realistic case is usually somewhere in between the two ends. This leads us to this question: Where should we start our search? This problem becomes trivial if the search space is small, since we can choose either end to start and search will complete in no time. Usually, the search space is, however, not small (it is 2^N). N is normally large ($N > 20$) in a data mining application. This makes the question about the starting point of search particularly pertinent and gives rise to another question: What are suitable search strategies? Search strategies can be influenced by search directions.

2.2.1 Search directions

Assuming no prior knowledge about where the optimal subset of features is in the search space, there will be no difference to start the search from either end (an empty set or a full set). That is, for a large number of problems, on average, a method searching in one direction will find the optimal subset as fast as a method searching in the other direction. Search directions are closely related

to feature subset generation. One direction is to grow a feature subset from an empty set (e.g., Sequential Forward Generation) and the other direction is to gradually remove least important features at that point from the full set (e.g., Sequential Backward Generation).

- *Sequential Forward Generation* (SFG) It begins with an empty set of features, S_{select}. As search starts, features are added into S_{select} one at a time (thus, sequential). At each time, the best feature among unselected ones is chosen based on some criterion. S_{select} grows until it reaches a full set of original features. A ranked list of selected features can also be established according to how early a feature is added into the list. If there is some prior knowledge about the number of relevant features (say m), we simply choose the first m features in the ranked list. A more general form of SFG is forward generation which starts with subsets of one feature, then subsets of two features, and so on.

- *Sequential Backward Generation* (SBG) It begins with a full set of features. Features are removed one at a time. At each time, the least important feature is removed based on some criterion. So, the feature set shrinks until there is only one feature. A ranked list of selected features can also be established according to how late a feature is removed. The last removed is the most important. SBG complements SFG since sometimes the best feature is easier to find than the least important one and sometimes it is the other way around. A more general form of SBG is backward generation which begins with subsets of all N features, the subsets of $N - 1$ features, and so on.

If there exists an optimal subset which is not in the middle range of the search space, it makes sense that we start search from both ends. Intuitively, it is faster than the search in one direction. If our above assumption of no knowledge about m is true, the single directional search will pass the middle point half the time for all problems. The bidirectional search can certainly avoid doing so. When search starts in both directions, the search in one direction usually stops earlier than in the other direction. Therefore, we gain by using bidirectional search most of the time.

- *Bidirectional Generation* (BG) It starts search in both directions, i.e., two searches proceed concurrently. They stop in two cases: (1) when one search finds the best (remaining) m features before it reaches the middle; or (2) when both searches reach the middle of the search space. BS takes advantage of both SFG and SBG. It is clear that the selected feature sets found by SFG and SBG respectively may not be the same due to their sequential adding/deleting of features.

Another type of feature generation enriches the spectrum of search directions. That is Random Generation. There is no particular direction to follow in feature generation. Properly restricting the search space based on the probabilistic probes, this type of generation complements other types of feature generation.

- *Random Generation* (RG) It starts search in a random direction. Adding or deleting features is also done at random. RG tries to avoid trapping into local optima by not sticking to a fixed way of subset generation. Unlike SFG or SBG, the size of a subset generated next cannot be determined, though we can see a trend of which the number of selected features is decreasing or increasing.

With four types of search directions, we are able to design search for different needs. Sometimes, we may want a quick and dirty solution to get a feel about the data sensitivity to feature selection; sometimes, we really want an optimal subset regardless of the cost of getting it; yet sometimes, we are happy with a subset that improves along time. Combined with various search strategies, we can design a feature selection algorithm that best suits the problem at hand.

2.2.2 Search strategies

Various sizes of search spaces open up an opportunity for the study of search strategies (Russell and Norvig, 1995). When the number of features (N) is small, the search space cannot be large; but it grows exponentially when N increases due to the relationship $S = 2^N$ where S is the size of a search space with respect to N, where 2 represents two choices: to choose or not to choose a feature. Brute-force search strategies are out of the question when N is large. In general, given a search space, the more you search it, the better the subset you can find. To search more means to consume more time. When this resource is not unlimited, we have to sacrifice optimality of selected subsets. Because too much of it means a selected subset is useless, the sacrifice has also a limit. The objective is to keep the optimality of a feature subset as much as possible while spending as little search time as possible. That is not an easy task. The efforts can be summarized into three categories.

- *Exhaustive/complete search* Exhaustive search will, as the name suggests, exhaust all possible subsets and find the optimal ones. In general, its space complexity (the number of subsets need to be generated) is $O(2^N)$. If we know the direction of search (e.g., forward generation), the search space is $\binom{N}{0}+\binom{N}{1}+...+\binom{N}{M}$, where M is the minimum number of features of a subset that satisfies some evaluation criterion. In the case of backward generation,

it starts with a full set until a minimum subset is found. The search space is $\binom{N}{N}+\binom{N}{(N-1)}+...+\binom{N}{M}$. It is obvious that no optimal subset can possibly be missed. That is because no matter which direction of the search continues, it is breadth-first search that checks one layer at a time as in Figure 2.1.

A natural question is "Can we do better?" In other words, if we do not want to miss out an optimal subset, do we have to resort to exhaustive search? The answer is we can do better in some cases since being complete (i.e., no optimal subset is missed) does not necessarily mean that search must be exhaustive (every state has to be visited) (Schlimmer, 1993). If the evaluation measure of a subset possesses a certain property (e.g., monotonicity), we can find an optimal subset without trying out all the subsets outlined in the previous paragraph. However, this improvement is not enough since the search space is still large and the saving is just a constant factor of 2^N. More about this will be discussed when Branch and Bound (Narendra and Fukunaga, 1977) is described in Chapter 3. Only exhaustive search can guarantee the optimality, when monotonicity cannot be satisfied.

- *Heuristic search* As the name suggests, it employs heuristics in conducting search. So it avoids brute-force search, but at the same time risks losing optimal subsets. It is a sort of depth-first search guided by heuristics. The space complexity of heuristic search could be just a path connecting the two ends as in Figure 2.1. The maximum length of such a path is N. The number of subsets generated is $O(N)$. Heuristic search is obviously much faster than exhaustive search since it only searches a particular path and finds a near-optimal subset.

The above two categories complement each other in a sense that one does what the other is incapable of doing. Heuristic search strategies cannot guarantee the optimality of a subset while complete search strategies can, but there is a possibility that you may not get a solution within a reasonably long period. This is because the latter may take extremely long time with unlimited memory/disk space. If one is short of either time or space, the result could be disastrous. If one neither wants to easily get stuck to the local optima, nor spend too much time, something else is needed. Therefore, there exits the third category - random search strategies. This is also related to *anytime algorithms* (Zilberstein, 1996) which can generate currently best subsets constantly and keep improving the quality of selected features as time goes by.

- *Nondeterministic search* Unlike the first two types of search strategies, this strategy searches for the next set at random, i.e., a current set does not directly grow or shrink from any previous set following a deterministic rule. There are two characteristics about this type of search strategies: (1) we

Table 2.1. A 2-d table about search strategies and search directions.

Search Direction	Search Strategy		
	Complete	Heuristic	Nondeterministic
SFG	✓	✓	✗
SBG	✓	✓	✗
BG	✓	✓	✗
RG	✗	✓	✓

do not need to wait until the search ends; and (2) we do not know when the optimal set shows up, although we know a better one appears when it's there.

With the understanding of search directions and search strategies, we can examine their possible combinations summarized in a 2-dimensional table (Table 2.1). ✗ means that particular combination is not sensible. For nondeterministic search, only random generation is considered possible whereas sequential types of feature generation are ruled out. Although in random generation, we can still control the growth or reduction of a feature subset, but no sequential relation can be found between two consecutively generated feature subsets.

2.3 SELECTION CRITERIA

We studied various search strategies and directions for optimal feature sets. We now need to address the issue of "what is a *good* feature?" Without defining the "goodness" of a feature or features, it does not make sense to talk about best or optimal features. Let's look at feature selection from the perspective of measuring features.

The need for evaluation of a feature or a subset of features is common to all search strategies. This evaluation issue is complex and multi-dimensional. For example, it can be measured in terms of (1) whether selected features help improve a classifier's accuracy and (2) whether selected features help simplify the learned results so that they are more understandable. Hence, we need to extensively discuss many aspects of feature evaluation. But we will discuss these in Chapter 5. As our task is feature selection for classification, the primary goal of pattern classification is to *maximize* predictive accuracy. It is, therefore, reasonable that predictive accuracy is the primary measure for feature evaluation, which is generally accepted and widely used by researchers and practitioners.

2.3.1 *The ideal classifier*

In order to estimate predictive accuracy, we need a classifier that is learned from the training data and can be tested on the test data. An optimal classifier is the equivalent of direct table lookup, which can, in turn, be formulated in precise mathematical terms - the probabilistic theory of Bayesian analysis (Weiss and Kulikowski, 1991). Adopting the Bayesian approach, the true class of a given object is considered as a random variable C taking values in the set $\{c_1, c_2, ..., c_d\}$. The initial uncertainty regarding the true class is expressed by the prior probabilities $P(c_i)$. Mathematically speaking, the table lookup criterion can be stated as selecting the class c_i with the greatest posterior probability for a given pattern of evidence $\mathbf{x}$ where c_i is chosen such that

$$P(c_i|\mathbf{x}) > P(c_j|\mathbf{x}) \quad \texttt{for all } i \neq j. \tag{2.1}$$

Since

$$P(c_i|\mathbf{x}) = P(\mathbf{x}|c_i)P(c_i)/P(\mathbf{x}), \tag{2.2}$$

where $P(\mathbf{x})$ is taken with respect to the mixed probability of $\mathbf{x}$,

$$P(\mathbf{x}|c_i)P(c_i) > P(\mathbf{x}|c_j)P(c_j) \quad \texttt{for all } i \neq j. \tag{2.3}$$

If the cost for all types of correct classification is zero and the cost for all types of incorrect classification is one, then the optimal Bayesian classifier assigns the instance to the class with the highest *a posteriori* probability. The probability of error for a given pattern of evidence, $P_e(\mathbf{x})$, which is expected after observing $\mathbf{x}$:

$$P_e(\mathbf{x}) = \mathbf{E}[1 - max\{P(c_1|\mathbf{x}), ..., P(c_d|\mathbf{x})\}] \tag{2.4}$$

$$P(\mathbf{x}) = \sum_{i=1}^{d} P(c_i)P(\mathbf{x}|c_i). \tag{2.5}$$

$P_e(\mathbf{x})$ can be used to rank $\mathbf{x}$. Since $\mathbf{x}$ can be univariate or multivariate, we need to exhaustively search all possible subsets for optimal subset selection of M features out of N. It is not practical when N is large. Algorithms based on the relative value of individual features are suggested. However, using the probability of error as the evaluation function for individual features does not ensure good error rate of the resulting subset, not even for the case of conditionally independent features. The probability of error may not be sensitive enough for differentiating between good and better features even if we

assume that the complete distribution of $\mathbf{x}$ can be obtained through exhaustive search (Ben-Bassat, 1982).

From the viewpoint of reducing error rate, nevertheless, there are no bad features for a Bayesian classifier. This is because we cannot improve predictive accuracy of a Bayesian classifier by eliminating a feature (Siedlecki and Sklansky, 1988). Unfortunately, there are also serious obstacles to the direct application of the Bayesian approach. Because sample sizes are finite, they are rarely sufficient to reliably estimate all the probabilities and statistics that the approach requires. If the complete distributions were known (unless the data set is very small and artificial, but we do not have such a luxury in data mining), there would not be a need for predicting classes. As a consequence, we need to rely on non-ideal classifiers (examples in the next section) and it is possible to improve the performance of a non-ideal classifier by removing redundant and/or irrelevant features. In short, the $P_e(\mathbf{x})$ rule can be neatly derived from the Bayesian analysis, but it has many obstacles to apply in practice. Since ideal alternatives for the $P_e(\mathbf{x})$ rule do not generally exist (Ben-Bassat, 1982), researchers explore various other measures.

2.3.2 Information measures

Information is a way of measuring the uncertainty of the receiver when he receives all messages. If the receiver knows what is coming, his expected surprise level (uncertainty) is low; if he does not know at all what is coming, a reasonable assumption is that all messages have almost equal probabilities to come, his expected surprise level is high. In the context of classification, messages are classes. An information measure U is referred to as the uncertainty function concerning the true class, and is defined so that larger values for U represent higher levels of uncertainty.

Given an uncertainty function U and the prior class probabilities $P(c_i)$ where $i = 1, 2, ..., d$, the information gain from a feature X, IG(X), is defined as the difference between the prior uncertainty $\sum_i U(P(c_i))$ and the expected posterior uncertainty using X, i.e.,

$$IG(X) = \sum_i U(P(c_i)) - \mathbf{E}[\sum_i U(P(c_i|X))] \tag{2.6}$$

A feature evaluation rule derived from the concept of information gain states that feature X is chosen over feature Y if $IG(X) > IG(Y)$. That is, a feature should be selected if it can reduce more uncertainty. In other words, if a feature cannot reduce uncertainty, it must be irrelevant. Since $\sum_i U(P(c_i))$ is independent of features, we can restate the rule as X is preferred to Y if $U'(X) < U'(Y)$, where $U' = \mathbf{E}[\sum_i U(P(c_i|X))]$. This very idea is used in

ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993) for selecting a feature to grow a decision tree.

A commonly used uncertainty function is Shannon's entropy,

$$-\sum_i P(c_i)\log_2 P(c_i).$$

Many other uncertainty functions have also been suggested. They were briefly reviewed in (Ben-Bassat, 1982).

2.3.3 *Distance measures*

This category of measures is also known as separability, divergence or discrimination measures. One typical type is derived from distances between the class-conditional density functions. For the two-class case, if $D(X)$ is the distance between $P(X|c_1)$ and $P(X|c_2)$, a feature evaluation rule based on distance $D(X)$ states that X is chosen over Y if $D(X) > D(Y)$. This is because we try to find the feature that can separate the two classes as far as possible. The larger the distance, the easier to separate the two classes. If $P(X|c_1)$ and $P(X|c_2)$ are equal (the feature X plays no role in separating c_1 and c_2), $D(X)$ is 0; if $P(X|c_1) = 1$ and $P(X|c_2) = 0$, or vice versa, $D(X)$ is maximum.

Distance functions between the prior and posterior class probabilities have also been proposed as a tool for feature evaluation. This is because a feature which may change more drastically the prior assessment concerning the true class is a better feature. This approach is, in principle, similar to the information gain approach except that distance functions are used instead of information functions.

2.3.4 *Dependence measures*

This type of measures is also known as association or correlation measures. They are designed to quantify how strongly two variables are associated or correlated with each other so that by knowing the value of one variable one can predict the value of the other. In feature evaluation, instead of checking how a feature changes information gain or *a posterior* class probability, we look at how strongly a feature is associated with the class. Denoting by $R(X)$ a dependence measure between feature X and class C, we choose feature X over feature Y if $R(X) > R(Y)$. Put it another way, we select a feature which associates more closely with class C. If X and C are statistically independent, they are not associated and removing X should not affect the class separability of the remaining features. If each value of X is associated with one value of C, we should expect $R(X)$ to attain its maximum and feature X should be selected.

Feature evaluation rules derived from dependence measures are closely related to feature evaluation rules derived from information and distance measures. They are considered as another group of measures because of its different way of viewing features in classification. That is, features strongly associated with class are good features for classification.

2.3.5 Consistency measures

The above three measures share one commonality, i.e., they attempt to find the best features that can *maximally* tell one class from the others. One common problem with them is that they cannot break tie between two equally good features. Therefore these measures cannot detect whether one of them is redundant. Consistency measures, however, attempt to find a *minimum* number of features that separate classes as consistently as the full set of features can. In other words, consistency measures aim to achieve $P(C|FullSet) = P(C|SubSet)$. Feature evaluation rules derived from consistency measures state that we should select the minimum subset of features that can keep the consistency of the data maintained by the full set of features. An inconsistency is defined as two instances having the same feature values but different classes. Using consistency measures, both irrelevant and redundant features can be removed.

2.3.6 Accuracy measures

This category of measures relies on classifiers. For a given classifier, among many possible subsets of features, chosen is the subset that can bring about best predictive accuracy. Obviously, accuracy measures stand out from the above four categories. The rationale is that since we want to have a good set of features to improve the accuracy of a classifier, why don't we use the classifier's accuracy as a measure? With this type of measures come some considerations. One consideration is how to truly estimate predictive accuracy and avoid overfitting. Another consideration is that a classifier takes time to be learned from the data by a learning algorithm (by the way, the first consideration often further slows down a classifier). Later in this chapter we will come back to these considerations again when discussing the wrapper models that use accuracy measures.

2.3.7 Discussion

Five different types of measures have been introduced: information, distance, dependence, consistency, and accuracy. The first three are closely related (Ben-Bassat, 1982). So we roughly have three major groups: classic measures, con-

sistency measures, and accuracy measures. But the first four types are all about class separability, and the last one is about accuracy. Thus, we arrive at a hierarchy (Figure 2.2) that shows how these five types are related.

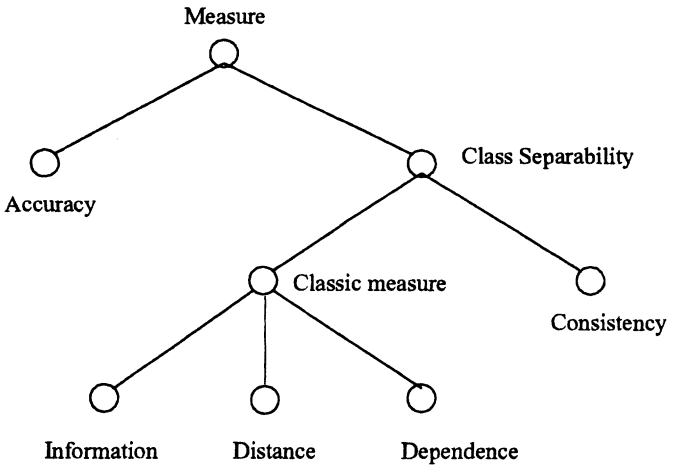


Figure 2.2. The relations between the five types of measures.

Definition 2 Feature relevance: *a relevant feature is the feature if it is removed the measure of the remaining features will deteriorate, be the measure accuracy, consistency, information, distance, or dependence.*

The definition of feature relevance is not based on the assumption that optimality is defined in terms of accuracy. Only when a measure is chosen can a feature's relevance be determined. When accuracy is the measure of optimality, if a feature's removal does damage to the classifier's estimated accuracy, this feature is relevant; when consistency is the measure, accuracy won't be estimated until a classifier is induced with selected features. The point is that the relevance of a feature depends on the measure chosen. If the removal of a feature does not affect the change of the measure, the feature is irrelevant. Some measures are not sensitive to redundant features; others are. (Kohavi and John, 1997) further differentiate between strong and weak relevance.

Naturally, we'd like to question whether these measures complement one another and how they fare in removing irrelevant and/or redundant features, and improving a classifier's accuracy. Intuitively, a feature selection algorithm using the accuracy measure may produce the best accuracy for a specific classifier without concern of whether features are consistent or dependent. Using

a consistency measure, we may remove redundant features, but it may not be so clear if the information, distance, or dependence measure can or cannot do so (for example, a tie between two features *might* mean one feature is redundant). When we probe further, we'll be able to experimentally verify or refute our conjectures. Note that all types of measures can remove irrelevant features. Another important issue that requires our immediate attention is what to measure, a single feature or a set of features? This is the topic of the next section.

2.4 UNIVARIATE VS. MULTIVARIATE FEATURE SELECTION

The focus of this perspective is on the relationship among features, i.e., whether we should consider adding or deleting an individual feature or a subset of features at each round of feature selection. We use various classifiers to explain how features are used so as to shed lights on how features should be treated in selection. Three representative approaches are discussed: (1) a naive Bayesian classifier; (2) a decision tree classifier; and (3) a neural network classifier. The first two classifiers assume features are independent of each other. For a naive Bayesian classifier, statistics about individual features are accumulated, the prior and univariate posterior class probabilities are calculated; for a decision tree classifier, a single feature is tested in splitting a decision node. A neural network classifier makes use of all features at each epoch in training the network. In short, the first two classifiers are univariate, and the third is multivariate in using features.

2.4.1 A statistical approach

As we discussed earlier in Section 2.3, Bayesian Classifiers are optimal. However, it is infeasible to apply them directly since they require the probabilities of all combinations of evidence, $P(x_1, x_2, ..., x_n|c)$. Therefore, people resort to the assumption of conditional independence of features.

When it is assumed that features A and B, with conditional probabilities $P(A|C)$ and $P(B|C)$, are probabilistically independent, the probability of A and B occurring together, $P(A, B|C)$, is given by the product of $P(A|C)$ and $P(B|C)$. This is the most usual simplification when applying the Bayes rule, either because it is approximately true, or just because it simplifies computations dramatically.

Thus born is a Naive Bayesian Classifier which is derived from Bayesian Classifiers assuming conditional independence of features (Domingos and Pazzani, 1996). As we know, given evidence e, an optimal classifier should select the class c_i with the greatest posterior probability, i.e., c_i is chosen such that

$$P(c_i|\mathbf{x}) > P(c_j|\mathbf{x}) \text{ for all } i \neq j.$$

The Bayes rule relates $P(c|\mathbf{x})$ to $P(\mathbf{x}|c)$ - the conditional probability of a given pattern of evidence for a specific class.

$$P(c|\mathbf{x}) = P(\mathbf{x}|c)P(c)/P(\mathbf{x})$$

An alternative formulation of the minimum error rate decision rule is:

$$P(\mathbf{x}|c_i)P(c_i) > P(\mathbf{x}|c_j)P(c_j) \text{ for all } i \neq j.$$

Because of the conditional independence, we can obtain $P(\mathbf{e}|c_i)$ through the following:

$$P(\mathbf{e}|c_i) = P(x_1|c_i)P(x_2|c_i)...P(x_n|c_i).$$

Thus, we need two groups of numbers to implement the Naive Bayesian Classifier: $P(c)$ and $P(x_i|c)$. Building a Naive Bayesian Classifier is equivalent to having tables of $P(c)$ and $P(x_i|c)$.

2.4.2 A decision tree approach

This approach is different from the Bayesian one in that it builds a decision tree to partition the data using information gain until instances in each leaf node have uniform class labels. At each test, a single feature is used to split the node according to the feature's values.

Basic Algorithm of Tree Induction

- Initialize by setting variable T to be the training set.
- Apply the following steps to T.
  1. If all elements in T are of class c_j, create a c_j node and halt[1].
  2. Otherwise select a feature F with values $v_1, v_2, ..., v_N$.
     Partition T into $T_1, T_2, ..., T_N$, according to their values on F.
     Create a branch with F as a parent node and $T_1, T_2, ..., T_N$ as the child nodes.
  3. Apply the procedure recursively to each child node.

Information gain is used to choose feature F. Hence, building from the training data, we obtain a tree with its leaves being class labels. When classifying a new instance, the instance traverses the tree according to (sub)root's (test) value of the tree down to a leaf. Its class is the predicted class for the instance.

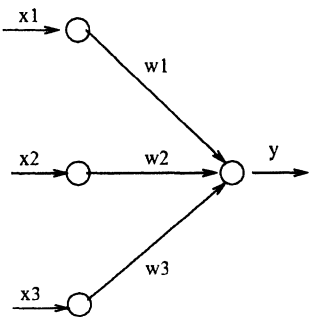


Figure 2.3. A simple neural network: Perceptron.

For the sake of efficiency, when building a decision tree, only one feature is used at a time to split the node and partition the data. Hence, features are used in a univariate fashion. See (Quinlan, 1986, Quinlan, 1993) for the details of handling continuous values, missing values, and many other issues related to decision tree induction, see (Mingers, 1989a, Mingers, 1989) for various techniques of tree pruning and selection measures.

2.4.3 A neural network approach

There are many types of neural networks for classification. We use a simple model, a single layer Perceptron (Hecht, 1990), to show how features are used in classification. A single-neuron perceptron can be seen in Figure 2.3. It is a linear threshold unit (LTU) with three input neurons and one output neuron. An LTU with one output neuron divides the input space into two regions: a region containing all the inputs (instances) that turn the LTU on and a region containing all the inputs (instances) that turn it off. So the LTU can classify instances into two classes (on or off). Classifying instances into more than two classes requires more output neurons (Hagan et al., 1996). Input neurons are represented by $\mathbf{x}$ and output neurons by $\mathbf{y}$. Links connecting input neurons with output one are attached with weights $\mathbf{w}$. For the example in Figure 2.3. $\mathbf{x} = [x_1, x_2, x_3]$, $\mathbf{w} = [w_1, w_2, w_3]$, $\mathbf{y} = [y]$, and x_i's are features.

$$y = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x}^T \geq \theta, \\ 0 & \text{if } \mathbf{w} \cdot \mathbf{x}^T < \theta. \end{cases}$$

where θ is threshold.

To train a perceptron network is to learn the weights $\mathbf{w}$ such that its predictive accuracy is maximum. A simplified perceptron learning rule is as follows:

given an instance $\mathbf{x}$ ($0 \leq x_i \leq 1$), its target t (either 0 or 1), δ is the learning rate which is a positive number less than 1.

$$\begin{aligned} &\text{if } y = t, \text{ then } \mathbf{w}^{new} = \mathbf{w}^{old}. \\ &\text{if } y = 1, \text{ and } t = 0, \text{ then } \mathbf{w}^{new} = \mathbf{w}^{old} - \delta\mathbf{x}. \\ &\text{if } y = 0, \text{ and } t = 1, \text{ then } \mathbf{w}^{new} = \mathbf{w}^{old} + \delta\mathbf{x}. \end{aligned}$$

At each round (epoch) of training, all features of an instance together contribute to weight modification, unlike in decision tree induction and Naive Bayesian Classification. In other words, features are used in a multivariate fashion. In classifying new instances, all these different learning algorithms need to use all the selected features in classification. The difference between univariate and multivariate lies only in the training phase in which we are concerned about how features are used to build a classifier.

Perceptrons are just one type of simplest neural networks. They have limited functions. Multi-layer perceptrons and other types of neural networks can accomplish complicated supervised and unsupervised learning (Hecht, 1990, Hagan et al., 1996). These topics are out of the scope of this writing.

2.4.4 Summary

Different classifiers have different biases and use features differently (univariate or multivariate). Three above mentioned classifiers show their unique ways of using features. Can't we use classifiers as feature selector? And which type of classifiers is the best feature selector? If the answer is yes to the first question, the derived feature selection rule is to choose features that remain in the final classifier. A decision tree classifier may only need a smaller number of features after training, but for the other two classifiers, it is not obvious that the feature selection rule can be applied directly since all features are involved in both training and classification[2]. Since the way in which features are used in a classifier affects the way of feature selection, we have reasons to believe that the choice of a feature selection algorithm depends on our need for performing feature selection. This leads us to the next section.

2.5 FILTER VS. WRAPPER MODELS

The simplest way of feature selection is to use the classifier's accuracy as the performance measure. Some researchers argued (Siedlecki and Sklansky, 1988, Kohavi, 1995) that if our objective is to minimize the classifier error rate, and the measurement cost for all the features is equal, then the classifier's predictive accuracy is most appealing. That is, we should build a classifier with an aim to achieve the highest predictive accuracy possible; and select the features used by the classifier as the optimal features. This model is so called the *wrapper* model.

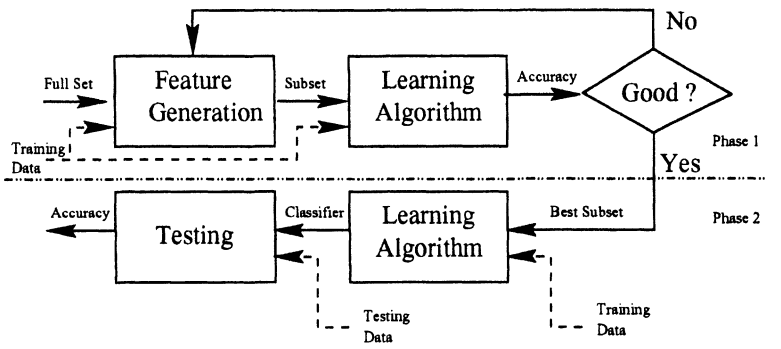


Figure 2.4. A wrapper model of feature selection.

However, for many reasons, extensive research effort (Ben-Bassat, 1982, Blum and Langley, 1997, Dash and Liu, 1997) has been devoted to the investigation of other indirect performance measures, mostly based on distance and information measures, in selecting features. This model is called the *filter* model. We discuss the two models in the context of two related approaches: machine learning vs. data mining.

2.5.1 A machine learning approach

The major concern of machine learning is to achieve high predictive accuracy. The problem is how we can improve a *classifier*'s performance on learning from the *data*. Since the classifier has been chosen, one way to improve its performance is to process the data through feature selection, or to provide the classifier with the data of better quality[3]. If we can select relevant features and remove noise, we may be able to improve a classifier's performance (accuracy, in particular).

The wrapper model of feature selection can achieve the purpose. Seen in Figure 2.4, a wrapper model consists of two phases: phase 1 - feature subset selection, which selects the best subset using a classifier's accuracy (on the training data) as a criterion; phase 2 - learning and testing, a classifier is learned from the training data with the best feature subset and tested on the test data. When feature subsets are systematically generated (following the chosen search direction), for each subset of features, a classifier is generated from the data with chosen features. Its accuracy is recorded and the feature subset with the highest accuracy is kept. When the selection process terminates, the subset

with the best accuracy is chosen. Phase 2 is the normal learning and testing process in which we obtain the predictive accuracy on the test data. Since we maintain only the best subset of features obtained in Phase 1, we need to re-learn the classifier associated with the best subset.

As the estimated accuracy of a classifier on the training data may not reflect its accuracy on the test data, the key issue here is how to truly estimate the accuracy. One way is to use cross validation (Weiss and Kulikowski, 1991) (more on this in Chapter 5 when we discuss issues of evaluation). Doing so will lengthen the feature selection process. However, the wrapper model seems the best way to improve a classifier's performance since both learning a classifier and selecting features use the same bias (John et al., 1994, Aha, 1998). Notwithstanding the difficulty of estimating the true accuracy, our goal of improving a classifier could be more than increasing accuracy. In such a case, we may have to consider alternative approaches. Another limitation with the wrapper model is about the choices of classifiers. Since in phase 1, a classifier is built as many times as the number of feature subsets generated, computation intensive classifiers should not be considered. Researchers often use heuristic learning methods like Naive Bayesian Classifiers or Decision Tree Induction (Kohavi and John, 1998).

2.5.2 A data mining approach

More often than not, there are other reasons to select features: such as noise removal, data reduction, in addition to increasing a classifier's predictive accuracy. Also, as we know, a wrapper model's capability of handling data with high dimension is limited by the chosen classifier. However, in the context of data mining, it is often a case that data size is huge and a classifier cannot directly be applied to the data. Therefore, it is necessary to pre-process the data via other means. As we also know that different classifiers have their unique biases, it is not wise to learn a classifier using the features selected by another classifier. For example, features selected by a decision tree induction algorithm may not be suitable for learning a neural network, and vice versa. Though the wrapper model can ensure the best accuracy of selected features, because of its expensive time complexity, limited choices of classifiers and incapability of handling huge sized data, researchers study feature selection methods of the filter model.

A filter model of feature selection also consists of two phases (Figure 2.5): phase 1 - feature selection using measures such as information, distance, dependence, or consistency, and no classifier is engaged in this phase; and phase 2 is the same as in the wrapper model, a classifier is learned on the training data with the selected features and tested on the test data.

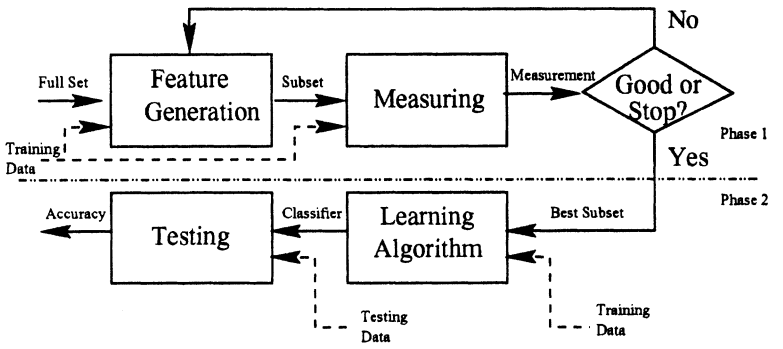


Figure 2.5. A filter model of feature selection.

The filter model has several characteristics: 1. it does not rely on a particular classifier's bias, but on the intrinsic properties of the data, so the selected features can be used to learn different classifiers; 2. measuring information gains, distance, dependence, or consistency is usually cheaper (in time complexity) than measuring accuracy of a classifier, so a filter method can produce a subset faster, other things being equal; and 3. because of the simplicity of the measures and low time complexity, a filter method can handle larger sized data than a classifier can; so in the case where a classifier cannot directly be learned from the large data, it can be used to reduce data dimensionality so that the classifier can be learned from the data with reduced dimensionality. However, there is a danger that the features selected by a filter model cannot allow a learning algorithm to fully exploit its bias.

2.6 A UNIFIED VIEW

Examining the filter and wrapper models (Figures 2.5 and 2.4), we notice that they have strong resemblance in phase 1, and are exactly the same in phase 2. With the understanding of previous section on search, selection measures, manner of feature selection, we can generalize all these aspects into a unified model. The model accounts for four parts: (1) feature generation, (2) feature evaluation, (3) stopping criteria, and (4) testing, as in Figure 2.6.

Part (4) is essentially phase 2 in the filter and wrapper models. It is rather general. Basically it learns a classifier from the training data with specified features and then the classifier is tested on the test data with these features. Depending on how the testing is performed (i.e., cross validation, boot strapping, etc.), estimated accuracy is obtained. This would enable us to see whether

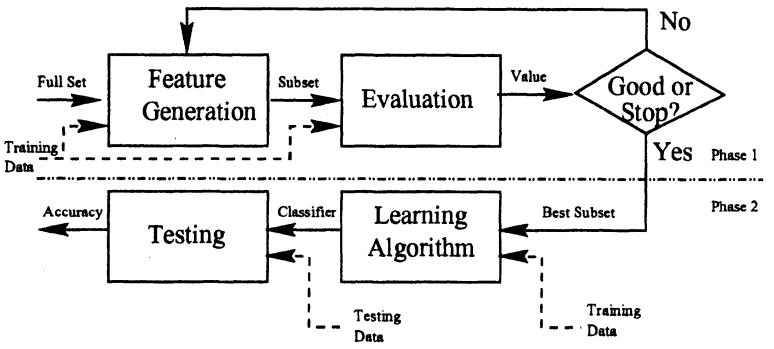


Figure 2.6. A unified model of feature selection.

feature selection can really help compared with using the full set of features. Part (3) is about stopping criteria. Phase 1 of feature selection stops under one of the three conditions: (a) when a subset of features is good enough according to an evaluation criterion; in the case of a wrapper model, for example, it is good enough if estimated accuracy is better than the accuracy obtained using the full set features[4]; (b) when some bound is reached, where a bound can be the minimum number of features we need, or the maximum number of subsets that can be generated, etc., and (c) the search completes. Stopping criteria vary under different circumstances: for some criteria, one need to set some thresholds *a priori*, and for others, thresholds are determined by algorithms.

Part (2) is evaluation of each feature subset using some measures. Equating a classifier with a measuring function and leaving the choice of a measure to a lower or more detailed level, the difference between a wrapper model and a filter model diminishes. The search of optimal feature subsets and the manner how features are used are clamped into Part (1) - feature generation. Following a certain search strategy, a feature or a feature subset is generated each time.

At an abstract level that hides some details about search and evaluation, feature selection can be viewed as a process with the above four parts. Every time the search space expands, a subset is generated in Part (1) and evaluated in Part (2). The search space expands until some stopping criterion is satisfied. Then the best subset of feature is output to phase 2 for inducing a classifier and testing the effect of selected features.

2.7 CONCLUSION

We discuss many important perspectives of feature selection: various search algorithms, selection criteria, univariate and multivariate feature generation and evaluation, filter vs wrapper models of feature selection. At the end, we have arrived at a unified model of feature selection. This allows us in the following chapters to independently analyze and discuss topics like feature generation, feature evaluation, and stopping criteria. It paves the way for us to understand various and numerous existing feature selection algorithms and look to new algorithms that may have been omitted.

Though we are conceptually well equipped at this point, as we know, we have left out many details in the unified model, such as combinations of search strategies and evaluation measures, influences of data types on different feature selection algorithms, choices of using various feature selection algorithms, etc. These are the topics of the subsequent chapters.

References

Aha, D. W. (1998). *Feature Weighting for Lazy Learning Algorithms*, pages 13–32. In (Liu and Motoda, 1998).

Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305.

Ben-Bassat, M. (1982). Pattern recognition and reduction of dimensionality. In Krishnaiah, P. R. and Kanal, L. N., editors, *Handbook of statistics-II*, pages 773–791. North Holland.

Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271.

Blumer, A., Ehrenfeucht, A., Haussler, D., and Warmuth, M. (1990). Occam's razor. In Shavlik, J. and Dietterich, T., editors, *Readings in Machine Learning*, pages 201–204. Morgan Kaufmann.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software.

Dash, M. and Liu, H. (1997). Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3).

Dietterich, T. (1997). Machine learning research: Four current directions. *AI Magazine*, pages 97–136.

Domingos, B. and Pazzani, M. (1996). Beyond independence: Conditions for the optimality of the simple Bayesian classifier. In Saitta, L., editor, *Machine Learning: Proceedings of Thirteenth International Conference*, pages 105–112. Morgan Kaufmann Internationals, Inc.

Fayyad, U., Piatetsky-Shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery: An overview. In Fayyad, U., Piatetsky-Shapiro, G.,

Smyth, P., and Uthurusamy, R., editors, *Advances in Knowledge Discovery and Data Mining*, pages 495–515. AAAI Press / The MIT Press.

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

Hagan, M., Demuth, H., and Beale, M. (1996). *Neural Network Design*. PWS Publishing Company.

Hecht, R. (1990). *Neurocomputing*. Addison-Wesley Pub. Company.

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher.

Kira, K. and Rendell, L. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 129–134. AAAI Press/The MIT Press.

Kohavi, R. (1995). *Wrappers for performance enhancement and oblivious decision graphs*. PhD thesis, Department of Computer Science, Standford University, Stanford, CA.

Kohavi, R. and John, G. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324.

Kohavi, R. and John, G. (1998). *The Wrapper Approach*, pages 33 - 50. In (Liu and Motoda, 1998).

Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In Saitta, L., editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 284–292. Morgan Kaufmann Publishers.

Liu, H. and Motoda, H., editors (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Liu, H. and Setiono, R. (1996). A probabilistic approach to feature selection - a filter solution. In Saitta, L., editor, *Proceedings of International Conference on Machine Learning (ICML-96)*, pages 319–327. Morgan Kaufmann Publishers.

Mingers, J. (1989a). An empirical comparison of pruning methods for decision tree induction. *Machine Learning*, 4:227–243.

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

Narendra, P. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Trans. on Computer*, C-26(9):917–922.

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

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

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*. Prentice Hall.

Schlimmer, J. C. (1993). Efficiently inducing determinations : a complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 284–290.

Siedlecki, W. and Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems That Learn.* Morgan Kaufmann Publishers, San Mateo, California.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, pages 73–83.

Notes

1. In practice, one may need more stopping criteria to handle various cases. Refer to (Quinlan, 1993).

2. Some researchers (Almuallim and Dietterich, 1994) argued that in some cases, one should not rely on a decision tree classifier to filter out irrelevant features. There exist neural network feature selectors, see one example in (Setiono and Liu, 1997).

3. Researchers recently studied methods like bagging and boosting to enhance a classifier's performance (Dietterich, 1997, Domingos, 1997). Bagging forms an ensemble of classifiers. Boosting massages data through learning many versions of a classifier

4. There could be many subsets of features with which accuracy is better than that with the full set of features. We select the subset of features with the best accuracy.

3 FEATURE SELECTION ASPECTS

With a unified model of feature selection, we are ready to discuss in detail different aspects of feature selection. The major aspects of feature selection are (1) search directions (feature subset generation), (2) search strategies, and (3) evaluation measures. The objective of this chapter is two-fold: (a) to study the various options for each aspect in a systematic and principled way and (b) to identify the essential and different characteristics of various feature selection systems.

3.1 OVERVIEW

The study of different perspectives of feature selection manifests that search and measure play dominant roles in feature selection; stopping criteria are usually determined by a particular combination of search and measure. Since search is about search directions and search strategies, with evaluation measures, we come up with a 3-dimensional structure, in Figure 3.1, that categorizes all possible feature selection algorithms in terms of search and measure.

There are 27 possible combinations of feature selection methods, considering all the possibilities. We will elaborate on each possibility along every dimension

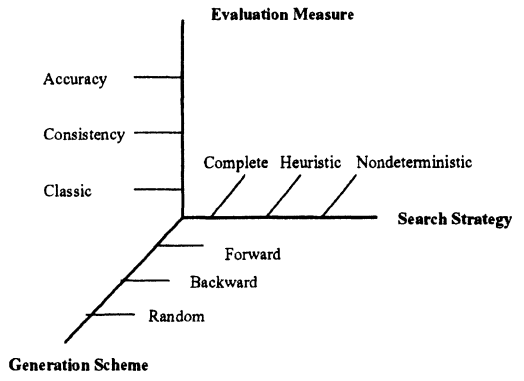


Figure 3.1. Three principal dimensions of feature selection: search strategy, evaluation measure, and feature generation scheme.

later in this chapter and on individual combinations in Chapter 4. From the point of view of a method's output, however, these methods can be grouped into just two categories. One category is about ranking features according to some evaluation criterion; the other is about choosing a minimum set of features that satisfies an evaluation criterion. We first discuss these two categories.

Feature Ranking Algorithms. In this category of feature selection algorithms, one can expect a ranked list of features which are ordered according to evaluation measures. A measure can be any of accuracy, consistency, information, distance, and dependence. In other words, an algorithm of this type does not tell you what is the minimum set of features, but only the importance (relevance) of a feature compared to others.

The idea is to evaluate each individual feature with a measure. This evaluation results in a value attached to a feature. Features are then sorted according to the values. The run time complexity of this simple algorithm (summarized in Table 3.1) is $O(N \times n + N^2)$ where N is the number of features, n the number of instances: $O(n)$ for step (1); $O(N)$ for step (2); and the **for** loop repeats N times. Many variations of this algorithm lead to different feature selection methods. What is common to these feature ranking methods is a ranked list of features.

One can simply choose the first M features for the task at hand if he knows what M is. It may not be so straightforward if M is unknown. Variants of this algorithm have been designed to overcome the unknown M problem. One

Table 3.1. A univariate feature ranking algorithm.

Ranking Algorithm
Input: **x** - features, U - measure

initialize: list $L = \{\}$ /* L *stores ordered features**/
for each feature x_i, $i \in \{1, \ldots, N\}$
begin
(1) $v_i =$ `compute`(x_i, U)
(2) position x_i into L according to v_i
end

Output: L in which the most relevant feature is placed first.

solution is to use the evaluation measure repeatedly to build a decision tree classifier (Section 2.4.2 of Chapter 2); the features used in the decision tree are selected. This variation can solve the unknown M problem. This treatment conceptually blurs the demarcation between feature ranking algorithms and minimum subset algorithms. Some researchers did use this method to select features. However, it may not be wise to use this on the data in which some irrelevant feature is strongly correlated to the class feature (John et al., 1994). It is not recommended by (Almuallim and Dietterich, 1994) if the goal is to find the minimum feature subset.

Minimum Subset Algorithms. It is quite often that one does not know the number of relevant features. In order to search for it, one may need to design other types of algorithms since feature ranking algorithms can only order features. Thus we have this category of minimum subset algorithms. An algorithm in this category returns a minimum feature subset and no difference is made for features in the subset. Whatever in the subset are relevant, otherwise irrelevant.

The **Min-Set** algorithm can be found in Table 3.2. The `subsetGenerate()` function returns a subset following a certain search method, in which a stopping criterion determines when `stop` is set to *true*; function `legitimacy()` returns *true* if subset S_k satisfies measure U. Function `subsetGenerate()` can take one of the generation schemes.

Here we introduce two types of algorithms from the viewpoint of the end results of feature selection. The details vary when we discuss each of the three dimensions (generation scheme, search strategy, and evaluation measure) for feature selection. In the following sections, we will elucidate each possibility

Table 3.2. A minimum subset algorithm. # - a cardinality operator returns the number of features in a set.

Min-Set Algorithm
Input: **x** - features, U - measure

initialize: $S = \{\}$ /* *S holds the minimum set**/
stop = *false*;
repeat
(1) S_k = `subsetGenerate(x)` /* `stop` can be set here*/
if
(2) `legitimacy`(S_k, U) is *true* and $\#(S_k) < \#(S)$
then
(3) $S = S_k$ /* *S is replaced by* S_k*/
until `stop` = *true*

Output: S - the minimum legitimate subset of features.

along a single dimension (focusing on one dimension at a time) by giving algorithms and examples.

3.2 BASIC FEATURE GENERATION SCHEMES

We now look at the ways in which feature subsets are generated. For instance, one can start with an empty set, and fill in the set by choosing the most relevant features from the original features. Or one can begin with the original full set, and remove irrelevant features. Among many variations, we choose three basic schemes (sequential forward, sequential backward and random) plus a combination of the first two, and others can be obtained by varying some parts of these basic schemes.

3.2.1 Sequential forward generation

This scheme starts with an empty set S and adds features from the original set F into S sequentially, i.e., one by one. Obviously, if one wants to obtain a ranked list, this scheme can provide that. In Table 3.3, some algorithmic details are given. The idea is *sequential forward generation* (hence **SFG**). At each round of selection, the best feature f is chosen by `FindNxt`(F). f is added into S and removed from F. So, S grows and F shrinks. There are two stopping criteria. If one wants to have a minimum feature subset, he can rely

on the first stopping criterion "S satisfies U". The generation terminates at the moment when S satisfies U - the evaluation measure. S is the minimum subset according to U. If one would like to have a ranked list, he can use only the second stopping criterion "$F = \{\}$". As we know, the first feature chosen is the most relevant, and the last feature chosen the least relevant. By adopting only the second stopping criterion, one can indeed obtain a ranked list. The second stopping criterion ensures that the algorithm should stop when the evaluation measure U cannot be satisfied even if all attributes are chosen by S (therefore F is empty). This is quite often when noise is present in the data.

Table 3.3. Sequential forward feature set generation - **SFG**.

SFG Scheme
Input: F - full set, U - measure

initialize: $S = \{\}$ /* *S stores the selected features* */
repeat
(1) $f =$ `FindNxt`(F)
(2) $S = S \cup \{f\}$
(3) $F = F - \{f\}$
until S satisfies U or $F = \{\}$

Output: S

SFG is just a simple scheme for forward feature set generation. Other forms can be easily obtained by varying statement (1) $f =$ `FindNxt`(F). For instance, if one suspects that two features together may play a more influential role (individually they may not), then he may modify `FindNxt()` into `FindNxt2()`, or finding up to the best two features at a time (Liu and Wen, 1993), i.e., having one more step of look-ahead. In order to apply `FindNxt2()`, we need to try $\binom{N}{1}$ plus $\binom{N}{2}$ combinations in order to find the best one or two features. Let's save the chosen features in S_1, the remaining features in S_2. Then, we choose from S_2 another one or two features in order to find next best features, which is determined by the value of U after combining these newly selected features with the ones in S_1. In theory, one can go up to N-step look-ahead and check all the combinations in order to choose the best subset. The more the steps of look-ahead, generally, the better a solution is. Nevertheless, that would take enormous computing time since the running time needed is exponentially proportional to the number of features N, or $O(2^N)$. Hence, the one-step look-ahead version of **SFG** is among the most commonly used schemes in subset generation because of its efficiency.

3.2.2 Sequential backward generation

This is a scheme which is quite an opposite of **SFG**. As shown in Table 3.4, the search starts with the full set F and finds a feature subset S by removing one feature at a time, so is named as **SBG** - sequential backward generation. Instead of finding the most relevant feature, `GetNxt`(F) looks for the least relevant feature f which is then removed from F. Although, it is the same as in **SFG** that F shrinks and S grows, but it is F that is the subset sought out. S only stores the irrelevant features, which may or may not be useful. If the full set is the minimum set itself, **SBG** will identify this in one loop. Otherwise, $F \cup \{f\}$ output by **SBG** is the minimum set that can satisfy U since F cannot satisfy U. The second stopping criterion "$F = \{\}$" ensures that the algorithm should stop if all instances in the data have the same class value. One may notice that unless F is empty, features in F are not ranked. In other words, we have no idea which features are more relevant than the other. Hence, it is not a suitable scheme that gives a ranked list, although features in S are ranked according to their irrelevancy.

Table 3.4. Sequential backward feature generation - **SBG**.

SBG Scheme
Input: F - full set, U - measure

initialize: $S = \{\}$ /* *S holds the removed features*/*
repeat
(1) $f =$ `GetNxt`(F)
(2) $F = F - \{f\}$
(3) $S = S \cup \{f\}$
until F does not satisfies U or $F = \{\}$

Output: $F \cup \{f\}$

In the same spirit of `FindNxt2`(), one may try `GetNxt2`() or its other variations. The similar run time complexity analysis applies, and **SFG** and **SBF**'s run time complexities are of the same order. Because of the nature of sequential removal, the minimal subset obtained in this scheme may not be the absolute minimal subset (i.e., the optimal one). One way to get the absolute minimal subset is by trying all the combinations.

3.2.3 *Bidirectional generation*

As we discussed briefly in Chapter 2, **SFG** and **SBG** complement each other. When the number of relevant features is smaller than $N/2$, **SFG** is quicker, if the number of relevant features (M) is greater than $N/2$, then **SBG** is faster. Without knowing the value of M, one would not have a clue which scheme between the two is better. Hence, born is this bidirectional generation scheme, shown in Table 3.5. The basic implementation of this scheme (**FBG** - forward and backward generation) is running schemes **SFG** and **SBG** in parallel. **FBG** stops if either (**SFG** or **SBG**) finds a satisfactory subset. A flag is needed to tell which one has found an optimal subset.

Table 3.5. Bidirectional feature set generation - **FBG**.

FBG Scheme

Input: F_f, F_b - full set, U - measure

initialize: $S_f = \{\}$, /* S_f *holds the selected features**/
$S_b = \{\}$ /* S_b *holds the removed features.**/

repeat

(1) $f_f =$ **FindNxt**(F_f) $f_b =$ **GetNxt**(F_b)

(2) $S_f = S_f \cup \{f_f\}$ $F_b = F_b - \{f_b\}$

(3) $F_f = F_f - \{f_f\}$ $S_b = S_b \cup \{f_b\}$

until (a) S_f satisfies U or $F_f = \{\}$ or
(b) F_b does not satisfies U or $F_b = \{\}$

Output: S_f if (a) or $F_b \cup \{f_b\}$ if (b)

3.2.4 *Random generation*

FBG's putting two sequential generation schemes together can help in getting a valid subset fast on average, but cannot help in finding an *absolute* minimum valid set. The sheer reason is that all the above three schemes adopt a hill-climbing heuristic hoping that by selecting the best (as in **SFG**) or removing the worst (as in **SBG**) sequentially, an absolute minimal subset (optimal) will emerge. Doing so will surely speed up the selection process, but if they hit a local minimum (a best subset at that moment), they have no way to get out because what has been removed cannot be added (as in **SBG**) and what has been added cannot be removed (as in **SFG**). This is one fundamental problem with a sequential method. Put it another way, the three schemes above cannot

guarantee to find the optimal feature subset. As opposed to this fixed rule of sequential generation, we can resort to random feature generation, hoping to avoid getting stuck at some local minima. This random generation scheme produces subsets at random. Basically one needs a good random number generator and tailors it to **RandGen**(F) so that every combination of features F, ideally, has a chance to occur and occurs just once. Several implementations of random number generators can be found in (Press et al., 1992). This scheme is summarized as S = **RandGen**(F) in Table 3.11 where S is a subset of features.

Although we introduce four schemes for feature generation in this section, there are three basic ones, the bidirectional feature generation is a combination of sequential forward and backward feature generation. Later on throughout the book, we consider the basic three schemes unless otherwise specified.

3.3 SEARCH STRATEGIES

In the preceding section, we mentioned an issue of optimal subsets when we discussed the feature generation schemes. This can be further elaborated in terms of search strategies. Roughly speaking, the search strategies can be of three types: (1) complete, (2) heuristic, and (3) nondeterministic. In this section, we will describe the characteristics of these search strategies and their usages. Then, for each search strategy, we take into account of the three basic feature generation schemes. As a consequence, we find a big variety of search methods at our disposal. In the following, we put feature selection into the perspective of search. The search space consists of all the combinations of feature subsets. The size of the search space is 2^N where N is the number of features.

3.3.1 Complete search

Exhaustive search is complete since it covers every combination of N features, $\binom{N}{1}, \binom{N}{2}, \ldots$ Under certain circumstances, search can be complete but not exhaustive. For instance, if an evaluation measure is monotonic, Branch & Bound (Narendra and Fukunaga, 1977) is complete search that guarantees an optimal subset. The difference between complete and exhaustive search is in that exhaustive search is complete, but complete search is not necessarily exhaustive. So we use complete search for both kinds of search. A measure U is monotonic if for any two subsets S_1, S_2, and $S_1 \subseteq S_2$, then $U(S_1) \geq U(S_2)$.

Two classic exhaustive search implementations are *depth-first search* and *breadth-first search*. Both types of search can be forward or backward in feature generation. In a forward generation scheme, it starts with an empty set, then considers the possible subsets of one feature, two features, etc. subsequently. This process is reversed for a backward generation scheme, starting with the

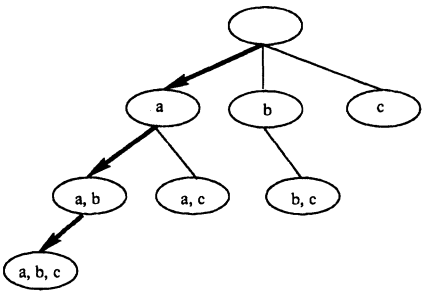


Figure 3.2a. Depth-first search illustration with three features a, b, c.

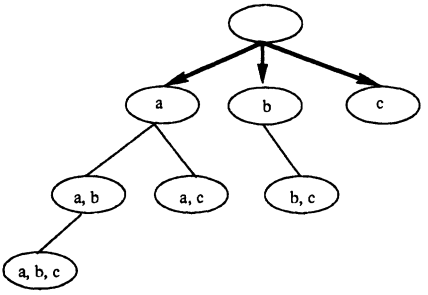


Figure 3.2b. Breadth-first search illustration with three features a, b, c.

full set of features. Regardless of their directions, the essence of the two types of search is the systematic examination of every possible subset. The difference is how the systematic search is carried out. We adopt the forward generation scheme in the explanation below. Assume that there are three features a, b, c in the full set F. We can avoid the duplicates of a subset by following an enumeration procedure. With the enumeration procedure, we can create a search tree instead of a search lattice. Since exhaustive search is under examination, we will visit all combinations of features (8 states including the empty set). As shown in Figures 3.2a and 3.2b, the depth-first search goes down one branch, backtracks to another branch until all braces are generated and checked, while the breadth-first search does it layer by layer, checking all subsets with one feature, then with two features, so on and so forth. Both search algorithms (**DEP** and **BRD**) are shown in Tables 3.5a, 3.5b and 3.6.

Two implementations of depth-first search are presented. One uses an explicit *stack* data structure; and the other employs implicit run-time stack. A stack is a last-in-first-out data structure. Breadth-first search is implemented using a *queue* data structure that enqueues subsets at each round of feature generation, dequeues the first subset in the queue for consideration to expand the search space. As in the example, the queue contains (a, b, c, ab, ac, bc, abc) if no node[1] is dequeued. When one works on a particular implementation, one should also give some consideration about the enumeration procedure. The working of the enumeration procedure is determined by the feature generation scheme, for instance, one needs to add one feature (choosing from F) at a time in the forward generation scheme, and to remove one feature at a time in the backward generation scheme. In Tables 3.5a and 3.5b, however, there is no explicit mentioning about the enumeration procedure for the sake of simplicity.

Table 3.5a. Exhaustive search: depth first - **DEP** with explicit use of a stack.

DEP Algorithm 1
Input: F - full set, S - stack, U - measure
initialize: *node* = null
S = null
DEP (*node*)
{
if *node* is the best subset so far w.r.t. U
Set = *node*
for all children C of *node*
push (C, S)
while (notEmpty(S)) {
node = pop (S)
DEP (*node*) }
}
Output: *Set*

Now we have a good picture of exhaustive search, let's look at an example of complete search: Branch & Bound search (Narendra and Fukunaga, 1977), and examine the difference between the two types of search. Branch & Bound is a variation of depth-first search of a lattice. Without any restriction, therefore, it is exhaustive search. With a given bound β, however, the search stops at a node whose measure is greater than β, and the branches extended from the node are pruned. The search backtracks and another branch will be searched. As seen in Figure 3.3, Branch & Bound usually adopts a backward feature generation

Table 3.5b. Exhaustive search: depth first - **DEP** without explicit use of a stack.

DEP Algorithm 2
Input: F - full set, U - measure
initialize: *node* = null
DEP (*node*)
{
if *node* is the best subset so far w.r.t. U
Set = *node*
for all children C of *node*
DEP (C)
}
Output: *Set*

Table 3.6. Exhaustive search: breadth first - **BRD**.

BRD Algorithm
Input: F - full set, Q - queue, U - measure
initialize: *node* = null
Q = null
BRD (*node*)
{
for all children C of *node*
enqueue(C, Q)
while (notEmpty(Q)) {
node = dequeue(Q)
if *node* is the best subset so far w.r.t. U
Set = *node*
BRD (*node*) }
}
Output: *Set*

scheme and begins with a full set of features. The success of Branch & Bound depends on a monotonic evaluation measure U. Let's take U as the number of errors made by a classifier. The evaluation of each subset S results in a value of U. If β is 12, as in the example shown in Figure 3.3, the search starts from the left-most branch, the value of subset (a, b) is 13, which is greater than β. Hence it backtracks to subset (a, c). Its value is 12, so the search continues.

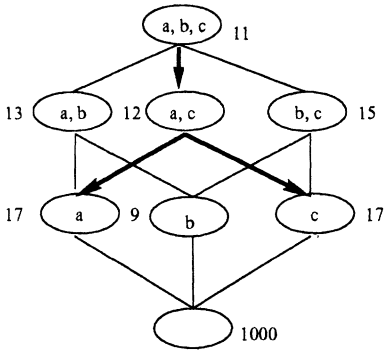


Figure 3.3. Branch & Bound search illustration with three features a, b, c. Numbers beside nodes are their costs.

Since the next two subsets (a) and (c) have higher values (17) the search stops, and (a, c) is recorded as the best subset so far found. The search continues to (b, c) but its value is greater than β, and the algorithm returns (a, c) as the best subset. As we see in the figure, the best subset should be (b) since it has the lowest U. Branch & Bound could not find it because the measure U used is obviously not monotonic (one case countering the monotonicity of U is that its value of subset (b, c) is greater than that of subset (c)). If U were monotonic, it would be guaranteed that the search is complete and subset (b) would be found. This example (Figure 3.3) illustrates the concept of Branch & Bound and shows the importance of a monotonic measure. The algorithm is described in Table 3.7. Later on, in Section "Evaluation Measures", various measures including monotonic ones will be introduced.

For exhaustive search, there is no parameter to be specified by a user. However, this situation changes when one tries to avoid exhaustive search and opts for complete search, more often than not, one needs to provide values for some parameters. In this Branch and Bound algorithm, for instance, the bound is supplied by the user. "How reasonable a bound is" plays a pivotal role. As is well known, the setting of values for parameters requires some sort of knowledge or educated guesses. In (Siedlecki and Sklansky, 1988), they showed some heuristic methods of finding a good bound. Later on, we will introduce another version of branch and bound which automatically sets the bound; the algorithm is named as ABB (Table 4.3). The reader may notice that the random feature generation scheme is not mentioned so far. This is because in discussing com-

Table 3.7. Complete search: Branch & Bound - **BAB**.

BAB Algorithm
Input: F - full set, Q - queue, U - measure
β - bound for some value of U

initialize: $node = F$
best $= \beta$
BAB ($node$)
{
for all children C of $node$ {
if (C's value of $U < \beta$) {
/* else, the branch starting with C is pruned*/
if (C's value $\leq$ best's value)
best $= C$ /* remember the best subset*/
BAB (C)
}
}
}
Output: **best** /* the best set w.r.t. U */

plete search, it is natural for us to exclude any randomness as the search must be at least complete.

3.3.2 Heuristic search

Complete search strategies are usually time consuming. Can we make some smart choices based on the minimum information available, but without looking at the whole picture? This is all what heuristic search is about. The rationale of this non-optimal strategy is three-fold: (1) it is quick to find a solution (i.e., a subset of features); (2) it usually can find near optimal solution if not optimal; and (3) the trade-off of optimality with speed is often worthwhile because of much gained speed and little loss of optimality (recall that an optimal subset is a minimum one that satisfies the evaluation criterion). Heuristic feature selection algorithms abound. We introduce three of them (best-first, beam, and approximate branch & bound search) in relation with depth-first, breadth-first, and Branch & Bound.

Best-first search is derived from breadth-first search. Instead of branching on all nodes at each layer, best-first search expands its search space layer by layer, evaluates all *newly* generated subsets (the child nodes of the root in the

beginning), chooses *one best* subset at each layer to expand, and repeat this process until no further expansion is possible. Some researchers call this type of search myopic since it only cares what is the best at each step. In other words, it is a one-step look-ahead strategy. A natural modification is to look ahead a few more steps. Doing so may improve the quality of chosen subsets, but the run time will be increased exponentially with the increased steps of look-ahead. The reader can refer to the feature generation schemes such as `FindNxt()` and `FindNxt2()` in Section 3.2. The optimality of a subset is guaranteed only when exhaustive search is in place. Table 3.8 shows the algorithm for best-first search. An example is also shown in Figure 3.4 in which a path is depicted. It is clear that the run time complexity of best-first search is much lower than $O(2^N)$ where N is the total number of available features. As a matter of fact, the run time complexity is $O(mN)$ where m is the maximum number of children a node can have, and m is always smaller than N. A node A is a child node of another node B if and only if (1) $|\#(A) - \#(B)| = 1$ and (2) A is different from B in value by one feature. As in Figure 3.4, (a) is a child node of the empty set (the root), (a, b) is a child of (a), (a, b, c) is a child of (a, b), but not of (a) which is a grandparent node.

Table 3.8. Heuristic search: best-first - **BEST**.

BEST Algorithm
Input: F - full set, Q - queue, U - measure
initialize: *node* = null
Q = null
BEST (*node*)
{
for all children C of *node*
C_{best} is the best among all C's w.r.t. U
enqueue(C_{best}, Q)
while (notEmpty(Q)) {
node = dequeue(Q)
if *node* is the best subset so far w.r.t. U
Set = *node*
BEST (*node*) }
}
Output: *Set*

Beam search can be understood as an extension of best-first search, or a limited version of breadth-first search. In best-first search, we only choose the best subset at that moment and proceed further. How about choosing the best

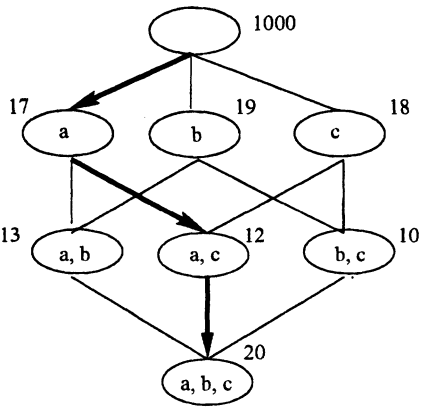


Figure 3.4. Best-first search illustration with three features a, b, c.

two subsets instead of one? In Figure 3.5, we show a case of beam search that chooses two best subsets to expand at every layer. As in Table 3.9, one can set the value of η to 2, 3, or up to N. Beam search is in effect breadth-first search when $\eta = N$. The reason we should consider more subsets is simple: we may not derive the best subset from the best two subsets at earlier steps (Cover, 1974). By relating beam search to breadth-first search, we clearly witness another example of trading time with optimality. Beam search's run time complexity is between best-first search and breadth-first search.

Approximate Branch & Bound is proposed by (Siedlecki and Sklansky, 1988) as an extension to Branch & Bound in the realization that it is rare to have a monotonic evaluation measure. The algorithm is regenerated in Table 3.10. As was shown in Figure 3.3, the best subset was missed due to the non-monotonicity of the measure. Can we still reach the best set? The answer is to relax bound β by δ so it allows Branch & Bound to continue the search when β is reached. In our example (first shown in Figure 3.3), now in Figure 3.6), if $\delta = 1$ (recall that $\beta = 12$), search does not stop at subset (a, b). This leads to the finding of the best subset (b) whose U is 9. At the first glance, one may think why not just take a larger β. It would achieve the same effect. However, conceptually, they are different. Introducing δ is a controlled relaxation. As we know that β is determined according to some prior knowledge, so after it is set, it should be left alone. δ is something linked with the monotonicity of a measure. When one suspects a measure's monotonicity, by varying δ, he can get a feeling about the measure. This is a heuristic way of overcoming

Table 3.9. Heuristic search: beam search - **BEAM**.

BEAM Algorithm
Input: F - full set, Q - queue, U - measure
η - a number of best children to evaluate
initialize: *node* = null
Q = null
BEAM (*node*)
{
for all children C of *node*
find η best C_{best}'s among all C's w.r.t. U
for all C_{best}'s
enqueue(C_{best}, Q)
while (notEmpty(Q)) {
node = dequeue(Q)
if *node* is the best subset so far w.r.t. U
Set = *node*
BEAM (*node*) }
}
Output: *Set*

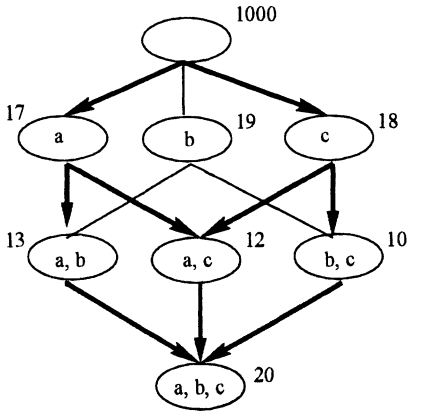


Figure 3.5. Beam search illustration with three features a, b, c, $\eta = 2$.

the non-monotonicity of a measure. Therefore, Approximate Branch & Bound does not aim to speed up, but to find the optimal set in case of a non-monotonic measure. It runs even longer than Branch & Bound.

Table 3.10. Heuristic search: approximate branch & bound - **ABAB**.

ABAB Algorithm

Input: F - full set, Q - queue, U - measure
β - bound for some value of U
δ - allowable deviation from β

initialize: $node = F$
`best` $= \{\}$
ABAB ($node$)
{
for all children C of $node$ {
if (C's value of $U \leq \beta + \delta$) {
/* else, the branch starting with C is pruned*/
if (C's value < best's value)
`best` $= C$ /* remember the best subset*/
ABAB (C)
}
}
}
Output: `best` /* the best set w.r.t. U */

3.3.3 Nondeterministic search

For both complete and heuristic search strategies, they share one property in common, i.e., they are all *deterministic*. That means no matter how many times one runs a particular algorithm, he can expect that the solution from any subsequent run is always the same as that of the first run. Here we introduce a set of algorithms that share another property - *nondeterministic*. For such an algorithm, one should not expect the same solution from different runs. One of the major motivations for developing this sort of algorithms is to avoid getting stuck in local minima as in heuristic search. Another motivation is to capture the interdependence of features which heuristic search is also incapable of capturing (See Figure 3.4 where the optimal solution (b, c) is missed).

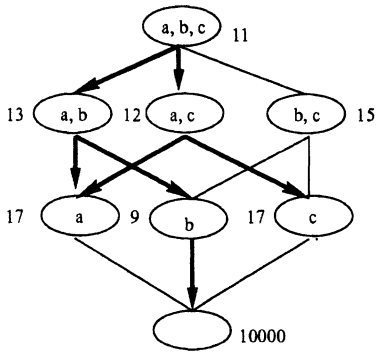


Figure 3.6. Approximate Branch & Bound search illustration with three features a, b, c.

We use one such algorithm shown in Table 3.11 to explain the working of nondeterministic search. **RAND** keeps only the current best subset that satisfies U as well as has the smallest cardinality. As **RAND** continues, it can only produce a better subset. Statement "print S_{best}" reports a better subset found. **RAND** is an any-time algorithm (Boddy and Dean, 1994). It means that one need not wait until the end of **RAND**'s running in order to find a subset. But can it find optimal subsets? If it is allowed a sufficiently long running period and armed with a good random function, yes, it can. This leads us to the question when it should stop. The problem with this scheme is that we don't know if we have found the optimal subset, we only know that if there is a better one than S_{best}, **RAND** will find it. In some cases, one cannot allow a program to run forever, therefore, a stopping criterion can be a number of maximum loops - this guarantees the termination of **RAND**, or the minimum cardinality of S_{best} - obviously this may not assure the stop of **RAND**. Since **RAND** starts with F and ends with S_{best}, it works in a backward fashion in terms of finding the solution. The way in which features are generated is random.

Researchers (Siedlecki and Sklansky, 1988) also proposed to apply Genetic algorithms, and Simulated annealing to feature selection. In a genetic algorithm (Goldberg, 1989), a solution is usually represented by a finite sequence of numbers (such a string is called a *chromosome*, each number is a *gene*). The algorithm manipulates a set of chromosomes (the population), in a manner resembling the mechanism of natural evolution. The chromosomes are allowed to crossover or mutate to produce chromosomes of the next generation. A crossover of two chromosomes produces offspring chromosomes. For instance,

Table 3.11. Random search - **RAND**.

RAND Algorithm
Input: F - full set
U - measure
initialize: $S = S_{best} = \{\}$ /*S - subset set*/
$C_{best} = \#(F)$ /*# - cardinality of a set*/
repeat
$S =$ **RandGen**(F)
$C = \#(S)$
if $C \leq C_{best} \wedge S$ satisfies U
$S_{best} = S$
$C_{best} = C$
print S_{best}
until some stopping criterion is satisfied
Output: S_{best} /*Best set found so far*/

we have two chromosomes (11|000) and (00|111) where | specifies the crossover point, the crossover of these two chromosomes produces two new chromosomes (00|000) and (11|111). If each gene represents a feature, the two new chromosomes represent one empty and one full set. A mutation of a chromosome produces a near identical copy with some components of the chromosome altered. After mutation on the second gene, a chromosome (00|111) becomes (01|111). A fitness function is required to evaluate the fitness of each chromosome. The fittest survives. If the population has n chromosomes, after mutation and crossover, only the fittest n chromosomes will make to the next generation of n chromosomes. A fitness function is, in the context of feature selection, an evaluation measure. The number of generations a genetic algorithm should produce can be pre-determined. When the algorithm converges (the population stays unchanged), solutions are reached.

A simulated annealing algorithm transforms an optimization problem in a problem-specific manner into an annealing problem. Without loss of generality, the objective function (an evaluation measure) is assumed to be minimized. Following an annealing schedule, the temperature is set high in the beginning to allow sufficient active activities (hoping to avoid getting stuck in local minima), then gradually cools down to certain equilibrium representing the best subset. Neural networks plus node pruning (Setiono and Liu, 1997) can also be applied to feature selection. The idea is straightforward: train a multi-layer perceptron as usual following back-propagation, then prune the connections between layers as much as possible without sacrificing the predictive accuracy. The input units

without connections to the hidden units can be removed. The remaining input units are selected features.

Among the four nondeterministic algorithms, **RAND** and genetic algorithms produce multiple solutions, but simulated annealing and neural networks give single solution. In (Jain and Zongker, 1997) they observe such difference in reviewing these and some other feature selection algorithms. All of the above algorithms rely on some evaluation measures to determine the ranks of features or legitimacy of subsets. With different measures, more variations of feature selection methods are possible. In the next section, we introduce and discuss some representative evaluation measures.

3.4 EVALUATION MEASURES WITH EXAMPLES

We recapitulate that there are three types of evaluation measures, i.e., classic, consistency and accuracy. Classic measures are further divided into information, distance, and dependence measures. For two subsets of features, S_i and S_j, one is preferred to the other based on a measure U of feature-set evaluation. S_i and S_j are indifferent if $U(S_i) = U(S_j)$ and $\#(S_i) = \#(S_j)$ where $\#$ is the cardinality of a set; S_i is preferred to S_j if $U(S_i) = U(S_j)$ but $\#(S_i) < \#(S_j)$, or if $U(S_i) < U(S_j)$ and $\#(S_i) \leq \#(S_j)$. We continue here the discussion about measures in Section 2 of Chapter 2, give details about representative measures in each type to such an extent that a reader can implement them with minimum efforts.

In order to show the working of each chosen measure, we use the data introduced in Chapter 1 and reproduce it here in Table 3.12 for easy reference. For each measure, we give its definition and some details for implementation, and show by example how it works for the data in Table 3.12. Data are translated into numbers where for attribute Hair, blonde = 1, brown = 2, and red = 3; for attribute Height, short = 1, average = 2, and tall = 3; for attribute Weight, light = 1, average = 2, heavy = 3; for attribute Lotion, yes = 1, no = 0; and for class attribute Result, sunburned = 1, and none = 0.

3.4.1 Classic measures

The measures here have existed for quite some time and extensively used in pattern recognition (Ben-Bassat, 1982). Many variations have also been proposed and implemented. As we know, the way of evaluating features is influenced by feature generation scheme and search strategy. Here we offer the very first step of feature evaluation - using one classic measure to choose the best feature. The next step of feature evaluation is about evaluating feature subsets, which is related to feature generation and search strategy. One heuristic method can be: after the best feature is chosen, we find the second best feature by pair-

Table 3.12. An example of feature-based data - revisited.

	Hair	Height	Weight	Lotion	Result
i_1	1	2	1	0	1
i_2	1	3	2	1	0
i_3	2	1	2	1	0
i_4	1	1	2	0	1
i_5	3	2	3	0	1
i_6	2	3	3	0	0
i_7	2	2	3	0	0
i_8	1	1	1	1	0

ing each unchosen feature with the best feature, and select the best pair, then the best triplet, etc. The run time complexity for the heuristic procedure is $O(N^2)$. One should not be surprised if the ranked list of features thus obtained is not the same as the ranked list obtained in the first step, nor the optimal list obtained from complete search.

Before we present the classic measures, some notations are again given for the reader's convenience: $P(c_i)$ is the prior probabilities for all classes i, and $P(\mathbf{x}|c_i)$ is the conditional probabilities of $\mathbf{x}$ given class c_i. By Bayes' theorem, we have

$$P(c_i|\mathbf{x}) = \frac{P(c_i)P(\mathbf{x}|c_i)}{P(\mathbf{x})}, \tag{3.1}$$

$$P(\mathbf{x}) = \sum P(c_i)P(\mathbf{x}|c_i). \tag{3.2}$$

In other words, we only need two groups of probabilities ($P(c_i)$, $P(\mathbf{x}|c_i)$) in order to get $P(c_i|\mathbf{x})$. As an example, we list the priors and class conditional probabilities in Table 3.13.

Information Gain. Shannon's entropy is used here as an example of information gain measure. In Figure 3.7, data D is split by feature X into p partitions $D_1, D_2, ..., D_p$, and there are d classes. The information for D at the root amounts to

$$I(D) = -\sum_{i=1}^{d} P_D(c_i) log_2 P_D(c_i),$$

the information for D_j due to partitioning D at X is

$$I(D_j^X) = -\sum_{i=1}^{d} P_{D_j^X}(c_i) log_2 P_{D_j^X}(c_i),$$

Table 3.13. Priors and class conditional probabilities for the sunburn data.

	Result (Sunburn)	
	No	Yes
P(Result)	5/8	3/8
P(Hair=1|Result)	2/5	2/3
P(Hair=2|Result)	3/5	0
P(Hair=3|Result)	0	1/3
P(Height=1|Result)	2/5	1/3
P(Height=2|Result)	1/5	2/3
P(Height=3|Result)	2/5	0
P(Weight=1|Result)	1/5	1/3
P(Weight=2|Result)	2/5	1/3
P(Weight=3|Result)	2/5	1/3
P(Lotion=0|Result)	2/5	3/3
P(Lotion=1|Result)	3/5	0

and the information gain due to feature X is defined as

$$IG(X) = I(D) - \sum_{j=1}^{p} \frac{|D_j|}{|D|} I(D_j^X),$$

where $|D|$ is the number of instances in D, and $P_D(c_i)$ are priors for data D.

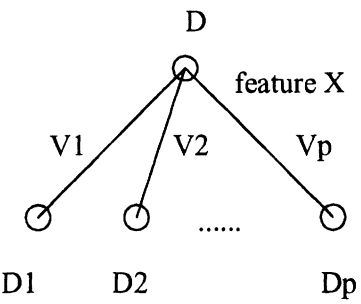


Figure 3.7. Information gain calculation example: Data D is partitioned by feature X into data subsets D_i, $i = 1, 2, ..., p$.

Table 3.14. Ordering features according to their information gains.

Information-Gain
Input: D - the training data set;
A_i - all features, $i = 1, 2, ..., N$
initialize: $L = \{\}$ /*L - empty list*/
for $i = 1$ **to** N
begin
calculate IG(A_i);
insert A_i in L in descending order w.r.t. IG(A_i);
end
Output: L /*The first A_i in L is the best*/

A feature ordering algorithm using information gain is shown in Table 3.14. Its time complexity to obtain the ranked list L is $O(N^2)$ where N is the number of features. However, it is only $O(N)$ if the best feature is sought.

Let's look at feature *Hair* which has three values (1, 2, 3). With this feature, we could partition the data into $D_1^{Hair} = \{i_1(1), i_2(0), i_4(1), i_8(0)\}$, $D_2^{Hair} = \{i_3(0), i_6(0), i_7(0)\}$, and $D_3^{Hair} = \{i_5(1)\}$, where $i_j(k)$ represents instance j with class value k. Since $I(D) = -\frac{5}{8}log_2\frac{5}{8} - \frac{3}{8}log_2\frac{3}{8} = 0.954434$, $I(D_1^{Hair}) = 2 \times (-0.5log_20.5) = 1$, $I(D_2^{Hair}) = 0$, and $I(D_3^{Hair}) = 0$, $IG(Hair) = I(D) - \frac{4}{8}I(D^{Hair}) = 0.454434$. This gain happens to be the largest among the four $IG(X)$: $IG(Lotion) = 0.347590$, $IG(Height) = 0.265712$, and $IG(Weight) = 0.015712$. Hence, feature *Hair* is the first in the ranked list.

Information gain has a tendency to choose features with more distinct values. Information gain ratio was suggested by (Quinlan, 1988) to balance the effect of many values. It is only applied to discrete features; for continuous features, a split point is found with the highest gain or gain ratio among the sorted values in order to split the values into two segments. Thus, information gain can be calculated as usual.

Distance Measures. We can evaluate features using distance measures. Simply replacing IG with *DD* - directed divergence or *V* - variance, we can obtain two more algorithms. Since they are very similar to the **Information-Gain** algorithm, we will only show their definitions and calculations. In both measures, we need $P(c_i|\mathbf{x})$ which can be calculated via Equation 3.1, $P(c_i)$ and $P(\mathbf{x}|c_i)$ can be found in Table 3.13.

1. Directed divergence DD is shown in Equation 3.3. The features are ranked as (Hair, Lotion, Height, and Weight) because DD(Hair)= 0.454434, DD(Height)= 0.265712, DD(Weight)= 0.015712, and DD(Lotion)= 0.347590.

$$DD(X_j) = \int [\sum P(c_i|X_j = x) log \frac{P(c_i|X_j = x)}{P(c_i)}] P(X_j = x) dx. \quad (3.3)$$

2. Variance V is shown in Equation 3.4. The features are ranked as (Lotion, Hair, Height, and Weight) because V(Hair)= 0.059082, V(Height)= 0.054525, V(Weight)= 0.005208, and V(Lotion)= 0.064600.

$$V(X_j) = \int [\sum P(c_i)(P(c_i|X_j = x) - P(c_i))^2] P(X_j = x) dx. \quad (3.4)$$

Dependence Measures. If we replace information gain with a dependence measure in the algorithm shown in Table 3.14, we obtain another algorithm. As an example, if we adopt Bhattacharyya dependence measure B in Equation 3.5, a ranked list is obtained as (Hair, Lotion, Height, Weight) because B(Hair)= 2.566696, B(Height)= 2.354554, B(Weight)= 2.101802, and B(Lotion)= 2.469646.

$P(x)$ can be obtained following Equation 3.2 and priors and class conditional probabilities are found in Table 3.13.

$$B(X_j) = \sum -log[P(c_i) \int \sqrt{P(X_j = x|c_i) P(X_j = x)} dx] \quad (3.5)$$

Summary. In the above, we give examples of ranking individual features for the classic measures. They can be extended to evaluating subsets of features, although the number of combinations of features is usually too large as we know. In order to reduce the run time complexity, one way of gradually adding features is to find the best feature, then find among the rest another feature that combines with the selected feature(s) to form a new subset of selected features. More will be elucidated using feature selection methods as examples in Chapter 4.

3.4.2 Consistency measures

We introduce *inconsistency rate* as one of consistency measures. Zero inconsistency means total consistency. Let U be an inconsistency rate over the dataset

given a feature subset S_i. The inconsistency rate is calculated as follows: (1) two instances are considered inconsistent if they are the same except for their class labels (we call these instances as matching instances), for example, for two instances (0 1 1) and (0 1 0), their values of the first two features are the same (0 1), but their class labels are different (1 and 0); (2) the inconsistency count is the number of all the matching instances minus the largest number of instances of different class labels: for example, in n matching instances, c_1 instances belong to label_1, c_2 to label_2, and c_3 to label_3 where $c_1 + c_2 + c_3 = n$. If c_3 is the largest among the three, the inconsistency count is $(n - c_3)$; and (3) the inconsistency rate is the sum of all the inconsistency counts divided by the total number of instances ($\mathcal{N}$). By employing a hashing mechanism, we can compute the inconsistency rate approximately with a time complexity of $O(\mathcal{N})$. Any standard data structure text books should have descriptions on various hashing mechanisms. The interested reader may consult (Kruse et al., 1997).

Here we give the proof outline to show that this inconsistency rate measure is monotonic, i.e., for two subsets S_i and S_j, if $S_i \subset S_j$, then $U(S_i) \geq U(S_j)$. Since $S_i \subset S_j$, the discriminating power of S_i can be no greater than that of S_j. As we know, the discriminating power is reversely proportional to the inconsistency rate. Hence, the inconsistency rate of S_i is greater than or equal to that of S_j, or $U(S_i) \geq U(S_j)$. The monotonicity of the measure can also be analyzed as follows. Consider three simple cases of $S_k (= S_j - S_i)$ without loss of generality: (i) features in S_k are irrelevant, (ii) features in S_k are redundant, and (iii) features in S_k are relevant. (We consider here data without noise.) If features in S_k are irrelevant, based on the definition of irrelevancy, these extra features do not change the inconsistency rate of S_j since S_j is $S_i \cup S_k$, so $U(S_j) = U(S_i)$. Likewise for case (ii) based on the definition of redundancy. If features in S_k are relevant, that means S_i does not have as many relevant features as S_j. Obviously, $U(S_i) \geq U(S_j)$ in the case of $S_i \subset S_j$. It is clear that the above results remain true for cases that S_k contains irrelevant, redundant as well as relevant features.

As for the sunburn example, the inconsistency rate is zero for the following cases: (1) two features (Hair and Lotion); (2) first three features (Hair, Height and Weight); and (3) all super-sets of cases (1) and (2), including the full set of four features. Since we prefer the subset with the minimum number of features, we choose case (1). Intuitively, it also makes more sense than case (2). Why should Height and Weight be linked to sunburned? This is a hindsight, of course, but it is also an effective way to verify if the results are sensible using our knowledge. Case (1) can be obtained by a sequential forward heuristic algorithm, and case (2) can be found by a sequential backward heuristic algorithm.

Since the condition for Branch & Bound to work optimally is that its evaluation measure be monotonic and this inconsistency rate measure is monotonic, we can expect to find an optimal feature subset by applying the inconsistency rate measure to Branch & Bound. Consistency measures are only suitable for discrete features. For continuous features, they have to be discretized before this kind of measure can be applied (Catlett, 1991, Kerber, 1992).

3.4.3 Accuracy measures

In theory, any classifier is qualified to provide predictive accuracy as an evaluation measure. However, in practice, we are usually constrained by many factors. Among many, we list three factors: (1) the choice of a classifier - many classification algorithms are extant for different needs (Mitch, 1997), we should always keep it in mind that feature selection is performed to improve a classifier; (2) the speed of learning - how fast can a classifier be induced plays an important role in practice, some applications can wait, but others cannot; and (3) the task at hand - generalization from the data is a goal, we do have tasks that may also emphasize things like explicitness, simplicity, or comprehensibility. We may choose one classifier which suits the task at hand best. However, due to the time constraint, we may exclude those time consuming learning algorithms such as neural networks, and genetic algorithms. Here we simplify the issue and assume no specific requirement on a classifier.

A common way of applying accuracy measure is Sequential Backward Search. The idea is to see if the classifier can maintain its accuracy by removing one feature at a time until there is only one feature or until accuracy deteriorates to an intolerable level. In Table 3.15, we present such an algorithm using the number of features as the stopping criterion. This algorithm's run time complexity is $O(N^2)$ since it removes one feature at a time until only one feature remains. However, we need to bear in mind that constructing a classifier, as in `induce()`, often requires a similar order of complexity. Its time complexity varies with different classification algorithms, so it is not given here. Function **reorganize** arranges the remaining features in order for easy indexing.

In the sunburn example, we can use a Naive Bayesian Classifier (NBC) in `induce()`. With feature set F, NBC's accuracy is 100%. By removing one of four features in turn, we obtain the following according to their accuracy rates ($ER = 1-$ accuracy):

$S_1 = F - \{\text{Hair}\}$	87.5%
$S_2 = F - \{\text{Height}\}$	100%
$S_3 = F - \{\text{Weight}\}$	100%
$S_4 = F - \{\text{Lotion}\}$	75%

Table 3.15. Ordering features according to accuracy.

```
Accuracy
  Input: D - the training data set;
         F - the full set of features;
         A_i - feature i, i = 1,2,...,N
  initialize: S = F                          /*S - full feature set*/
              L = {}                         /*L - empty list */
  ER = induce(D,S)                /*ER - error rate with set S*/
  repeat
    min = ER;
    for i = 1 to N
     begin
      S_i = S - A_i
      ER_i = induce(D,S_i)
      if min > ER_i
        min = ER_i
        index = i
     end
    N = N - 1
    remove A_index from S and append it to L
    reorganize S from 1 to N
  until N = 1
  append A_1 to L          /*Features in L are in original indexes*/
  Output: reversed L              /*The first A_i in L is the best*/
```

So, we can remove features *Height* and *Weight* sequentially (S_5 = {Hair, Lotion}) and repeat the removal of one feature procedure, we have

$S_6 = S_5 - \{\text{Hair}\}$ 75%

$S_7 = S_5 - \{\text{Lotion}\}$ 75%

Based on the results above, we have a ranked list {Lotion, Hair, Weight, Height }.

3.5 CONCLUSION

In this chapter, we study feature selection in three individual aspects: search direction (how features are generated), search strategy, and evaluation measure. The isolation of one aspect from the others allows us to focus on the specific problems each aspect experiences and on the possibilities each aspect can have. When we discuss each individual aspect, we cannot help sense the interweaving

effects of the others. A particular method of feature selection is basically a combination of some possibilities of every aspect. In the beginning of the book, we started our analysis of perspectives of feature selection in a top-down approach; after studying each aspect of feature selection here, we are ready to go up in a bottom-up manner. Combining aspects with different possibilities, we will be able to create or reproduce many effective feature selection methods.

References

Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305.

Ben-Bassat, M. (1982). Pattern recognition and reduction of dimensionality. In Krishnaiah, P. R. and Kanal, L. N., editors, *Handbook of statistics-II*, pages 773–791. North Holland.

Boddy, M. and Dean, T. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *European Working Session on Learning*.

Cover, T. (1974). The best two independent measurements are not the two best. *IEEE Trans. Systems, Man and Cybernictics*, 4:116–117.

Dash, M. and Liu, H. (1997). Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3).

Dietterich, T. (1997). Machine learning research: Four current directions. *AI Magazine*, pages 97–136.

Domingos, P. (1997). Why does bagging work? a Bayesian account and its implications. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 155 – 158. AAAI Press.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc.

Jain, A. and Zongker, D. (1997). Feature selection: Evaluation, application, and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):153–158.

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher.

Kerber, R. (1992). ChiMerge: Discretization of numeric attributes. In *AAAI-92, Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 123–128. AAAI Press/The MIT Press.

Kruse, R., Tondo, C., and Leung, B. (1997). Data structures & program design in C. International Edition.

Liu, H. and Wen, W. (1993). Concept learning through feature selection. In *Proceedings of the First Australian and New Zealand Conference on Intelligent Information Systems*, pages 293–297.

Mitch, T. (1997). *Machine Learning.* McGraw-Hill.

Narendra, P. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Trans. on Computer*, C-26(9):917–922.

Press, W., Teukolsky, S., Vetterling, W., and Flannery, B. (1992). *Numerical Recipes in C.* Cambridge University Press, Cambridge.

Quinlan, J. (1988). Decision trees and multi-values attributes. In J.E., H., Michie, D., and J., R., editors, *Machine Intelligence*, volume 11. Oxford University Press.

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

Setiono, R. and Liu, H. (1997). Neural network feature selectors. *IEEE Trans. on Neural Networks*, 8(3):654–662.

Siedlecki, W. and Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220.

Notes

1. A node is a state in the search space and actually a subset of features.

4 FEATURE SELECTION METHODS

With the unified model, we were able to study the three major aspects in the previous chapter. Now we look at possible combinations of these aspects, which can be used to construct a feature selection method. The objective of this chapter is three-fold: (a) to categorize the existing methods in a framework defined by the three major aspects; (b) to discover what have not been done and what can be done; and (c) to pave the way towards a meta system that links applications to specific methods.

The study in Chapter 3 makes it possible for us to come up with a framework that can be used to categorize all types of feature selection methods. For each aspect (a dimension in the framework), we generally identify three possibilities. Theoretically speaking, the framework allows twenty seven combinations (categories) of feature selection methods. How do various existing feature selection methods stand in this framework? What is the necessity for each combination or how different these combinations are? What is even more interesting is how each combination is related to a certain type of application. More refined variations of these twenty seven combination are expected. We believe that the understanding of the framework can equip us with sufficient knowledge to embark on challenging tasks in practice.

Based on our earlier study, we can consider each possibility of any aspect as a building block, or component, and by putting these components together sensibly, one can create his very own method that is specially designed for a task at hand. The procedure to follow is straightforward:

1. Select a feature generation scheme;
2. Select a search strategy;
3. Pick an evaluation measure;
4. Determine a stopping criterion; and
5. Combine the above four components.

Before we start trying this procedure, it is desirable to know exactly what are the possible combinations we can have. A 3-dimensional structure shown in Chapter 3 (Figure 3.1) is one way of categorizing the combinations. Based on this structure, extending from Table 2.1 of Chapter 2, we have a new table in which we include types of measures (classical, consistency, and accuracy) for each entry. Each combination is a 3-tuple, <SD,SS,ME >, here SD is one of the three search directions, SS one of the three search strategies, and ME one of the three measures. In total, there could be 27 combinations ($3 \times 3 \times 3$). However, some combinations of search direction and strategy are impractical: for example, when the search strategy is nondeterministic, it is only reasonable that features are generated randomly; when the search direction is random, it does not make sense to perform exhaustive search. Therefore, we have a total of 18 combinations.

Using Table 4.1 as a framework to study the existing feature selection methods, we can understand which combinations have been tried and implemented and which combinations have not. This understanding can help us in analyzing the existing methods and studying their disadvantages and advantages to tailor the methods for particular tasks, and focus on new feature selection methods. Remember that this categorization is rather approximate and should only be used as a guideline. We expect to improve and/or revise this 3-d structure as research develops further.

4.1 REPRESENTATIVE FEATURE SELECTION ALGORITHMS

This section presents some representative feature selection algorithms as examples to show how the feature selection components are put together in forming a workable algorithm. The algorithms are described and their components are identified. Some of them will be used in evaluation of some data sets with known characteristics in the next chapter. These algorithms are presented in terms of

Table 4.1. A 3-d structure for feature selection algorithms categorization.

Search Direction	Search Strategy					
	Complete		Heuristic		Nondeterministic	
Forward	Classic:	C1	Classic:	C2	Classic:	✗
	Consistency:	C3	Consistency:	C4	Consistency:	✗
	Accuracy:	C5	Accuracy:	C6	Accuracy:	✗
Backward	Classic:	C7	Classic:	C8	Classic:	✗
	Consistency:	C9	Consistency:	C10	Consistency:	✗
	Accuracy:	C11	Accuracy:	C12	Accuracy:	✗
Random	Classic:	✗	Classic:	C13	Classic:	C14
	Consistency:	✗	Consistency:	C15	Consistency:	C16
	Accuracy:	✗	Accuracy:	C17	Accuracy:	C18

their search strategies: exhaustive, heuristic and stochastic. We also discuss some variations, such as weighting methods, hybrid methods and incremental methods.

4.1.1 Complete methods

Six combinations (C1, C3, C5, C7, C9, and C11) can be found in this category and all employ exhaustive or complete search. Complete search and exhaustive search cover the whole search space. We give two examples of this kind: one is forward search - the size of a subset grows gradually until a subset satisfies a chosen evaluation criterion; the other is backward search - the size of a subset decreases gradually until a chosen evaluation criterion cannot be satisfied anymore. In these two methods, a consistency (or inconsistency) measure is used, i.e., we discuss, in particular, C3 and C9. One may use other measures to determine the quality of a feature subset, which would result in other combinations.

The first method is Focus (Almuallim and Dietterich, 1991). It considers all the combinations among N features starting from an empty subset: $\binom{N}{1}$ subsets first, $\binom{N}{2}$ subsets next, ... When Focus finds a subset that satisfies the consistency measure, it stops. The outline of the method is shown in Table 4.2. Regardless of the run-time complexity of consistency checking (as we know in a previous section, it is about $O(N)$), Focus needs to generate $\sum_i^m \binom{N}{i}$ subsets in order to find a minimum subset of m features that satisfies the consistency criterion. When m is not small (i.e., $m > N/2$), the run-time cost is quite prohibitive. Some heuristic variations of Focus were discussed in (Almuallim

and Dietterich, 1994). This method expands the search space until it finds a good subset.

Table 4.2. Focus - an exhaustive forward search algorithm.

Focus
Input: F - all features x in data D
U - inconsistency rate as evaluation measure

initialize: $S = \{\}$

for $i = 1$ to N
for each subset S of size i
if Cal$U(S, D) = 0$ /* *CalU(S, D) returns inconsistency*/*
return S
Output: S - a minimum subset that satisfies U

ABB (Liu et al., 1998) is an automated version of Branch and Bound algorithm (Narendra and Fukunaga, 1977). By "automated" it means that the bound is determined automatically, unlike a conventional algorithm of Branch and Bound in which the bound is predetermined. In Table 4.3, ABB starts with the full set of features, removing one feature at a time in a breadth-first search fashion until no more feature can be removed while the inconsistency criterion is still satisfied. ABB expands the search space as well as prunes it. A branch is pruned when it cannot grow due to its violation of the inconsistency criterion. When a branch stops growing, its root could be a solution. At the end, a set(s) with the minimum number of features is selected if it satisfies evaluation measure U. In essence, ABB searches a lattice. However, by introducing an enumeration scheme and a legitimacy test, it conducts a tree search. The enumeration scheme is: the full set is represented by an array of 1's (0 means a feature in that position is not selected); start from the right end or find the first '0' from the left end as the marker; create child nodes by changing one '1' at a time to '0' from right to left until each '1' to the left side of the marker is changed in turn. The legitimacy test is to avoid testing the child node of a pruned node. Recall that a node is a subset of features. A node is *legitimate* if the Hamming distance between this node and a pruned node is not 1.

The reader may notice that the implementation of ABB is of tree breadth-first search, while in the implementation of BAB and ABAB, the search is of lattice depth-first. The equivalence of the two implementations (lattice depth-first and tree breadth-first) can be proven if the evaluation measure is monotonic. Or more generally, the order of next subset generation does not affect

Table 4.3. ABB - an automated branch and bound (backward search) algorithm.

ABB Algorithm

Input: S - all features x in data D,
U - inconsistency rate as evaluation measure,
Q - an empty queue,
S_1, S_2 - subsets

initialize: $L = \{S\}$
$\delta = Cal\mathrm{U}(S, D)$
ABB(S, D)
for each feature x in S {
$S_1 = S - x$ /*Remove one feature at a time.*/
enque(Q, S_1) }
while notEmpty(Q) {
$S_2 = \mathrm{deque}(Q)$;
if (S_2 is legitimate $\wedge$ Cal$U(S_2, D) \leq \delta$)
$L =$ **append**(S_2, L)
ABB(S_2, D)
}
S_{min} = the minimum subset(s) in L satisfying U.
Output: S_{min}

the result's optimality and the algorithm's efficiency if the evaluation measure is monotonic. Since these two implementations generate different sequences of subsets, if the evaluation measure is not monotonic, the equivalence cannot be ensured. This is because we cannot guarantee that the search is complete.

In the algorithm of ABB, Cal$U(S, D)$ calculates U - the inconsistency rate of D with all features S. The working of ABB is best explained in detail through an example. Let's consider a simple example in which a data set is described by four features, assuming only the first two are relevant. The root S_0 = (1 1 1 1) of the search tree is a binary array with four '1's. Refer to Figure 4.1: following ABB, we expand the root to four child nodes by turning one of the four '1's into '0' (L2 in Figure 4.1). All these four are legitimate child nodes because the root is a valid node. The child nodes expanded from the current parent may be illegitimate if they are also children of some pruned nodes. The four nodes are S_1 = (1 1 1 0), S_2 = (1 1 0 1), S_3 = (1 0 1 1), and S_4 = (0 1 1 1). Since one of the relevant features is missing, $U(S_3)$ and $U(S_4)$ will be greater than $U(S_0)$ where U is the inconsistency rate on the given data. Hence, the branches rooted by S_3 and S_4 are pruned and will not grow further. Since

ABB is a breadth-first search algorithm, it expands S_1 first. Following our enumeration procedure, we now have three more new nodes (L3 in Figure 4.1). That is S_5 = (1 1 0 0), S_6 = (1 0 1 0), and S_7 = (0 1 1 0). However, S_6 and S_7 are illegitimate since they are also children of pruned nodes S_3 and S_4 respectively. This can be determined by the Hamming Distance (HD) of two nodes, e.g., S_6 and S_3. Since their HD is 1, S_6 is also a child of S_3. Only when a new node passes the legitimacy test will its inconsistency rate be calculated. Doing so improves the efficiency of ABB because P (number of patterns) is normally much larger than N (number of attributes). The rest of the nodes ($S_8, ..., S_{11}$) are generated and tested in the same spirit.

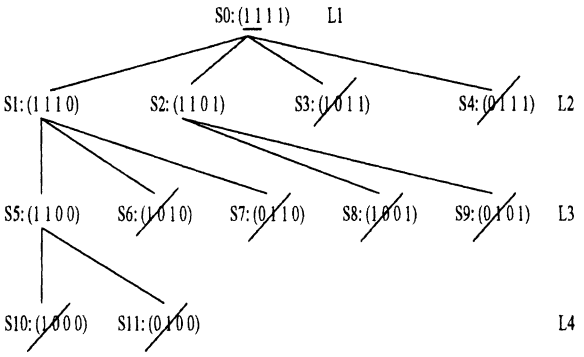


Figure 4.1. A simple example with four features. The first two are relevant.

In retrospect, there is a subtle difference between Focus and ABB. ABB requires a feature evaluation measure to be monotonic while Focus does not. Hence, in Focus, nothing prevents us from using a classifier's accuracy as a measure if we change the stopping criterion to "finding the best accuracy among all the subsets".

In the literature, we can also find some examples of other combinatins. They are BFF (C1) (Xu et al., 1988), Beam Search (C5) (Doak, 1992), and Branch & Bound (C7) (Narendra and Fukunaga, 1977). An overview of these methods is presented in (Dash and Liu, 1997).

4.1.2 Heuristic methods

There are many heurisitic feature selection methods. In general, they sacrifice the promise of optimal subsets for a quick solution, although the goal is still to

find an optimal solution. The simplest method is to induce a classifier and select the features used by the classifier (the reader may notice that this is a variation of C12 - sequential backward heuristic selection using accuracy as the measure). Let's call this method **Wrap1** shown in Table 4.4. For the sunburn data, as seen in Section 3.4.3 of Chapter 3, we ran NBC, and features *Hair* and *Lotion* were chosen. We now run C4.5 (Quinlan, 1993) - a decision tree induction program to induce a decision tree. In the resulting decision tree are two features, *Hair* and *Lotion*. So we just select these two features. It has been shown, however, that it is generally not a good idea to use decision tree induction methods (e.g., ID3 or C4.5) for feature selection (Almuallim and Dietterich, 1994). Later on we will be able to verify this finding on datasets such as Corral and Parity by observing that an ID3 like learning algorithm will select irrelevant features, leading to deterioate accuracy. Naive Bayesian Classifier cannot be used in **Wrap1** since it does not select any features, but uses all. Researchers have tried pruned neural networks for feature selection and achieved good results (Setiono and Liu, 1997). The features can be ranked according to their number of connections to the next layer and their weights. Or we can simply be satisfied with a subset of features that still have connections after pruning. However, this type of learning algorithms is usually time-comsuming.

Table 4.4. A heuristic feature selection algorithm - inducing a classifier.

Wrap1
Input: **x** - features
LA - learning algorithm

initialize: set $S = \{\}$ /* *L stores selected features**/
$C =$ `induceClassifier(x, LA)`
$S =$ `findAllFeatures(C)` /* *Find all features in C**/
Output: S.

A more sophisticated version of using a classifier to determine the quality of a subset is to go for sequential forward or backward selection. An example for sequential backward selection (C12) has been shown in Table 3.15 using a classifier's accuracy as the measure. A sequential forward selection algorithm (C6) works as follows: begin with an empty set S, in each loop (the maximum number of loops is N), choose one feature from the unchosen set of features that gives the best accuracy combining with the already chosen features in S.

An even more sophisticated method was proposed in (Pudil et al., 1994), called "floating" algorithms. It is basically a variation of bidirectional search

in which it allows a variable (thus floating) number of features to be included or excluded. If the search is dominantly forward, it is of sequential forward floating selection; otherwise, sequential backward floating selection. (Kohavi and John, 1998) proposed a new way to change the search space toplogy by creating dynamic operators that directly connect a node to nodes considered. These compound operators make a backward search, starting from the full set of features, practical.

Again, the reader may use other evaluation measures (e.g., classic measures or consistency measure). For example, SetCover proposed by (Dash, 1997) is one that uses the inconsistency measure (C4) to check if selected features can maximumly distinguish the instances from different classes; if they do, the distinguishable instances are removed, and extra features are introduced one by one to separate the instances of different classes. Combinations C2 and C8 suggested in Table 4.1 can be found in (Mucciardi and Gose, 1971) and (Koller and Sahami, 1996), respectively.

How about combinations like C13, C15, and C17? In these combinations, features are randomly generated, but search is heuristic. Regardless of measures used, one implementation of these combinations is adopting a heuristic search algorithm and in each sub-search space, features are randomly generated and these subsets of features form the possible sub-search spaces.

Which measure performs best really depend on many factors (predictive accuracy, simplicity of a classifier, comprehensibility of the results, run-time, etc.) and we will discuss this issue in the next chapter. However, one thing is certain that different measures will give different near optimal subsets, and this should be acceptable because we know the trade-off between optimality and efficiency. However, the near optimal solution should not be too far away from the optimal one to be useful. The optimality is defined by the measure used and the optimal subset under one measure may not be optimal under another. We will witness more evidence in Chapter 5.

4.1.3 Stochastic methods

Now let's turn our attention to stochastic methods. These methods are the result of our continued pursuit of optimal subsets without exhaustive search. Unlike the methods in the other two categories above, features are not *sequentially* added or removed from a subset (depending on forward or backward search). This category of methods basically allows search to follow feature subsets that are randomly generated.

Genetic algorithms and Simulated annealing are two often mentioned methods (Siedlecki and Sklansky, 1988). A detailed account of genetic algorithms for feature selection can be found in (Siedlecki and Sklansky, 1989). In (Jain and

Zongker, 1997), the algorithm was reimplemented and tested. They concluded that it is difficult to compare the GA method with the sequential methods. One reason is that there are several parameters for which no guidance is available on how to specify their values. Combinations such as C14, C16, and C18 in Table 4.1 can be found in various implementations of genetic algorithms or simulated annealing for feature selection (Jain and Zongker, 1997, Siedlecki and Sklansky, 1989) depending on the measures used. In both algorithms, features are randomly generated when a fitness function or a utility function is optimized. These functions are closely related to the evaluation measures for feature subsets (Liu and Motoda, 1998).

We discuss two stochastic algorithms here: LVF (C16) and LVW (C18). LVF is a Las Vegas algorithm for Filter feature selection. LVF (Liu and Setiono, 1996b) consists of a random procedure that generates random subsets of features and an evaluation procedure that checks if each subset satisfies the chosen measure, seen in Table 4.5. The emergent property of the two procedures in LVF is the optimal subset of features. The evaluation measure used in LVF is inconsistency rate. One of the two parameters is an *allowed inconsistency rate* that can be estimated by the inconsistency rate of the data with all features. The other is the maximum number of subsets randomly generated, which is, as a matter of fact, a stopping criterion that can be changed to other stopping criteria. One alternative can, for example, let the loop run forever and report a solution whenever it finds a better subset. This is a kind of anytime algorithms.

In Table 4.5, *maxTries* is a number linked to the number of original features, i.e., $c \times N$. The rule of thumb is that the more features in the data, the more difficult it is for feature selection. c is a constant. We can start c with some number (e.g., 77 as we used in our experiments), or simply make it as N^2. In general, the larger *maxTries* is, the better solution we can expect. Another more reasonable way of setting *maxTries* is to link to the search space the user wants to cover. As we know, the complete search space is 2^N, if the user wants to cover $p\%$ of the space, he can set $maxTries = 2^N \times p\%$. Function **randomSet**() requires a good random number generator to produce different subsets of features. The better the random number generator is, the less likely that the same subset is repeatedly generated.

LVF is a quite simple algorithm. That also means it can easily be modified. We can change one of the two procedures. If we changed the random procedure, we would not have a random search algorithm. So, we can only change the evaluation procedure if we want this algorithm to remain in this category. Assuming that we decide to use a classifier's estimated accuracy as the measure, we have LVW (a Las Vegas algorithm for Wrapper feature selection), shown in Table 4.6 (Liu and Setiono, 1996a). Statistical methods are available for estimating a classifier's accuracy. Some methods are to be introduced in Chap-

Table 4.5. A stochastic filter feature selection algorithm - LVF.

LVF

Input: *maxTries* - the maximum number of loops
U - the inconsistency measure
D - a dataset with N features
γ - an allowed inconsistency rate

initialize: list $L = \{\}$ /* *L stores equally good sets**/
$C_{best} = N$
for *maxTries* loops
begin
$S =$ **randomSet**(seed)
$C = \#(S)$ /* # - the cardinality of S */
if $(C < C_{best} \wedge \text{Cal}U(S, D) < \gamma)$
$S_{best} = S$
$C_{best} = C$
$L = \{S\}$ /* L is reinitialized*/
else if $(C = C_{best} \wedge \text{Cal}U(S, D) < \gamma)$
$L =$ **append**(S, L)
end

Output: L /*all equivalently good subsets found by LVF*/

Table 4.6. A stochastic wrapper feature selection algorithm - LVW.

LVW

Input: *maxTries* - the maximum number of loops
LA - a learning algorithm
D - a dataset with N features
F - a full set of features

initialize: list $L = \{\}$ /* *L stores sets with equal accuracy*/

$A_{best} =$ **estimate**(D, F, LA)
for *maxTries* loops
begin
$S =$ **randomSet**(seed)
$A =$ **estimate**(D, S, LA)
if $(A > A_{best})$
$S_{best} = S$
$A_{best} = A$
$L = \{S\}$ /* L is reinitialized*/
else if $(A = A_{best})$
$L =$ **append**(S, L)
end

Output: L /*all equivalently good subsets found by LVW*/

ter 5. We use 10-fold cross validation for LVW. The point of mentioning LVW here is that a new feature selection algorithm can be conveniently designed by changing some component of another algorithm. We will see later that even though we only change the evaluation measure, LVF and LVW could be very different in terms of run time as well as subsets selected. (The reader may try to guess how and why they are different based on the knowledge of these two measures employed.)

maxTries is determined in the same way as in LVF, and it is the only parameter to be set for LVW. However, in order to run LA - the learning algorithm, one may need to set some parameters required by the learning algorithm, but one could just opt for the default parameters. Function **estimate**() is in for LVW instead of Cal$U$() (See Table 4.6). There are many ways to implement it. More details are delayed till Chapter 5. LVF and LVW use consistency and accuracy as evaluation measures respectively. Classic measures can also be used in LVF.

So far we have covered the standard methods of feature selection by considering various components of feature selection. We are now ready to move away a bit from the norm and look at some variations of feature selection methods.

4.1.4 Feature weighting methods

Relief was proposed by (Kira and Rendell, 1992b) in the wake of understanding that real world problems involve much feature interaction (i.e., several features depend on each other in determining an instance's class), many induction algorithms (such as ID3 or C4.5) have problems with choosing relevant features, and complete search methods (such as Focus) are inefficient as well. **Relief** selects features that are statistically relevant. Although its objective is still selecting features, it does not explicitly generate feature subsets and test them like many methods discussed above. Instead of generating feature subsets, Relief focuses on sampling of instances without explicit search for feature subsets. The underlying idea is that relevant features are those whose values can distinguish among instances that are near each other. Therefore, two nearest neighbors (one instance from each class) are sought for each given instance (I), one is near-hit (H), the other is near-miss (J)[1]. Ideally, a feature is relevant if its values are the same between I and near-hit, and different between I and near-miss. This checking can be implemented in terms of some distance between a feature's values: the distance should be minimum for I and H, and maximum for I and J. The distance of each feature for each randomly picked instance is accumulated in a weight vector $\mathbf{w}$ of the same dimension as the number of features. The relevant features are those having their weights exceeding a relevance threshold τ. This threshold can be determined by a statistical method of interval estimation (Kira and Rendell, 1992a). The sample size m can be varied and a larger m implies more reliable approximation. The Relief algorithm is presented in Table 4.7. As we see, Relief does not fit in a standard model of feature selection in terms of search and feature generation. However, it does evaluate a feature using a distance measure.

In Table 4.7, `diff()` calculates the difference between the values of some feature for two instances. For discrete features, the difference is either 1 (the values are different) or 0 (the values are the same), while for continuous features, the difference is the actual difference normalized to the interval $[-1, 1]$.

Relief can handle both discrete and continuous data and has been shown effective for both independent features and features with strong dependencies among them. Some problems with Relief are (1) it is limited to two-class data sets; (2) it is insensitive to redundant features. (Kononenko, 1994) extends the original Relief to deal with noisy, incomplete, and multi-class data sets. Both (Kira and Rendell, 1992a) and (Kononenko, 1994) have shown that feature

Table 4.7. A multivariate feature ranking algorithm.

Relief
Input: **x** - features
m - number of instances sampled
τ - adjustable relevance threshold

initialize: $\mathbf{w} = 0$
for $i = 1$ to m
begin
randomly select an instance I
find nearest-hit H and nearest-miss J
for $j = 1$ to N
$\mathbf{w}(j) = \mathbf{w}(j) - \mathtt{diff}(j, I, H)^2/m + \mathtt{diff}(j, I, J)^2/m$
end

Output: **w** greater than τ

weighting helps in decision tree induction. As reported by (Aha, 1998), feature weighting also plays an important role in lazy learning algorithms.

4.1.5 Hybrid methods

Different search strategies require varied amounts of recourses and lead to differed results. Can we exploit their advantages and avoid their disadvantages? Focus is a forward selection algorithm, ABB is a backward selection algorithm. So, when relevant features are few (M is small), Focus should naturally be chosen, but if relevant features are many, ABB is a better choice. When we have no knowledge about the number of relevant features, we can resort to a combination of both - a hybrid method. Another hybrid method is to combine LVF with ABB. In the experiments conducted in (Liu and Dash, 1997), it is noticed that LVF usually reduces the number of features quickly in the first few hundred loops and then slowly finds improved solutions. One graph (Mushroom) from the experiments is reproduced here in Figure 4.2. As for ABB, we realize that if the number of irrelevant and/or redundant features is small, it can find the relevant features fast. Combining the two (LVF and ABB) seems a natural choice. This is because LVF can efficiently remove some irrelevant or redundant features in the initial stage of search, and when ABB takes over the search, the number of irrelevant or redundant features is smaller than the one in the beginning of the search. It is expected that ABB can search the

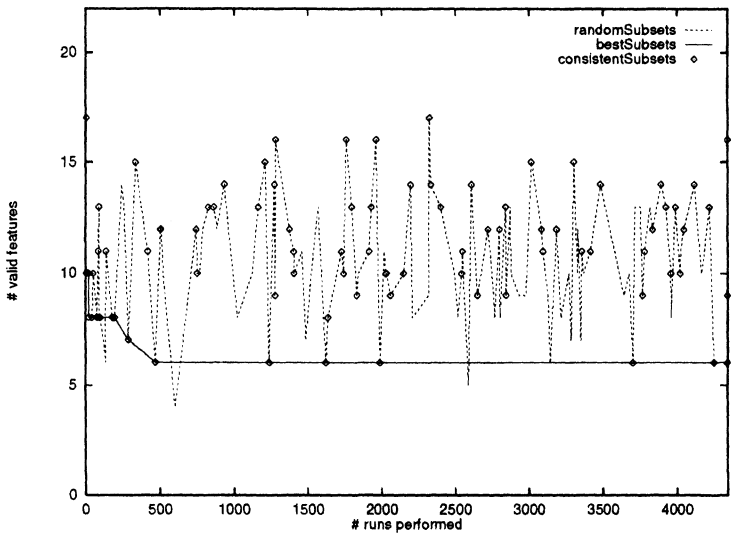


Figure 4.2. A typical trend of the decreasing number of valid features versus the number of runs performed. The Mushroom data is used in the experiment.

rest efficiently. However, it should be noted that this hybrid approach cannot guarantee the optimality of the final selected subset(s) because the output of LVF may not be the supersets of optimal subsets. That is, if we use the subsets found by LVF for the input of ABB and these subsets do not contain optimal subsets, it is obvious that the hybrid method cannot find optimal subsets. In the experiment of LVF, it is also noticed that equivalently good subsets abound in the early stage. The chance of excluding the supersets of optimal subsets can be reduced by applying ABB to all the equivalently good subsets. This results in a hybrid algorithm (QBB) shown in Table 4.8.

QBB (quick Branch and Bound) is composed of LVF and ABB. It runs LVF for *num* loops, hands over all chosen subsets to ABB to continue the search for better subsets. The value of *num* influences the results of LVF. Although the longer LVF runs, the better the subsets[2], as we know, until we find optimal subsets, any intermediate subsets may not even contain optimal subsets. The number of selected subsets also becomes smaller as LVF runs longer. On one hand, we want to reduce the features in the subsets that are fed to ABB; on the other hand, we want the number of these subsets not to be too small so that we can reduce the chance of losing optimal subsets. Thus, we need to

Table 4.8. A hybrid algorithm of feature selection.

QBB
Input: num - the number of loops in LVF
γ - an allowed inconsistency rate
D - a dataset with N features

initialize: list $L_{LVF} = L_{ABB} = \{\}$
$L_{LVF} =$ **LVF**(D, γ, num) /* run LVF with num loops*/
for each $S' \in L_{LVF}$
$S =$ **ABB**(S', D)
$L_{ABB} =$ **append**(S, L_{ABB})
Output: S_{min} /*The minimum subset in L_{ABB}*/

find a good transition point that balances the two concerns such that QBB can quickly find the optimal subset with high probability. Note that in QBB the number of subsets by LVF equals the number of times ABB is run. A large number of subsets means the slow down of QBB. (Liu and Dash, 1997) reports that if the amount of time is a constant, the experimental results suggest that the time should be equally divided between LVF and ABB.

We may construct a hybrid algorithm by combining different feature selection methods. For instance, we could have another hybrid system of LVF and a neural network feature selector when the number of features is very large. (Wang and Sundaresh, 1998) proposed a simple combination of breadth first and depth first search - another hybrid search method to quickly prune the search space for the monotonic evaluation function. The possible combinations are abundant. We will develop a contingency theory about effectively using feature selection algorithms and their combinations later in the next chapter. Building a hybrid method of feature selection can benefit from that finding.

4.1.6 Incremental approach

With the size of data getting bigger and bigger, all feature selection methods also face a problem of oversized data due to computer's limited memory. Therefore there is a need for overcoming this oversizing problem. A question is "Do we really need so much data for a feature selection method to choose relevant features?" Or can we settle for less data? The answer is negative. Let's use the inconsistency rate as an example. Since the whole data is needed to determine the inconsistency rate, any reduction of data will lead to an optimistic but misleading inconsistency rate. On one hand, we have the problem of oversized

data (which also slows down the feature selection process at best), so we are forced to reduce the data to a suitable size; on the other hand, we need all the available data to measure the quality of selected features. Some analysis of a feature selection method can help solve this problem.

We know that some portion of a huge sized data set can represent the data set reasonably well. The point is which portion and how large it should be. Instead of looking for the right portion (it is too difficult on its own), we can randomly select a portion p, use that portion to find a subset of features that satisfies the evaluation criterion, and test this subset on the unchosen part r of data (the whole data set minus the portion). There are two possibilities: (1) no inconsistency is found, it means that the task is accomplished; and (2) some inconsistencies are found, it means that the portion of data is not the right portion we wished. We can remember these inconsistent instances by adding them into the chosen portion and restart the feature selection on the *slightly* enlarged portion. The iteration process continues until no inconsistency is found[3]. Here we assume no inconsistency is allowed for the whole data set. If we allow some inconsistencies, then the same idea applies as long as for the two parts of data (p and r), the total number of inconsistencies should be within the allowable threshold.

Now, let's consider another issue, that is, what is a proper value for p. Intuitively, its size should not be too small or too large. If it is too small (only a few hundreds of instances, for example), after first iteration, many inconsistencies are found, these instances are added into portion p, and portion p is virtually almost as large as the whole data set. If it is too big (close to the size of the whole data), we still have the oversizing problem plus some overheads in making sure that the inconsistency criterion is met. In general, when p is large enough, the saving on run-time could be immaterial; in some extreme cases, the saving may be negative. A simple way to get out of this dilemma is to choose a percentage of data, say 10%. Or the size of p is roughly proportional to the number of features (N). The right percentage can be determined in experiment. The bottom line is that the portion p in the intensive computation part should be *just* small enough that the feature selection algorithm can work on the data. The above idea is implemented in an algorithm called LVI (Liu and Setiono, 1998) as a showcase, which is reproduced in Table 4.9. Function `CalIncon` returns an inconsistency rate as its value and the inconsistent data in *inconData*. It is clear that the underlying idea may also be extended to other feature selection methods.

Table 4.9. LVI - an incremental multivariate feature selection algorithm.

LVI
Input: *maxTries* - the maximum number of loops
U - the inconsistency measure
$D_0 = p\%$ of D chosen randomly
initialize: $D_1 = D - D_0$
loop
S_{LVF}= **LVF**(D_0, γ, S)
if (**CalIncon**$(S_{LVF}, D_1 + D_0, inconData) < \gamma$)
return(S_{LVF})
else
$D_0 =$ **append**$(inconData, D_0)$
$D_1 =$ **remove**$(inconData, D_1)$
end of loop

Output: S_{LVF}

4.2 EMPLOYING FEATURE SELECTION METHODS

It is often that the above methods are used independently. That is, one method is employed to select features and data is rearranged according to the selected features. The issue is then which method should be chosen among many available ones. This is a topic to be discussed in the next chapter. Often we need some prior knowledge about data and application to make a decision, assuming we know well what each feature selection method can offer. Depending on the amount of prior knowledge about data we have, we can employ these methods in combination in two modes:

- *Sequential*: In this mode, several methods are used in sequence. Hybrid methods are such an example. Basically, the sequential use of different methods tries to deploy a method in a certain stage of feature selection by taking advantage of each method's capability to improve feature selection performance. We have shown one example (QBB) above of sequential employment of two feature selection methods with different search strategies.

- *Parallel*: In this mode, different methods are run in parallel. It requires resources to do so. Since there is little knowledge about data and domain, it is not suitable to pick any single method, run it and get the results. When resources permit, methods that suit different data should run in parallel. Whichever method that finishes early will provide the first subset of features.

Whether the subset is optimal or not depending on the method. For example, Focus, LVF and ABB can be run in parallel, either Focus or ABB finishes first, it is certain that an optimal subset is reached. If LVF finishes first, the subset may or may not be optimal.

In reality, one can still use one computer to engage the feature selection methods in a parallel mode. The only risk is that an inappropriate choice may keep one waiting forever.

4.3 CONCLUSION

A 3-dimensional structure has resulted from the study of previous chapters. This structure is the last stage of a spiral process of "general-specific-general". In Chapters 2 and 3, feature selection is decomposed into individual aspects. Considering each possibility of an aspect as a component in assembling a feature selection method, we combine these components in this chapter to show by example that how a feature selection method can be built. Based on sufficient details supplied here, a reader can choose the best components with respect to a task at hand and assemble his very own feature selection method. The reader can also follow the leads provided in this chapter to learn lessons from existing similar methods implemented by other researchers and create his superior method. The 3-dimensional framework also points out some future work in feature selection research. As a guideline, it relates what has been done to what needs to be done. This framework maps out possible combinations of different feature selection methods and categorizes the existing methods. What is more important is that the framework provides a base for further improvement.

This framework is also an important step towards solving the matching problem between tools and applications as we are going to study in detail in the next chapter. With this framework, it is possible to characterize the feature selection methods and get us ready to match problems with tools and develop a contingency theory of how to apply different features methods with respect to various types of data sets.

References

Aha, D. W. (1998). *Feature Weighting for Lazy Learning Algorithms*, pages 13–32. In (Liu and Motoda, 1998).

Aha, D. W. and Bankert, R. L. (1995). A comparative evaluation of sequential selection algorithms. In *Proceedings of the Fifth International Workshop on Artificial Intelligence and Statistics*, pages 1–7.

Aha, D. W., Kibler, D., and Albert, M. K. (1991). Instance-based learning algorithms. *Machine Learning*, 6:37–66.

Almuallim, H. and Dietterich, T. (1991). Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*. AAAI Press/The MIT Press.

Almuallim, H. and Dietterich, T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2):279–305.

Bobrowski, L. (1988). Feature selection based on some homogeneity coefficient. In *Proceedings of the Ninth International Conference on Pattern Recognition*, pages 544–546.

Callan, J. P., Fawcett, T. E., and Rissland, E. L. (1991). An adaptive approach to case-based search. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence*, pages 803–808.

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

Dash, M. (1997). Feature selection via set cover. In *Proceedings of IEEE Knowledge and Data Engineering Exchange Workshop*, pages 165–171. IEEE Computer Society.

Dash, M. and Liu, H. (1997). Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3).

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

Doak, J. (1992). An evaluation of feature selection methods and their application to computer security. Technical report, Davis CA: University of California, Department of Computer Science.

Domingos, P. (1997). Why does bagging work? a Bayesian account and its implications. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 155 – 158. AAAI Press.

Ichino, M. and Sklansky, J. (1984a). Feature selection for linear classifier. In *Proceedings of the Seventh International Confernce on Pattern Recognition*, volume 1, pages 124–127.

Ichino, M. and Sklansky, J. (1984b). Optimum feature selection by zero-one programming. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-14(5):737–746.

Jain, A. and Zongker, D. (1997). Feature selection: Evaluation, application, and small sample performance. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(2):153–158.

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher.

Kira, K. and Rendell, L. (1992a). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 129–134. AAAI Press/The MIT Press.

Kira, K. and Rendell, L. (1992b). A practical approach to feature selection. In Sleeman and Edwards, P., editors, *Proceedings of the Ninth International Conference on Machine Learning (ICML-92)*, pages 249–256. Morgan Kaufmann.

Kohavi, R. and John, G. (1998). *The Wrapper Approach*, pages 33 – 50. In (Liu and Motoda, 1998).

Kohavi, R. and Sommerfield, D. (1995). Feature subset selection using the wrapper method: Overfitting and dynamic search space topology. In *Proceedings: First International Conference on Knowledge Discovery & Data Mining*, pages 192 – 197. Morgan Kaufmann Publishers.

Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In Saitta, L., editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 284–292. Morgan Kaufmann Publishers.

Kononenko, I. (1994). Estimating attributes : Analysis and extension of RELIEF. In *Proceedings of the European Conference on Machine Learning*, pages 171–182.

Langley, P. (1994). Selection of relevant features in machine learning. In *Proceedings of the AAAI Fall Symposium on Relevance*. AAAI Press.

Liu, H. and Dash, M. (1997). Hybrid search of feature subsets. Technical Report 9/97, Information Systems and Computer Science Department, National University of Singapore, Lower Kent Ridge Road.

Liu, H. and Motoda, H., editors (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Liu, H., Motoda, H., and Dash, M. (1998). A monotonic measure for optmial feature selection. In *Machine Learning: ECML-98*. Springer-Verlag.

Liu, H. and Setiono, R. (1996a). Feature selection and classification - a probabilistic wrapper approach. In *Proceedings of the Ninth International Conference on Industrial and Engineering Applications of AI and ES*, pages 419–424.

Liu, H. and Setiono, R. (1996b). A probabilistic approach to feature selection - a filter solution. In Saitta, L., editor, *Proceedings of International Conference on Machine Learning (ICML-96)*, pages 319–327. Morgan Kaufmann Publishers.

Liu, H. and Setiono, R. (1998). Scalable feature selection for large sized databases. In *Proceedings of the Fourth World Congress on Expert Systems (WCES'98)*. Morgan Kaufmann Publishers.

Modrzejewski, M. (1993). Feature selection using rough sets theory. In Brazdil, P., editor, *Proceedings of the European Conference on Machine Learning*, pages 213–226.

Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 190–198, New Brunswick, New Jersy. Morgan Kaufmann.

Mucciardi, A. N. and Gose, E. E. (1971). A comparison of seven techniques for choosing subsets of pattern recognition. *IEEE Transactions on Computers*, C-20:1023–1031.

Narendra, P. and Fukunaga, K. (1977). A branch and bound algorithm for feature subset selection. *IEEE Trans. on Computer*, C-26(9):917–922.

Oliveira, A. L. and Vincentelli, A. S. (1992). Constructive induction using a non-greedy strategy for feature selection. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 355–360, Aberdeen, Scotland. Morgan Kaufmann.

Pudil, P., Novovicova, J., and Kittler, J. (1994). Floating search methods in feature selection. *Pattern Recognition Letters*, 15:1119–1125.

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

Schlimmer, J. C. (1993). Efficiently inducing determinations : a complete and systematic search algorithm that uses optimal pruning. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 284–290.

Segen, J. (1984). Feature selection and constructive inference. In *Proceedings of the Seventh International Conference on Pattern Recognition*, pages 1344–1346.

Setiono, R. and Liu, H. (1997). Neural network feature selectors. *IEEE Trans. on Neural Networks*, 8(3):654–662.

Sheinvald, J., Dom, B., and Niblack, W. (1990). A modelling approach to feature selection. In *Proceedings of the Tenth International Conference on Pattern Recognition*, volume 1, pages 535–539.

Siedlecki, W. and Sklansky, J. (1988). On automatic feature selection. *International Journal of Pattern Recognition and Artificial Intelligence*, 2:197–220.

Siedlecki, W. and Sklansky, J. (1989). A note on genetic algorithms for large-scale feature selection. *Pattern Recognition Letters*, 10:335–347.

Skalak, D. B. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 293–301, New Brunswick, NJ. Morgan Kaufmann.

Vafaie, H. and Imam, I. F. (1994). Feature selection methods: genetic algorithms vs. greedy-like search. In *Proceedings of the International Conference on Fuzzy and Intelligent Control Systems.*

Wang, K. and Sundaresh, S. (1998). *Selecting Features by Vertical Compactness of Data*, pages 71 - 84. In (Liu and Motoda, 1998).

Xu, L., Yan, P., and Chang, T. (1988). Best first strategy for feature selection. In *Proceedings of the Ninth International Conference on Pattern Recognition*, pages 706–708.

Notes

1. The problem is of two classes. A many-class (say k classes) problem can be changed to k two-class problems.

2. LVF only keeps the better subsets according to the criteria such as inconsistency rates and number of features in a subset.

3. This idea of trying a portion of data first is similar to windowing in ID3.

5 EVALUATION AND APPLICATION

No matter how great a tool is, if it is applied to a wrong problem, it may delay problem solving, or even worse, cause harm. Experts are needed to match tools with problems. They are not only just domain experts who know a lot about the problems they are facing, but experts who are familiar with feature selection methods. If it is not impossible for us to have such experts, these experts are rare to find. The matching problem still exists regardless of whether we can find such experts or not. The second best solution is to abstract both tools and problems. By relating tools with problems, hopefully, we can help solve the matching problem. Abstracting problems can be done by domain experts according to characteristics of data; abstracting tools requires some measures that can tell us how *good* each method is and under what *circumstances* it is good. In order to wisely apply feature selection methods, we need to first discuss their performance.

5.1 PERFORMANCE ASSESSMENT

When we evaluate a new software system (a feature selection method in this context), the usual questions we pose are (1) Does it work? (2) How well does

it work? and (3) Under which conditions does it work? If it works, we are intrigued to know whether it works better than its peers; and if it works well, we'd like to know when it does *not* work. Performance assessment can lead us to find answers to the questions.

Performance is an aspect of behavior that can be measured quantitatively (i.e., by some value). Assessment can tell what the value is. To judge a system's performance, we need to put in perspective the objectives of the system. As to feature selection for classification, the three common objectives are listed below. The first two are about classifiers, and the last one is of data reduction.

1. *Generalizability.* A classifier is generalized from data for two essential purposes: (1) it summarizes data into some structural representation such as decision trees, rules, and so on[1]; and (2) by doing so, it can predict unseen instances better than direct table look-up (treating the data as a table). Generalizability is about improving the predictive accuracy of a classifier with structural representation.

   The subsequent questions are now "What is an error?" and "How to measure it?" The two questions are trickier than they look. We will devote Section 5.2 to them.

2. *Comprehensibility.* This is closely related to the first purpose of classifier induction - learning structural representations. It is generally agreed that data sets, even if they are simple, are difficult for humans to find regularities and understand underlying relationships. One of the reasons is that there are too many hypotheses to verify and the focus span of a human being is rather limited. Structural representations can, however, improve human's comprehensibility of the data since they are more concise than the data[2]. Although it is a subjective matter which structural representation is more understandable than the other, it is reasonable to pursue the simplest possible for a given structural representation. In other words, the simpler a representation, the easier it is to comprehend. For example, other things being equal, we'd prefer a smaller decision tree than a larger one.

3. *Data simplicity.* The complexity of data can normally be measured in two dimensions, the number of features and the number of instances. The latter is usually determined by the number of features and the number of values each feature can take. For example, a dataset with two binary features can be no more than four instances. The more features there are, the more complex a dataset is, if we are talking about data of the same type. Selecting some features over the others means dimensionality reduction. Therefore, we simplify the data description. Simplified data usually gives rise to simpler structural representation, so better comprehensibility. However, evidence

shows that it is not true that the more the reduction of the number of features, the better the comprehensibility. There must be an optimal point for us to *stop* reducing dimensionality so that the data is sufficiently simple, but not too simple. Only the classifier that can predict well is worthy of our comprehension. Therefore, predictive accuracy can be used as one stopping criterion for dimensionality reduction.

From the view point of feature selection, we want to increase a classifier's generalizability and comprehensibility through data simplification. Obviously, considering any one objective alone is insufficient. The three objectives are intertwined. It is ideal that we could reduce the data, improve accuracy of a classifier, as well as obtain simplified representation induced by the classifier. It is not an easy task to achieve all three objectives. (Murphy and Pazzani, 1994) showed, for example, that decision trees with best predictive accuracy may not be the simplest trees. This tells us that this is a multi-dimensional optimization problem and we may need to trade off one objective for another. It is a matter of focus vs. balance. If we cannot achieve all the three objectives, we may focus on two of them according to the practical needs. In design of classifiers, for example, predictive accuracy is usually the foremost concern, since a lousy classifier cannot even summarize the data well, let alone providing clues about regularities in the data. Hence, unless otherwise specified, generalizability is our first priority in performance assessment. Another direction is to devise an overall measure of mixed dimensions. The importance of each dimension is reflected in the weights attached to the dimensions.

For the above three dimensions (Generalizability, Comprehensibility and Data simplicity), we can derive three assessment measures to be evaluated independently:

Predictive accuracy - this is a commonly used measure to assess generalizability: *the higher, the better* because a classifier with high accuracy means it tests well on unseen data that is not involved in inducing the classifier;

Complexity of representations - this is used to indirectly assess the comprehensibility of a classifier: *the simpler, the better* because simplier things are easier to understand; and

Number of features selected - this is a measure for assessing the size of data: *the smaller, the better* because a smaller number of features, implying as well a smaller number of instances required, means fewer potential hypotheses, faster learning, and simpler end results.

We can evaluate a feature selection method along these three dimensions. If a method can achieve best scores in all three, it is obviously the best method

overall. Otherwise, we need to compare along each dimension and check which method gets better scores in more dimensions. With these three dimensions in mind, we should also consider some important practical factors:

- *Speed of feature selection methods.* How fast can we obtain results? This is particularly pertinent when we deal with large data. We are usually constrained by the computing resources or pressed by deadlines. In such situations, we may prefer faster methods, or any-time algorithms.

  Some questions of our concern are "Can we afford additional time?", "What if a feature selection method is more run-time costly than a classifier?"

- *Generality of selected features.* This issue brings us back to the purposes of feature selection stated in Chapter 1. The generality of selected features is not always the most important. However, we do care about it. In these cases, not only do we want to improve a classifier's performance, also we would like to identify which features are relevant and which are irrelevant or redundant in order to reduce the data collection effort.

The ultimate measure is strongly linked to the above five factors. Put it another way, these factors can guide us in choosing a proper feature selection method. In order to obtain values of the measures and compare them between different feature selection algorithms, we need to rely on statistics.

5.2 EVALUATION METHODS FOR CLASSIFICATION

We first review basic statistics and hypothesis testing, next define classification errors and their assessments, and then introduce commonly used evaluation methods.

5.2.1 Basic statistics

The purpose of this part is to gather the basic statistics here, make it convenient for the reader to understand what is behind the performance evaluation. For the details, the reader is referred to (Mendenhall and Sincich, 1995, Cohen, 1995). The first book is about statistics for engineering and the sciences, and the second book is about empirical methods for artificial intelligence. Since data sets are samples of populations, we consider only statistics with samples. In the following, a variable (feature) is denoted as X, its number of values n.

Frequency histogram It is a bar graph of frequency distribution. For the example of our sunburned data in Table 1.1 of Chapter 1, a frequency histogram for feature *Hair* is shown in Figure 5.1. Hair color *blonde* appears 4 times, for example. A frequency histogram is one way of visualizing a feature.

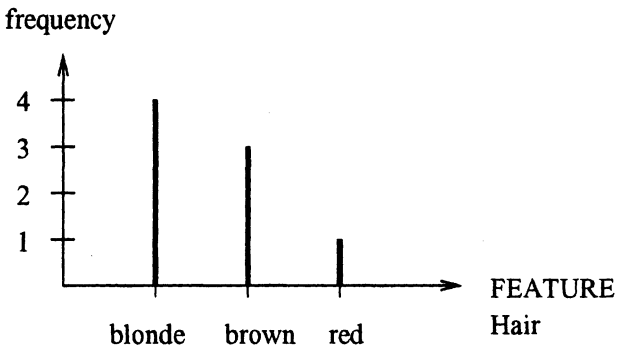


Figure 5.1. A frequency histogram for feature Hair of the sunburned data.

Mean It is the sum of all values in a feature divided by the number of values.

$$\bar{X} = \frac{\sum X}{n} \tag{5.1}$$

The mean for a discrete random variable (feature) is called the expected value $E(X)$.

$$E(X) = \sum XP(X) \tag{5.2}$$

It is essentially a weighted average of all of the possible values of the random variable, with the probability values ($P(X)$) used as the weights.

Whereas a mean identifies the typical value in a feature, it does not provide any information about the extent to which the values differ from one another.

Variance It is a measure of dispersion that takes into consideration the difference between each value in a feature and the mean of the feature. The sample variance is represented by s^2

$$s^2 = \frac{\sum (X - \bar{X})^2}{n - 1} \tag{5.3}$$

The sample variance includes a correction factor ($\frac{1}{n-1}$) so that it is an unbiased estimator of the population variance. The sample standard deviation is represented by s:

$$s = \sqrt{\frac{\sum (X - \bar{X})^2}{n - 1}} \tag{5.4}$$

Both s^2 and s require the determination of deviations from the mean $\bar{X}$. So, people usually use their computational formulae as follows.

$$s^2 = \frac{\sum X^2 - ((\sum X)^2/n)}{n-1} \tag{5.5}$$

$$s = \sqrt{\frac{\sum X^2 - ((\sum X)^2/n)}{n-1}} \tag{5.6}$$

The variance for a discrete random variable X is denoted by $V(X)$.

$$V(X) = \sum (X - E(X))^2 P(X) \tag{5.7}$$

Its computational form is

$$V(X) = \sum X^2 P(X) - (\sum XP(X))^2 \tag{5.8}$$

Covariance It is a measure of linear association between two features X and Y.

$$\mathbf{cov}(X,Y) = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{n-1} \tag{5.9}$$

If $\mathbf{cov}(X,Y)$ is 0, there is no linear association between X and Y; if it is positive, ascending values of one feature are matched with ascending values of the other; if it is negative, ascending values of one feature are matched with descending values of the other. The trouble with $\mathbf{cov}(X,Y)$ is that it can be an arbitrarily large positive or negative number, depending on the variances of X and Y. When we compare between $\mathbf{cov}(X,Y)$ and $\mathbf{cov}(Z,Y)$ for example, we need to standardize the values first.

Correlation It is a normalized form of a covariance and obtained as follows:

$$r_{X,Y} = \frac{\sum (X - \bar{X})(Y - \bar{Y})}{(n-1)s_X s_Y} \tag{5.10}$$

where s_X is the sample standard deviation for feature X. Its computational form is

$$r_{X,Y} = \frac{\sum XY - \frac{(\sum X)(\sum Y)}{n}}{(n-1)s_X s_Y} \tag{5.11}$$

where s_X and s_Y can be computed as in Equation 5.6.

Hypothesis Testing It is one kind of statistical inference that answers a yes/no question about a population and assesses the probability that the answer is wrong. In comparing two classifiers or assessing the effect of feature selection on a classifier, hypothesis testing is a commonly used tool for us to draw the

1. Formulate the null hypothesis (H_0) and the alternative hypothesis (H_1).
2. Specify the level of significance to be used.
3. Establish the critical value or values of the sample mean.
4. Determine the value of the sample mean (z after conversion).
5. Make the decision: to reject or to accept.

Table 5.1. A hypothesis testing procedure about the sample mean.

conclusion. Table 5.1 shows a procedure of hypothesis testing about the sample mean.

The null hypothesis (H_0) specifies the parameter value (μ_0) to be tested. The alternative hypothesis (H_1) specifies the parameter value or values which are to be accepted if the null hypothesis is rejected. The null hypothesis is given *the benefit of the doubt* - it will be accepted unless the sample result is clearly inconsistent with it[3]. The design of null hypothesis and alternative hypothesis is influenced by whether a test is of one-tail or two-tail.

A one-tail test is used when we are concerned about possible deviations in only one direction from the hypothesized value. A two-tail test is used when we are concerned about a possible deviation in either directions from the hypothesized value. A one-tail test can tell us if one is greater than the other; and a two-tail test can answer questions like whether the two things are different. For example, null and alternative hypotheses can be as follows:

	One-Tail	Two-Tail
H_0:	$\mu = \mu_0$	$\mu = \mu_0$
H_1:	$\mu = \mu_1(\geq \mu_0)$	$\mu = \mu_1(\neq \mu_0)$

Note that the alternative hypothesis is not necessarily the negation of the null hypothesis. H_1 can be any hypothesis that is not H_0.

Level of significance It is the statistical standard used as the basis for rejecting the null hypothesis. If a 5% level of significance is specified, the rejection of H_0 comes with the probability of 5% for committing a type I error (rejecting a true null hypothesis). Since the probability of type I error is designated by α and is equal to the level of significance, α also specifies the level of significance. The other type of error is accepting a false null hypothesis, it is called a type II error. The probability of a type II error is designated by β. The probability

of a type II error can be reduced without allowing the probability of a type I error to increase by increasing the sample size.

Because of the central limit theorem (refer to (Kazmier and Pohl, 1987), for example), we can adopt the normal distribution (corresponding to a Z-test) as the sampling distribution in hypothesis testing when number of instances $n \geq 30$; use the student's t distribution (corresponding to a t-test) when $n < 30$. It is generally more convenient to convert the sample mean into a z value ($z = \frac{\bar{X}-\mu_0}{s_{\bar{X}}}$) for the normal distribution. Whether we need to specify one or two critical values depends on whether a one-tail or two-tail test is involved, the former requires one critical value, the latter two values ($\pm$ critical value). A critical value is in the same units of measurement as the sample mean (z now) and identifies the value of z that would lead to the rejection of H_0 at the designated significance level (α). The value of the sample mean is calculated from the sample and converted to a value of z. The value of z is compared with the critical value(s) in order to make the decision (rejection or acceptance of H_0).

When use of the t distribution is appropriate, t replaces z as the test statistic.

$$t = \frac{\bar{X} - \mu_0}{s_{\bar{X}}} = \frac{\bar{X} - \mu_0}{s/\sqrt{n}} \tag{5.12}$$

In the t test, the degree of freedom (df) of the sample is required which is the number of instances (P) minus one, or $df = P - 1$.

One alternative to the above hypothesis testing about mean is the p-value approach. The observed significance level, or p-value, for a specific statistical test is the probability (assuming H_0 is true) of observing a value of the test statistic that is at least as contradictory to the null hypothesis, and supportive of the alternative hypothesis, as the one computed from the sample data (Mendenhall and Sincich, 1995). Instead of using the critical value(s) and selecting α *a priori* and then conducting a test as outlined above, we can compute and report the value of the appropriate test statistic and its associated p-value. It is left to the reader to judge the significance of the result. By convention, one does not reject the null hypothesis unless $p < 0.05$. An alternative is that the null hypothesis will be rejected only if the p-value is less than the fixed significance level α chosen by the reader. For a one-tail test, the p-value is 0.5 minus the area determined by z in the normal distribution (The table of normal curve areas can be found in Appendix II page 1094 (Mendenhall and Sincich, 1995). The whole area is 1, and the half of it is 0.5); for a two-tail test, the p-value is doubled when reference is made to a symmetrical probability distribution such as the normal distribution. In order to compare between the two approaches, the critical values are points on the horizontal axis and determined by α, but the p-value is an area determined by z.

Here is an example to show how the two approaches differ in a one-tail test. Assume there are 9 instances, the significance level α required is 0.05. We calculate that $\bar{X} = 200$, $s = 50$, $s_{\bar{X}} = 50/\sqrt{9}$, and $\mu_0 = 120$. So, $df = 9-1 = 8$, $t = \frac{(200-120)}{50/3} = 4.8$. The null hypothesis is $H_0 : \mu = \mu_0$ and the alternative hypothesis is $H_1 : \mu = \mu_1 (> \mu_0)$. Referring to Table 7 - Percentage points of the t distribution in (Murdoch and Barnes, 1993), for $\alpha = 0.05, df = 8$, the critical value is 1.86. Since $(t = 4.8) > 1.86$, t is in the region of rejection (the tail section), so we reject the null hypothesis and adopt the alternative hypothesis. Let's look at the p-value approach. Using the same Table 7, at the entry of $df = 8$, we notice that 4.8 is between 4.501 (for $\alpha = 0.001$) and 5.041 (for $\alpha = 0.0005$). Therefore, $0.0005 < p < 0.001$, certainly, $p < 0.05$, so, H_0 is rejected. A p-value tells us the probability of making a type I error.

After we know how the one-sample testing works, we are ready for two hypothesis testing procedures particularly useful in our performance assessment. For example, if we want to see the effect of feature selection on a classifier, we can check a classifier's accuracy *before* and *after* applying feature selection in cross validation.

Testing the difference between two means In this testing, our null hypothesis is that the two means are not different. That is,

$$H_0 : (\mu_1 - \mu_2) = 0,$$

$$H_1 : (\mu_1 - \mu_2) \neq 0.$$

The necessary formulas are shown below.

$$z = \frac{\bar{X}_1 - \bar{X}_2}{s_{\bar{X}_1 - \bar{X}_2}} \tag{5.13}$$

$$s_{\bar{X}_1 - \bar{X}_2} = \sqrt{s^2_{\bar{X}_1} + s^2_{\bar{X}_2}} \tag{5.14}$$

$$s_{\bar{X}_1} = s_1/\sqrt{n_1} \quad \text{and} \quad s_{\bar{X}_2} = s_2/\sqrt{n_2} \tag{5.15}$$

When the samples are small ($n < 30$), we should use the t test by replacing z with t in Equation 5.13 above.

This procedure is appropriate for situations in which two random samples are independently obtained. In a comparison of two sample means, such samples are sometimes collected as pairs of values. For example, before and after feature selection application, the samples collected from k-fold cross validation can be dependent, i.e., accuracy for the same block of data. Then, we need to consider the paired hypothesis testing.

Testing the difference between two means based on paired observations Let d be the difference between each pair of values, H_0 be that the average difference in the population is zero. Thus, the two sets of samples are reduced to one set of d values. This is much like the one-sample hypothesis testing with $H_0 : \mu_d = 0$.

$$\bar{d} = \frac{\sum d}{n} \tag{5.16}$$

$$s_d = \sqrt{\frac{\sum (d - \bar{d})^2}{n-1}} = \sqrt{\frac{\sum d^2 - n\bar{d}^2}{n-1}} \tag{5.17}$$

$$s_{\bar{d}} = \frac{s_d}{\sqrt{n}} \tag{5.18}$$

$$z = \frac{\bar{d} - \mu_d}{s_{\bar{d}}} \tag{5.19}$$

where z is replaced with t when $n < 30$, and μ_d is the hypothesized difference between the two means. Since we want to test the null hypothesis that there is no difference between the means of the two populations ($H_0 : \mu_d = 0$), the test statistic z is simplified to

$$z = \frac{\bar{d}}{s_{\bar{d}}}. \tag{5.20}$$

With $z, \bar{d}$ and α, we can conduct hypothesis testing to reject or not to reject H_0. When $n < 30$, this procedure is called the paired t test.

5.2.2 Error rate and how to measure it

The objective of a classifier is to classify/predict new instances correctly. The commonly used measure is predictive accuracy. Since new instances are supposed not to be seen by the classifier in its learning phase, we need to estimate a classifier's predictive accuracy. Following (Weiss and Kulikowski, 1991), we use apparent and true error rates. The true error rate is statistically defined as the error rate of the classifier on an asymptotically large number of new cases that converge in the limit to the actual population distribution. The apparent error rate is the the error rate of the classifier on the training data that were used to induce the classifier. So, what is an error?

An error in classification is simply a misclassification: the classifier under consideration is presented with an instance and it classifies the instance wrongly. If all errors are of equal importance, an error rate (r) is the number of errors

Table 5.2. A confusion matrix for three classes (0, 1, and 2).

	True Class (j)		
Classification (i)	0	1	2
0	25	0	0
1	0	23	3
2	0	2	22

($\#E$) divided by the number of instances ($\#I$). The accuracy (a) is one minus the error rate.

$$r = \frac{\#E}{\#I} \tag{5.21}$$

$$a = 1 - r = \frac{\#I - \#E}{\#I} \tag{5.22}$$

The types of errors can be as many as $m^2 - m$ ($= \frac{(1+(m-1))(m-1)}{2} \times 2$), where m is the number of classes. If there are only two classes (positive or negative, i.e., T, F), we can have two types of errors: (1) it is T, but classified as F - false negative, or (2) it is F, but classified as T - false positive. If there are more than two classes, the types of errors can be summarized in a confusion matrix as shown in Table 5.2. There are 6 types of errors ($= 3^2 - 3$). Every class supposedly contains 25 instances in this example. Only for class 0, however, all 25 instances are correctly classified as so; for classes 1 and 2, two instances of class 1 are wrongly classified as class 2, and three instances of class 2 are incorrectly classified as class 1. Each error is denoted as e_{ij} - classifying class j into class i.

So far we consider that every error is equally bad. When different types of errors are associated with different risks or costs, we need to resort to a risk or cost matrix (an example in Table 5.3) which assigns a cost c_{ij} to a type of errors; c_{ij} is the cost for an error made by wrongly classifying j into i where $i \neq j$. The total cost C for all types of errors can be calculated as

$$C = \sum_i \sum_j e_{ij} c_{ij}, \tag{5.23}$$

where e_{ij} are found in the confusion matrix (Table 5.2).

We have detoured a bit and discussed how to include costs in performance assessment. Recall that our goal is to estimate true error rates, which can empirically be approximated by a very large number of new instances gathered independently from the instances used to design the classifier. Unfortunately, we

Table 5.3. A cost matrix for various types of errors.

	True Class (j)		
Classification (i)	0	1	2
0	0	2	3
1	1	0	1
2	1	3	0

usually do not have such luxury to have a very large number of new instances. On the other hand, apparent error rates tend to be biased optimistically (i.e., lower than true error rates) and are not a good estimator of true error rate since the classifier has seen all the instances when it reports its apparent error rate. Here we describe several methods that are commonly used to estimate true error rate.

The simplest procedure is to split the data into two parts: a training set and a test set randomly drawn from the data (usually two thirds for training and the remaining one third for testing). Only the training set is used in learning the classifier, and then it is tested on the test data to get a so called "honest" error rate because the test data has not been seen by the classifier in its learning phase. How does this procedure work in estimating true error rate? Analysis reveals problems: for sure it is not suitable for small to medium sized data. In some sense, a classifier's performance is determined by the data available to it. A change in the data usually results in a change in a classifier. In general, the more the data, the better a classifier learns. When we hold out one third of the data, there is less data for learning. Besides, one third of small or medium sized data would not be large enough to estimate the true error rate. It is obvious that one way out of this dilemma is to hold out as few instances as possible. Holding one out is the best we can do. This is a procedure called leaving-one-out by statisticians.

It is a simple and straightforward procedure. For N instances, each instance is held out once in a loop of N iterations. For each iteration, the $N-1$ instances are used as the training data, one held out as the test data. At the end of the loop, the error rate is the total number of errors on the N test instances divided by N. Although the leave-one-out error rate estimator is an almost unbiased estimator of the true error rate, it does have some problems. One is that it is computationally expensive since for N instances, N classifiers have to be learned to obtain the error rate. Another is that it is an estimator with high variance. Both the bias and variance of an error rate estimator contribute to the

inaccuracy and imprecision of the error rate estimate. For the second problem, researchers (Kong and Dietterich, 1995, Kohavi and Wolpert, 1996) studied the decomposition of bias and variance, and suggested that one could employ bagging (Breiman, 1996) to reduce variance (Domingos, 1997). (Friedman, 1997) suggests that certain types of (very high) bias can be canceled by low variance to produce accurate classification.

For the first problem, when the data set is large and leaving-one-out is computationally too expensive, an intuitively solution is to leave more out, for example, leaving some percentage of data out. (Breiman et al., 1984) reported, through their extensive experiments on the CART program, that leaving 10 percent of data out seemed to be adequate and accurate. This testing procedure is called k-fold cross validation, where k is one divided by the percentage hold-out data. If we hold out 10 percent of data, $k = 1/0.10 = 10$; for holding out 20% of data, k is 5. In performing k-fold cross validation, one randomly divides the data into k partitions (folds); each fold is reserved in turn for testing in a loop of k iterations, the rest is used for learning the classifier. It is just like leaving-one-out, but now it leaves one fold out instead of one instance. The advantage of k-fold cross validation is that it only induces k classifiers, no matter how large the data set is.

It is noticed that k-fold cross validation has less variance than leaving-one-out. However, the problem of high variance is still with k-fold cross validation. This could be compensated by conducting multiple runs of k-fold cross validation . A procedure of 3 runs of 10-fold cross validation is basically a 10-fold cross validation wrapped up in three loops. In the beginning of each loop, the data is shuffled. This procedure produces results of 30 classifiers.

If we would like to conduct some hypothesis testing (for example, to see the effect of before and after applying feature selection), we may ask what is k in k-fold cross validation. Is it true that the larger k is, the better? The answer is negative: for hypothesis testing, samples should be no larger than required to show an effect, and this is determined by what kind of significance level (α) one wants to achieve. As shown by (Cohen, 1995), if we can reject a null hypothesis with a sample of size N, nothing is gained by collecting more data, or making k larger. Therefore, without particular reasons, we should be satisfied with $k = 10$, i.e., a 10-fold cross validation. But if a data set is small, what should be done if one wants to reduce variance?

One may apply the Bootstrapping procedure when the data set is small. Simply put, it is sampling with replacement: it randomly picks N instances from the data set to form the training set, and leaves the unpicked instances in the testing set. As suggested in (Weiss and Kulikowski, 1991), one should repeat this bootstrapping procedure for 200 times.

5.2.3 *Commonly used evaluation methods*

With the statistical procedures introduced above, we now discuss the commonly used evaluation methods in classification using these procedures (hypothesis testing, cross validation, and so on).

Learning curves are used to show trends. For example, if we are interested to know if the more the instances, the better the results of classification, we can randomly divide the data into m chunks and observe the change (increase or decrease) of predictive accuracy of a classifier by using 1, 2, ..., and all m chunks of data to obtain m accuracy rates. As shown in Figure 5.2, we observe the constant increase of accuracy when more data is used.

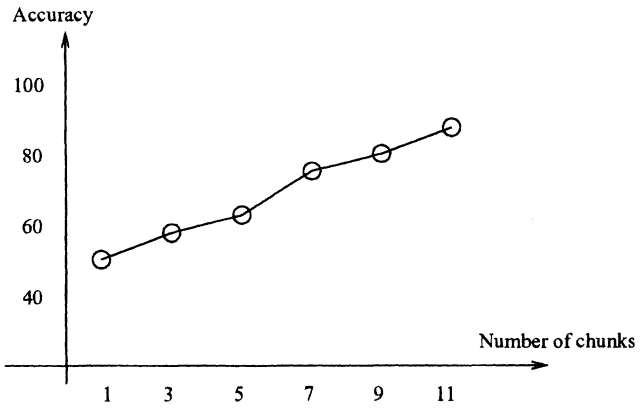


Figure 5.2. A learning curve: observing the change of accuracy with increasing instances. One chunk of data is roughly equal to N/m instances, and $m = 11$.

We may also use the learning curve to show the effect of added features if we have a ranked list of features. Starting with one feature (the most relevant one first) and gradually adding next most relevant feature one by one, we calculate its accuracy rate. If the ranked list of features truly represents the nature of these features, we should expect a sort of bell curve that would show that till some stage, adding features will increase the accuracy, after that stage, adding more features will decrease the accuracy, due to the introduction of irrelevant or redundant features and data overfitting by the classifier. In short, learning curves are used to display the performance variability.

Comparison using different classifiers Because different classifiers inherit different learning bias, it is often seen in an empirical study that a group of different kinds of classifiers (decision trees or neural networks, for example) are

engaged to see the effects on different classifiers. Cross validation can be used to make samples for hypothesis testing about means (paired or not paired). For instance, the null hypothesis can be that there is no difference between different classifiers, or the effect on each classifier is the same.

"Before-and-after" comparison This is another evaluation method that can tell us whether the change of data (e.g., with or without feature selection) affects the performance of the classifier. Hypothesis testing can tell us whether the changed performance, before and after the data is changed, is statistically significant (recall that the p-value method will indicate how significant it is).

Repeating experimental results For a deterministic classifier or a feature selector, this is not an issue since the results will be the same no matter how many times a learning algorithm is run over the same data. It is a different story if a non-deterministic classifier is involved. How can we be sure that one run is not significantly different from another? A common method is to run a couple of dozen times and to observe if the good performance persists or occurs frequently. Alternatively, a paired hypothesis test can be conducted between every two different runs and see how many pairs are statistically different. This method would be much more computationally expensive than simply running a learning algorithm for a fixed number of times. We can also apply this approach to checking if we can repeat the selected features for a non-deterministic feature selector.

If necessary, we may combine some of the above evaluation methods. One combination could be learning curve with "before-and-after", or comparison among different classifiers with "before-and-after". We will show some combinations in Section 5.4 "Evaluation: Some Examples".

5.3 EVALUATION OF SELECTED FEATURES

Clearly, the above evaluation methods can easily be applied to evaluating how effective feature selection is to a learning algorithm. That is, we can see if using selected features can change the performance of a learning algorithm for better. The idea is that only relevant features can at least retain the performance, and removing irrelevant and/or redundant features will only improve the performance. The evaluation approach under this principle is called *indirect evaluation*. Since some data sets are collected with known characteristics, we can adopt a more straightforward approach - *direct evaluation*. In other words, these data sets can be used to assess feature selection methods, without building any classifier, whether relevant features are selected. In this case, we directly measure the result of feature selection using the known characteristics. If this is not possible as in many real world applications, some indirect measures should be employed.

5.3.1 Direct evaluation of features

This is the most objective and straightforward way of evaluating features. If we know *a priori* that some features are relevant, we can expect a good feature selection method to find these features either in the front of a ranked list or in the minimum subset. Knowing that some features are irrelevant or redundant can also help. We simply do not expect these features to be in the front of a ranked list or to be in the minimum subset at all.

For non-deterministic feature selection methods, we may need to do more experiments to see if the selected features can be frequently reselected. Although direct evaluation is ideal in checking whether a feature selection method works well, it is common that we do not have *a priori* knowledge. Hence we have to rely on indirect evaluation.

5.3.2 Indirect evaluation of features

Instead of checking whether features are relevant or not, we now engage a learning algorithm - learning will result in a classifier and monitor the change of the classifier's performance with the change of features. Let's concentrate on error rates as the performance indicator at the moment. Under normal circumstances, 10-fold cross validation is recommended to estimate the true error rates. In some recent work, we can see that multiple runs of cross validation and bootstrapping are used in verification to reduce variations in error estimation.

For ranked lists of features, we may plot learning curves about the decreasing or increasing of errors when adding extra features one by one according to their relevance. If 10-fold cross validation is performed, its average error rates can be plotted. The paired t test (since the sample size is 10) can be done to check the statistical significance of every two neighboring feature subsets (S_i and S_j), where S_j has one more feature than S_i, and that feature is the most relevant among the remaining features ($F - S_i$) and F is the full set of features.

For minimum subsets of features, we can simply conduct the "before-and-after" experiment. One set of results is the error rates of the classifier learned on the full set of features. The other set is the one learned on the selected features. The paired t test again can be done to see how statistically significant the change of error rates is after we use selected features compared to using the full set of features. Now we may face the following question.

5.3.3 Which type of evaluation should we use?

Obviously, we can always apply direct evaluation in the experiments. In practice, however, direct evaluation is *rarely* used. The reason is simple: if we know *a priori* which features are relevant or not, we can just choose these relevant

ones without applying any feature selection methods. When we think of using feature selection methods, it is usually because we have no idea which features are relevant or irrelevant, so we have to turn to indirect evaluation. But, how can we be sure that the features that bring about lowest error rates are relevant ones? One way of boosting our confidence in the selected features is to perform both direct and indirect evaluation for the data sets we know *a priori* their characteristics. It is a two-step procedure: (1) we check if a feature selection method can find only the known relevant features; and (2) we conduct cross validation and hypothesis testing to see if the performance change is significant by using these selected features. If the experimental results are affirmative, we can feel confident to accept that the experimented feature selection method can find relevant features and these features do help improve the classifier's performance. One might think a one-step procedure can also solve this confidence problem. That is, just check if the features that produce the lowest error rates are the known relevant features or not. Doing so, we limit the feature selectors to one type only - the wrapper model; and there is no filter feature selector possible in this picture[4]. But our task is to evaluate any feature selector. Hence, the two-step procedure is more suitable.

We can also take a pragmatic stand: since one of the ultimate goals of feature selection is to improve predictive accuracy, as long as the accuracy improves, we just accept the fact and admit that only the features used are relevant. But how can we know that a set of features determined by the experiments with one classifier can also be good for another classifier? If possible, we often do hope to select the relevant features independent of any classifier. In such cases, at least two classifiers with *different* learning bias should be used to see if the same set of selected features can improve the performance of both classifiers. For example, we can use a decision tree learning algorithm and a neural network learning algorithm.

5.4 EVALUATION: SOME EXAMPLES

Evaluation experiments can be extensive and time-consuming. Without a purpose, experiments can go astray. Here we use some examples to demonstrate what are the goals of our experiments and how to design, organize and conduct these experiments.

5.4.1 Experiment purposes and design

Various feature selection methods are sensitive to different kinds of features. Some methods may be good for selecting relevant features, but insensitive to redundant ones. They would select redundant features as relevant ones. Some methods may not be effective in removing noisy or random features that are

highly correlated with the class feature. A basic experiment of feature selection methods involves data and a classifier. We adopt a goal oriented approach in choosing data sets and classifiers. We spell out the various goals of experiments in explaining how data sets and classifiers are selected.

Selection of data for experiments. There are many data sets available. We can choose to simply test them all, or to adopt a goal-oriented approach by defining goals of experiments and then selecting data sets that can be tested to explain some concerning points. We follow the latter approach and first define the goals of experiments with data. The goals are to find solutions to the following questions:

- **Noise** - Can a feature selection method handle the data with noise?
- **Irrelevance** - Can a feature selection method remove irrelevant features?
- **Redundancy** - Can a feature selection method eliminate redundant features?
- **Correlated randomness** - Can a feature selection method identify random features highly correlated with classes?
- **Interdependency** - Can a feature selection method extract interdependent features that determine the class of an instance jointly?
- **Data type** - Can a feature selection method handle various types of data?
- **Data size** - Is a feature selection method more suitable for small sized data or for large sized data (mainly concerning the number of features)?
- **Scalability** - Can a feature selection method scale well with large data (mainly concerning the number of instances)?

To further simplify the task of differentiating relevant features from irrelevant ones, we choose data sets with known relevant features. The data sets below (summarized in Table 5.4) allow us to verify if a feature selection method can achieve one of the goals above. Most of them can be obtained from the UCI Irvine repository of machine learning databases (Merz and Murphy, 1996).

- **Corral** The data was designed in (John et al., 1994). There are six binary features, A_0, A_1, B_0, B_1, I, and C. Feature I is irrelevant, feature C is correlated to the class label 75% of the time. The Boolean target concept is $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. C4.5 chose feature C as the root. This is an example of data sets in which if a feature like C is removed, a more accurate tree

will result. Among well shuffled 64 (2^6) instances, 32 instances make up the training data, another 32 instances the test data.

- **Monk1, Monk2, and Monk3** The data sets were described and used in (Thrun and et al, 1991). They were used for a performance comparison of different machine learning algorithms. Each data set has six discrete features plus one Boolean class feature. The training and test data sets are separately provided.
  - Monk1. Only three out of six features are relevant.
  - Monk2. All feature are relevant to the target concept. We want to see that a feature selection method does not remove any relevant features.
  - Monk3. Only three out of six feature are relevant, what's more is that in the training data the classes of 5% of the data were wrongly labeled. This data set is used to see how a feature selection method can handle this kind of noise.

- **Parity5+5** The target concept is the parity of five bits. These five features are interdependent in determining classes. The data set contains 10 features, of which 5 are uniformly random (irrelevant). The training set contains 100 instances randomly selected from all 1024 instances. Another independent 100 instances are drawn to form the test set. Most heuristic feature selectors will fail on this sort of problems since an individual feature does not mean anything.

- **Parity5+2** This is a modified version of Parity5+5 by replacing its 6th and 7th columns of random values with its 1st and 2nd columns, respectively. In other words, we introduce two redundant features, in addition to the three (8th - 10th) irrelevant features.

- **Led17** This data is generated artificially by a program at the UCI repository. It contains 24 features among which the first 7 are used to display a value between 0 - 9 in the seven segment display system. The remaining 17 features are generated randomly. All the values are binary except the class which takes a value between 0 and 9 (representable in seven segments). The number of instances to be generated is determined by the user. In our experiment, we use 20000 instances.

- **Iris** There are four continuous features, the class feature has three values (Setosa, Versicolor and Virginica). The length and breadth of both petal and sepal were measured on 50 flowers of each class. In total, there are 150 instances. The data is separated into training and test sets by taking odd

Table 5.4. Summary of data sets used in evaluation of feature selection methods. Notations: C - the number of distinct classes, N - the number of features, S - the size of the data set, S_d - the size of the training data, S_t - the size of the test data. Training and test data sets are split randomly if not specified.

Data	C	N	S	S_d	S_t
Iris	3	4	150	75	75
CorraL	2	6	64	32	32
Monk1	2	6	432	124	432
Monk2	2	6	432	169	432
Monk3	2	6	432	122	432
Par5+5	2	10	1024	100	100
Par5+2	2	10	1024	100	100
Led17	10	24	20,000	5000	5000

and even instance numbers, respectively. Two features (Petal Length and Petal Width) are relevant ones (Michie et al., 1994).

Selection of classifiers for experiments. There are many classifiers available. Following the way we select data sets, we adopt the goal oriented approach to choosing classifiers in experiments. The main objectives are (1) to check accuracy before and after feature selection, and (2) to examine whether the selected features are proper for different classifiers. Probably, the reader may suggest that since we know what are the relevant features, the second objective can easily be achieved by comparing the selected features with the known ones. In spite of the fact we know the relevant features, we wish to confirm the findings from other angles. Thus, even if we do not know the relevant features, as it is normally a case in practice, we can rely on the insights gained in experiments conducted here to interpret the results.

Classifiers below are chosen as representatives of different types of classifiers:

- C4.5 (Quinlan, 1993) is a decision tree induction program that builds decision trees following a divide-and-conquer strategy and uses information gain to select a feature as a node to partition data available to it.
- NBC (Weiss and Kulikowski, 1991) is a simplified version of Bayesian Classifier based on the Bayesian rule assuming individual features are independent from each other.
- SNNS (Zell and et al, 1995) is a comprehensive package of different neural network training algorithms. We use its implementation of back-propagation

in our experiments. The reason to include this neural network algorithm is that it works differently from C4.5 and NBC. It uses all features at each epoch of training.

Some details of these classification algorithms were provided in Section 2.4 of Chapter 2. For more details, the reader should consult the relevant references.

Before we describe and discuss experiments, a side note is worth mentioning. Rob Holte discovered that almost all classification problems in the machine learning community's corpus (Merz and Murphy, 1996) could be solved by attending to a single feature (Holte, 1993). For example, the mushroom problem (edible or poisonous) can be determined by odor. Does this mean that almost all of nature's classification problems are so easily solved, or does it mean that the corpus is unrepresentative? The investigation into the two questions is, however, out of the scope of this work.

Summary. The usual purposes of experimenting with feature selection methods are to see how each feature selection method responds to different kinds of data, to experience how different feature selection methods perform on a data set, and to observe the effects on a classifier before and after feature selection by different methods. In addition, we also wish to demonstrate by example how the various evaluation techniques are used.

5.4.2 Experiments and results

The feature selection methods used in experiments are of various types: complete, heuristic, wrapper, nondeterministic, hybrid, and incremental, using different evaluation measures. They are explained in Table 5.5. The **Wrap1** algorithm is expanded into two algorithms: WSFG - Wrapper of Sequential Forward Generation and WSBG - Wrapper of Sequential Backward Generation. For Las Vegas algorithms, we set the number of runs as 25% of the search space (2^N) for each data set.

The experiments of the feature selection methods in Table 5.5 are organized into two groups: direct evaluation and indirect evaluation. In each group, we organize what we want to verify from experiments into questions (Q1 to Q11) and contemplate some conjectures about what kind of results we expect as outcomes of various experiments. The latter is based on our now better understanding of feature selection methods.

- Direct evaluation.

  Q1 Can the feature selection methods find the known relevant features and remove *noisy*, *irrelevant*, and *redundant* features?

Table 5.5. A summary of feature selection methods used in experimentation. Comp - Complete, Heur - Heuristic, Wrap - Wrapper, Nond - Nondeterministic, Hybr - Hybrid, Incr - Incremental, Fwd - Forward, Bwd - Backward, Rnd - Random.

Method	Search	Direction	Measure	Output
Focus	Comp	Fwd	Consistency	List
ABB	Comp	Bwd	Consistency	Subset
B&B	Comp	Bwd	Consistency	Subset
Relief	Heur	-	Distance	List
SFG	Heur	Fwd	Entropy	List
WSFG	Heur/Wrap	Fwd	Accuracy	List
WSBG	Heur/Wrap	Bwd	Accuracy	List
LVF	Nond	Rnd	Consistency	Subset
LVW	Nond/Wrap	Rnd	Accuracy	Subset
QBB	Hybr	Bwd	Consistency	Subset
LVI	Incr	Rnd	Consistency	Subset

This question is best answered by running feature selection methods on the chosen data sets and comparing the selected or ranked features with the known relevant ones. One may guess that every method can select some features but may not be able to remove all irrelevant, redundant, or noisy features. Therefore, if the known relevant features are selected, we also want to know whether the irrelevant, redundant, or noisy features are all or partially removed. The results of experiments are shown in Tables 5.6 and 5.7. ABB cannot complete its run for Led17 after a few days. This is expected because for this particular data set, N is large (24) and M is small (7). In the tables, N.A. means a particular method is not applicable. For example, ABB is not suitable for the Iris data because it consists of continuous features.

Q2 How do these methods respond to different data types (*binary, discrete, continuous*)?

Different evaluation measures limit the methods that employ these measures to certain data types. This can clearly be observed in Tables 5.6 and 5.7. N.A. indicates that the methods cannot apply to the data. For example, most methods cannot handle the Iris data, which means that they cannot handle continuous data type.

- Indirect evaluation.

Q3 Do feature selection methods help in classification?

Table 5.6. Selected features using training data. K.R.F. means known relevant features. $\dot{d}$ means feature d is redundant.

Method (K.R.F.)	Corral (1-4)	Monk1 (1,2,5)	Monk2 (1-6)	Monk3 (2,4,5)	Parity5+5 (1-5)	Parity5+2 (1-5,$\dot{6}$,$\dot{7}$)
ABB	1 - 4	1 2 5	1 - 6	1 2 4 5	1 - 5	3 - 7
B&B	1 - 4	1 2 5	1 - 6	1 2 4 5	1 - 5	3 - 7
QBB	1 - 4	1 2 5	1 - 6	1 2 4 5	1 - 5	1 3 4 5 7
FOCUS	1 - 4	1 2 5	1 - 6	1 2 4 5	1 - 5	1 - 5
LVF	1 - 4	1 2 5	1 - 6	1 2 4 5	1 - 6	1 3 4 5 7 8
LVI	1 - 4	1 2 5	1 - 6	1 2 4 5	1 - 6	3 - 7
LVW(C4.5)	1 - 4	1 2 4	1 - 6	1 2 4	1 - 5	2 - 7
LVW(NBC)	1 - 4	1 2 4 5	3 - 6	1 2 4 5	1 2 4 5 7	1 - 5 7 8
Relief	1 - 4 6	1 2 5	2 3 4 6	2 4 5	1 - 5 7	1 - 7

Method (K.R.F.)	Led17 (1-7)	Iris (3,4)
ABB	*can't complete in days*	N.A.
B&B	1 3 4 5 9 10 12 - 23	N.A.
QBB	1 - 5	N.A.
Focus	1 - 5	N.A.
LVF	1 - 5 9 10 13 19 23	N.A.
LVI	1 - 5 7 9 14 20 23 24	N.A.
LVW(C4.5)	1 - 5 7 8 10 11 17 20 22 24	1 - 3
LVW(NBC)	1 - 5 7 8 12 14 18 - 22	N.A.
Relief	1 - 9 14 19 20 24	1 - 4

Table 5.7. Ordered features using training data. K.R.F. means known relevant features. $\dot{d}$ means feature d is redundant.

Method (K.R.F.)	Corral (1-4)	Monk1 (1,2,5)	Monk2 (1-6)	Monk3 (2,4,5)
WSFG(C4.5)	4 1 2 3 5 6	5 1 2 6 4 3	1 2 3 5 4 6	2 5 1 3 4 6
WSFG(NBC)	4 6 1 2 3 5	5 1 2 4 6 3	1 2 3 5 4 6	2 5 1 3 4 6
SFG(Entropy)	4 6 1 2 3 5	5 1 4 2 3 6	5 4 6 1 2 3	2 5 1 6 4 3
WSBG(C4.5)	2 1 3 4 6 5	1 5 4 6 2 3	3 1 2 4 5 6	5 2 4 6 1 3
WSBG(NBC)	4 3 5 1 6 2	5 1 4 2 3 6	5 3 1 2 4 6	2 5 3 4 6 1
Relief	1 2 3 4	2 5 1	3 6 2 4	2 5 4

Method (K.R.F)	Parity5+5 (1-5)	Parity5+2 (1-5,$\dot{6}$,$\dot{7}$)	Iris (3,4)
WSFG(C4.5)	1 2 10 3 9 4 5 6 8 7	1 2 10 4 5 9 3 6 7 8	3 1 2 4
WSFG(NBC)	1 2 10 3 4 5 6 7 9 8	1 2 10 3 4 8 9 6 5 7	N.A.
SFG(Entropy)	1 2 4 8 4 9 10 6 3 7	10 4 7 2 1 6 9 3 5 8	N.A.
WSBG(C4.5)	5 2 6 1 3 4 8 7 9 10	5 4 10 8 1 2 3 6 7 9	3 4 1 2
WSBG(NBC)	1 7 8 3 4 9 6 10 2 5	8 10 1 6 2 9 3 4 5 7	N.A.
Relief	3 2 4 7 5 1	3 4 5 2 7 1 6	3 4 1 2

Based on our previous study, we expect that *Feature selection should not adversely affect a classifier's accuracy and its size.*

In order to verify this conjecture, we conduct 10-fold cross validation of classifiers (C4.5 and NBC) and hypothesis testing for the classification results before and after feature selection with the null hypothesis that the two results should not be significantly different.The two performance measures are error rate and tree size for C4.5, and error rate and table size for NBC. We used the known relevant features as the selected features.

The results are summarized in Table 5.8 and 5.9. We notice that (1) if there is any difference that is statistically significant, the classifiers perform better *with* selected features; (2) even if we disregard the statistical significance, in only one case - Parity5+5, C4.5 performs slightly worse after feature selection (63% compared with 62%). Further observations about C4.5 and NBC are as follows:

- C4.5 can select features, although not always, which is evidenced in Table 5.8: for some entries (Monk2, Monk3, Parity5+5, Parity5+2, and Iris), there is absolutely no difference between before and after feature selection.
- After feature selection, the table size of NBC always decreases, sometimes, the decrease can be as large as a few hundreds (see the entry of Led17 in Table 5.9); however, the improvement in terms of predictive accuracy is little, and if there is any, it is not statistically significant. NBC is a good approximation of Bayesian classifiers for these data sets. When the average values of the two compared groups are the same, p-values are not calculated and indicated by "-" in the tables.

Q4 Are there any differences between different feature ranking methods?

Intuitively, features are ranked differently by various ranking algorithms. This point can find its support by the following: we can use various classifiers in ranking features in a wrapper model; beecause different classifiers use different biases, we can reasonably expect that the orders of features ranked by different classifiers are not the same. As is shown in Table 5.7, orders of ranked features are indeed not the same.

Q5 How general is a ranked list of features?

In the same spirit of the above thinking, we can turn the question (Q4) into a more specific one - "Can a list of features ranked by one classifier be used by another classifier to improve the second classifier's performance?", assuming a wrapper model. If the answer is yes, we also answer question Q5.

Table 5.8. Average error rates (E-Rate) and average tree size (Tr-Size) of C4.5 Before and After feature selection and their p-values. The known relevant features are used as selected ones.

Data Set	E-Rate (Before)	E-Rate (After)	p-Value	Tr-Size (Before)	Tr-Size (After)	p-Value
Corral	6.00	0.0	0.0	14.6	0.0	0.0
Monk1	1.4	0.0	0.0	43.0	41.0	0.05
Monk2	34.6	34.6	-	14.0	14.0	-
Monk3	1.1	1.1	-	19.0	19.0	-
Parity5+5	0.08	0.00	0.05	62.6	63.0	0.4
Parity5+2	0.0	0.0	-	63.0	63.0	-
Led17	0.00	0.00	-	19.0	19.0	-
Iris	5.40	5.40	-	9.0	9.4	0.7

Table 5.9. Average error rates (E-Rate) and average table size (Ta-Size) of NBC Before and After feature selection and their p-values. The known relevant features are used as selected ones.

Data Set	E-Rate (Before)	E-Rate (After)	p-Value	Ta-Size (Before)	Ta-Size (After)	p-Value
Corral	15.00	11.66	0.535	26	18	-
Monk1	34.18	31.99	0.876	36	22	-
Monk2	38.50	38.50	-	36	36	-
Monk3	3.63	3.63	-	36	22	-
Parity5+5	61.86	56.76	0.91	42	22	-
Parity5+2	62.15	56.76	0.793	42	22	-
Led17	0.00	0.00	-	490	110	-
Iris	N.A.	N.A.	N.A.	N.A.	N.A.	N.A.

Figure 5.3 shows that features ordered by one classifier may not be suitable for another. The data is Parity5+5 which is split into the training and test data sets. Two classifiers are NBC and C4.5. In Figure 5.3(a), two curves of the error rates of C4.5 on the test data are plotted. One is about using a list ranked by C4.5, and the other is about using a list ranked by NBC. We can see that for the list ranked by NBC, C4.5 cannot reach the lowest error rate achieved on the list ranked by C4.5. In Figure 5.3(b), the error rates of NBC are plotted. The situation is reversed: for the list ranked by C4.5, NBC cannot reach the lowest error rate achieved on the list ranked by NBC. Therefore, we need to keep in mind this observation from Figure 5.3 and to be aware that the possible shortcoming of using one classifier in feature ranking and another classifier for classification.

WSFG(C4.5) and WSFG(NBC) are used in this experiment. Since the orders of the two feature ranking algorithms are different, we only specify the number of features used in Figure 5.3. The exact orders of the features can be found in Table 5.7 for WSFG(C4.5) and WSFG(NBC).

Q6 How does a classifier fare with a ranked list of features?

For a properly ranked list of features, we should expect to observe a convex learning curve with its peak somewhere in between such that accuracy improves as more features are used until it reaches the peak, then it drops when more features are added. In Figure 5.4, we plot error rates of the test data with increasingly enlarged feature subsets. The training data is used by C4.5 to rank the features. We can see the concave shapes for error rates (the complementary shapes for accuracy). A curve with such a shape indicates that the ordered features are at least good for the classifier concerned. It is not true that we can always see these curves because we usually employ heuristic methods[5] for feature ranking. Note that the results in Tables 5.8 and 5.9 are obtained through 10-fold cross validation which is different from just using the test data to estimate error rates.

Q7 How can feature selection help a classifier learn faster?

As it was claimed in Chapter 1 (Section 1.3) that intuitively, feature selection helps because we have the same amount of data but a reduced number of features. Other things being equal, with selected features, a classification algorithm should require fewer instances to learn a classifier than with original features. We divide the Parity5+5 data into small chunks of 32 instances each. Two learning curves are drawn for using selected features or all features in Figure 5.5 as we use 1, 2, ..., and 32 chunks to learn. 10-fold cross validation is used to estimate error rates

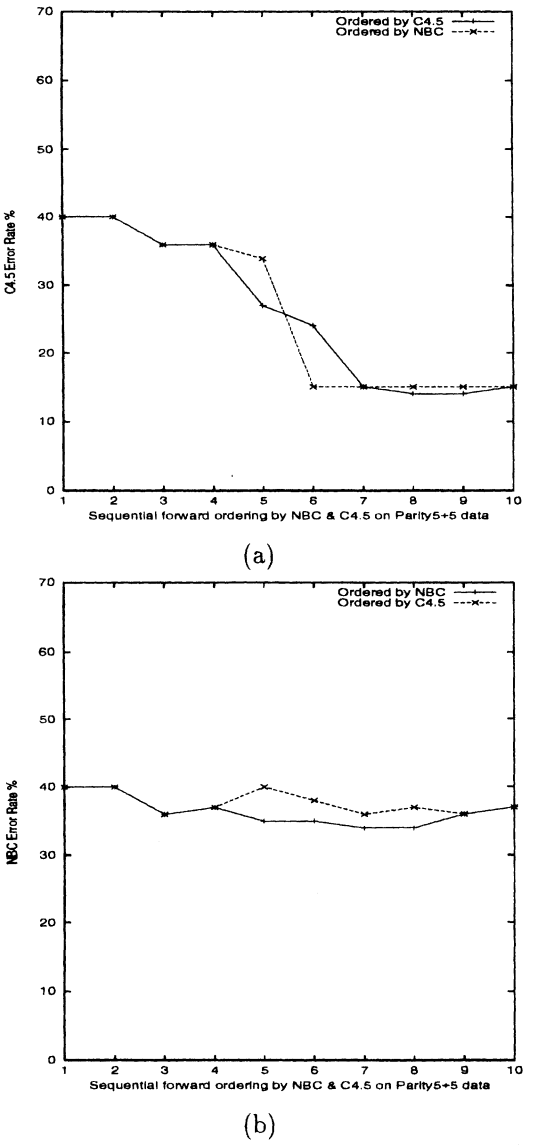
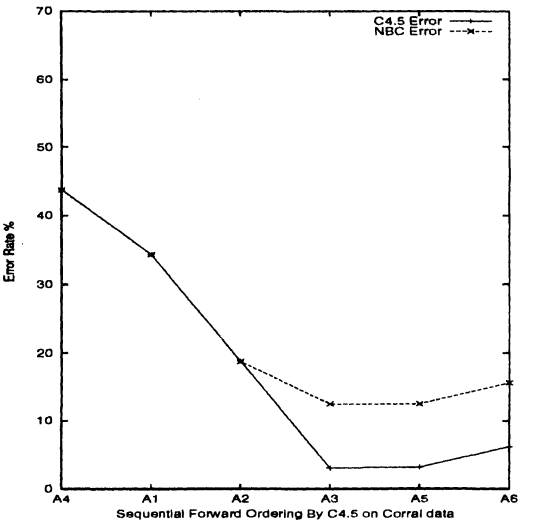
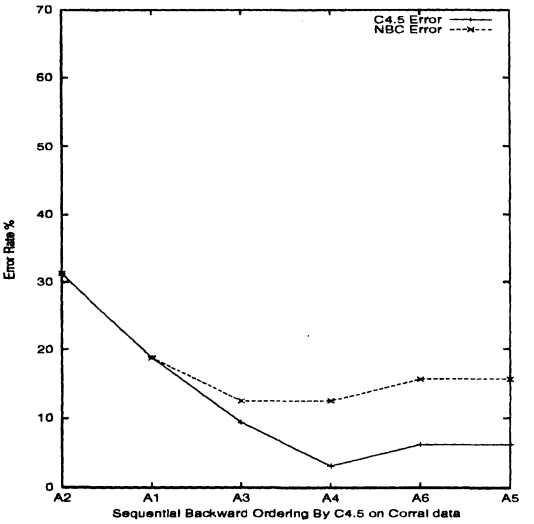


Figure 5.3. (a) Error rates of C4.5 on the Parity5+5 data using the features ordered by NBC and C4.5. (b) Error rates of NBC on the Parity5+5 data using the features ordered by NBC and C4.5.



(a)



(b)

Figure 5.4. (a) Features of the Corral data are ordered by WSFG using C4.5; (b) Features are ordered by WSBG using C4.5. Test results (two curves) are obtained by C4.5 and NBC respectively.

for data of various sizes. Using selected features, C4.5 reaches its zero error rate when six chunks are used; using all features, C4.5's error rate approaches to zero when 28 chunks are used.

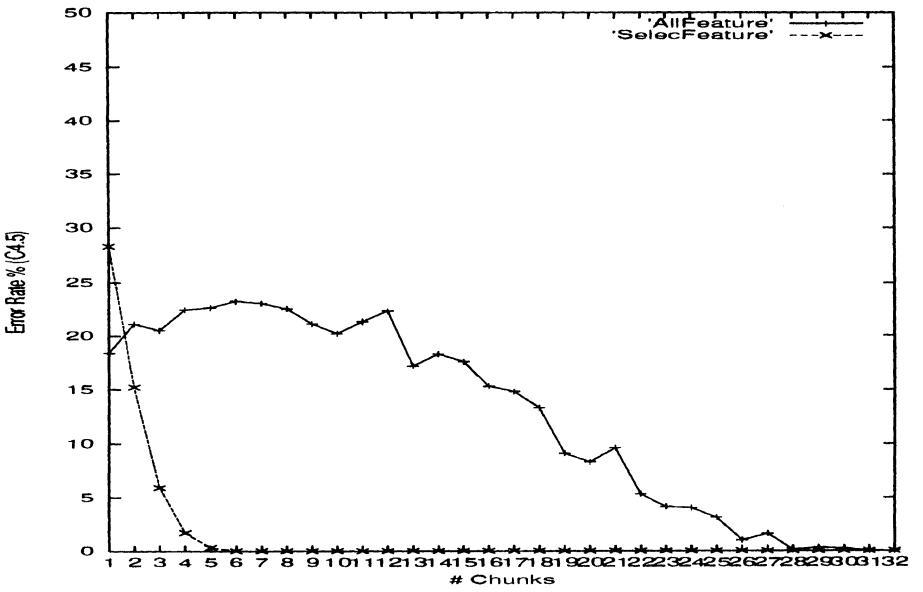


Figure 5.5. The effect of feature selection on the data size required for the learning to converge. The learning curves with and without feature selection show average error rates of 10-fold cross validation for C4.5 on the Parity5+5 data with various chunks of data.

Q8 What is the time performance of a filter selector versus a wrapper one?

As we discussed earlier, a filter feature selector should be faster than a wrapper one because the filter selector uses a computationally much cheaper evaluation measure than relying on a classifier to estimate the accuracy using selected features. In Table 5.10, we give the results of LVF (a filter selector) and LVW (a wrapper selector). The number of runs for each algorithm is the same and set as 25% of 2^N. When the size P of data sets is small, the difference of run times between LVF and LVW is still tolerable. When both P and N are large as in the Led17 data, LVW takes days to finish. We should be aware of this point in practice and strike a balance between a tolerable computational time

Table 5.10. A run time comparison between LVF and LVW.

Data Set	Time By LVF (Sec)	Time By LVW (Sec)
Corral	0.328	16.535
Monk1	0.334	17.385
Monk2	0.283	16.921
Monk3	0.271	16.836
Parity5+5	0.252	241.325
Parity5+2	0.386	201.812
Led17	36.705	days

and a proper learning bias, as we know that LVW employs C4.5 for evaluation and C4.5 is a fast learning algorithm.

Q9 Do we need different evaluation measures?

We mentioned in Section 2.3.7 of Chapter 2 that different measures have different effects on removing irrelevant or redundant features. Many conjectures can be verified here through experiments. As our data sets include various correlated-noisy, irrelevant or redundant features, we can judge how effective these evaluation measures are in removing them. To check if a measure is good in removing correlated-noisy features, we can look at the result of the Corral data; to check if a measure is effective in removing redundant features, we can examine the result of the Parity5+2 data; to check if a measure is useful in removing irrelevant features, we can review the results of the Monk1 and Led17 data; to verify if a measure does not cause the removal of any relevant feature, we can inquire into the Monk2 data; and to determine if a measure is capable of removing noisy features, we can inspect the result of the Monk3 data.

Tables 5.6 and 5.7 have provided the results of various feature selection or ordering methods (SFG(Entropy), WSFG(C4.5), Relief, WSBG(NBC), and LVF) that employ different evaluation measures (entropy, information gain, distance, accuracy, and consistency). By comparing their results with the known relevant features, the reader can check how each measure works. One can see that for the Monk3 data, if the default setting is used for all feature selection methods, only Relief works well (keeping only three relevant features). In our experiments, we noticed that for those methods that employ the consistency measure, if we specify the allowed inconsistency rate as 5%, they can all reduce six features to three features (#2, #4, #5). Without the specification of 5%, they

Table 5.11. Error rates of three classifiers on Corral and Parity5+5 data before and after feature selection. Features are selected by Focus.

Classifier	Corral (Before)	Corral (After)	Parity5+5 (Before)	Parity5+5 (After)
NBC	9.37	31.25	58.0	59.0
C4.5	12.5	12.5	47.0	22.0
SNNS	2.77	2.77	47.56	25.4

cannot remove the irrelevant feature (#1) because the training data has 5% instances whose class labels are flipped to introduce noise.

Q10 Do different classifiers respond similarly to feature selection?

We know that different classifiers have their unique ways of using features. For all the wrapper selectors, different lists of ordered features are obtained as shown in Table 5.7.

We want to further examine the effects of feature selection on different classification algorithms: NBC, C4.5, and SNNS. We use the training and test data sets of Corral and Parity5+5 to have a preliminary experiment. Features are selected by Focus. The results of C4.5, NBC, and SNNS are presented in Table 5.11. The reader may notice that the results of Table 5.11 for C4.5 and NBC are not synchronized with those in Tables 5.8 and 5.9. The reasons are due to their different error estimation methods. Here we do not perform 10-fold cross validation and simply use the training data to train and the test data to test. As was pointed out earlier, the variances caused by this estimation method are expected to be larger. SNNS is stopped when no improvement of predictive accuracy is observed for the test data. Despite the limitation of the experiment, we observe that for C4.5 and SNNS, feature selection helps, but for NBC, it is not true, which is consistent with our understanding that there are no bad features for a Bayesian classifier.

Q11 Do search directions matter?

Intuitively, they matter. That is, for example, sequential forward generation (WSFG) and sequential backward generation (WSBG) do not generate the same sequence of ranked features. Exhibited in Figure 5.6 are the two curves of error rates (C4.5) on the Iris data using features ordered by C4.5. The features are ordered using the training data and error rates are obtained from the test data. It is clearly shown that search directions do matter.

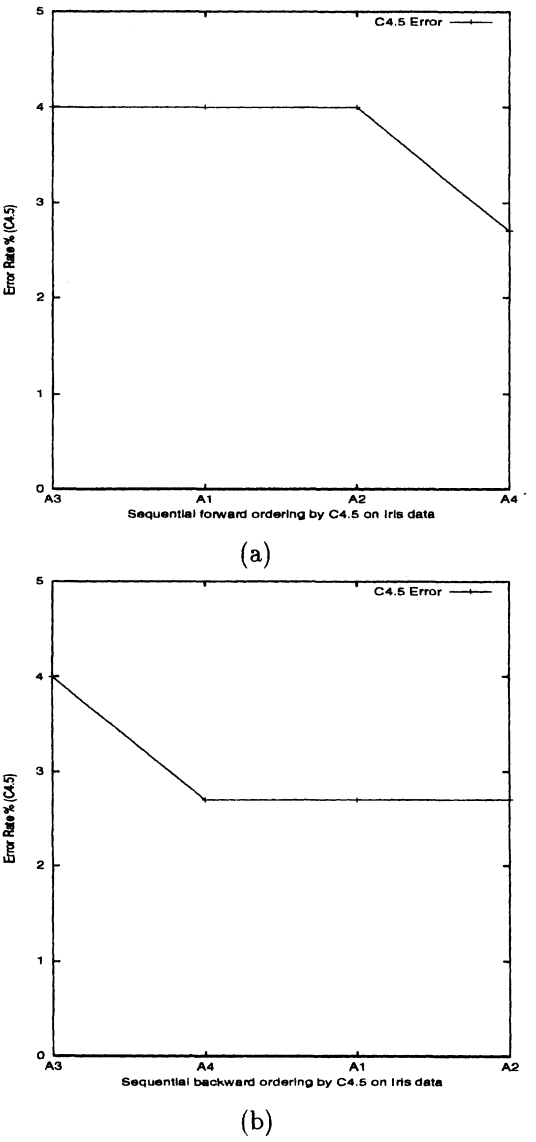


Figure 5.6. (a) Error rates of C4.5 on the Iris data using features ordered (sequential forward) by C4.5; (b) Error rates of C4.5 on Iris using features ordered (sequential backward) by C4.5.

Summary. Many more questions can be asked and many more experiments can be conducted. The above questions and experiments just serve as examples of what we can do in evaluation of feature selection methods. The purpose of this section is not to conduct extensive comparison or evaluation, but to provide the reader with some basics to get ready for such challenges if necessary.

5.4.3 Issue of scalability

When we treat feature selection as a search problem, we focus on the number of features (N). The search time increases exponentially with the increase of features, assuming exhaustive search regardless of search direction. This is, however, only one dimension of any data sets represented in feature-values. The other dimension is the number of instances (P), although P is eventually determined by N and the number of values each feature can have. The size of a data set can be practically measured by $N \times P$. In reality, both N and P can be large or small. If N is large but P is small, a data set of such nature indicates a sign of sample shortage. Usually too many inductive hypotheses can be derived from the data. In the context of data mining, the problem we are facing commonly is that both N and P are large. We handle the largeness of N by devising heuristic or probabilistic search methods. Overcoming the largeness of P has to take other directions since an evaluation measure can only be accurate when all the available data is used. For example, we cannot have a correct count of inconsistencies unless all the instances are checked. Our experience with the inconsistency checking shows clearly the gain in speed from using a sorting algorithm ($O(P \log_2 P)$) to employing a hashing algorithm (roughly $O(P)$). Any improvement from $O(P)$ for inconsistency checking would be very difficult to achieve. As P is increasingly large in some applications, practitioners are seriously concerned with the issue of scalability. LVI (introduced in Chapter 4) is an example showing that we can still improve in speed if P is *large*. Since some overhead is involved (see the entry for LVI in Table 4.9), the speed gain will only be evident when P is sufficiently large. The current state is that the issue of scalability is far from being solved. It is occurring everywhere ranging from inductive learning, clustering, and feature selection to feature discretization.

In (Liu and Setiono, 1998), three data sets with different sizes (small, medium, and large) were chosen to verify the overhead issue. The data sets are: (i) Vote - 16 features with 435 instances; (ii) Mushroom - 22 features with 8125 instances; and (iii) Parity Mix - 20 features with 10,000 instances. The results were obtained by randomly partitioning the data and running LVI 10 times using 10%, 20%, ..., and 90% of the data, and by running LVF 10 times using 100% of the data; p-values were calculated for each result of LVI and the re-

sult of LVF. Averages values of 10 times (means) were indicated in Figures 5.7 and 5.8. The basic findings are: (1) the effectiveness of LVI becomes more obvious when the data size (P) is larger. The gain in time is not obvious in data Vote, but apparent in data Parity Mix as shown in Figure 5.7; and (2) LVI does not sacrifice the quality of feature selection (i.e., number of features). As seen in Figure 5.8, there is no significant difference between means of numbers of features.

5.4.4 Remarks

Although we employ three types of classification algorithms, the experiments conducted here serve no purpose in evaluating different types of classifiers. We just try to show the effects of feature selection. Thorough and extensive comparisons, for example, between symbolic and connectionist classifiers can be found in (Fisher and McKusick, 1989, Dietterich et al., 1990, Shavlik et al., 1991, Quinlan, 1994).

5.5 BALANCE BETWEEN DIFFERENT PERFORMANCE CRITERIA

This is a ubiquitous problem in feature selection, feature discretization, clustering, as well as in rule induction. The point is that relying on any single criterion is simple minded. More often than not, more than one criterion is needed, and the issue is how to weigh these criteria and combine them in a utility function so that by optimizing the function, we can hopefully achieve a balanced performance. This is quite common in research fields such as operations research (Hillier and Lieberman, 1990) and genetic algorithms (Goldberg, 1989).

We use rule induction as an example to elaborate on the need for a balanced measure. We believe that practitioners in data mining are familiar with production rules induced from learning systems like CN2 (Clark and Niblett, 1989) or C4.5 (Quinlan, 1993). When two induction algorithms are applied to the same classification problem, they will most likely extract different sets of rules. Naturally, people are interested in finding certain metrics that can be used to compare the *goodness of extracted rules*. Intuitively, a good rule set, in addition to high predictive accuracy, should contain rules as compact as possible whose semantics could be easily understood. It is more easily said than done. One of the reasons could be the goodness of a rule set sometimes is rather subjective and/or application dependent. (Yen and Chen, 1995) propose to measure the compactness of a rule set in three dimensions: (1) the number of rules, (2) the number of instances covered by a rule, and (3) the number of conditions of a rule. The reasoning is that a more compact rule set is easier to understand.

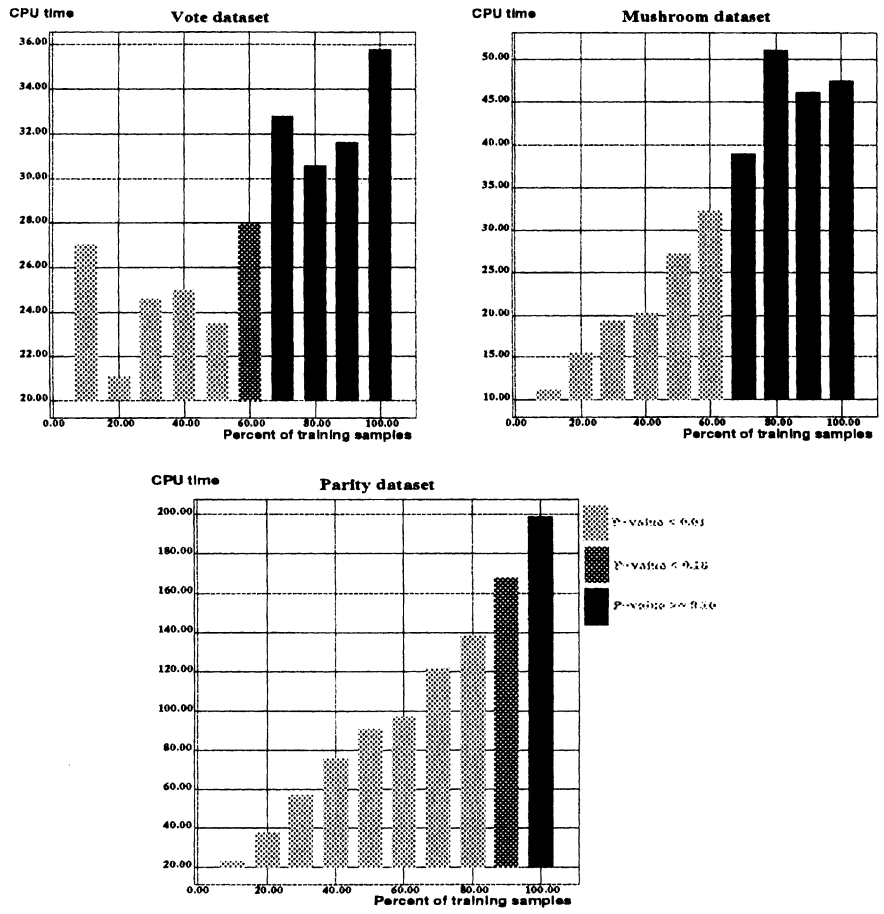


Figure 5.7. Average CPU time (seconds). The result for 100% data is used as the reference. The difference between any two samples (e.g., 10% vs. 100%, or 20% vs. 100%) is the most significant (light grey), significant (dark grey), or insignificant (black).

The comprehensive measure is therefore:

$$E = \frac{1}{r}\sum_{i=1}^{n}(\frac{t_i}{n} \times \frac{1}{C_i}) \tag{5.24}$$

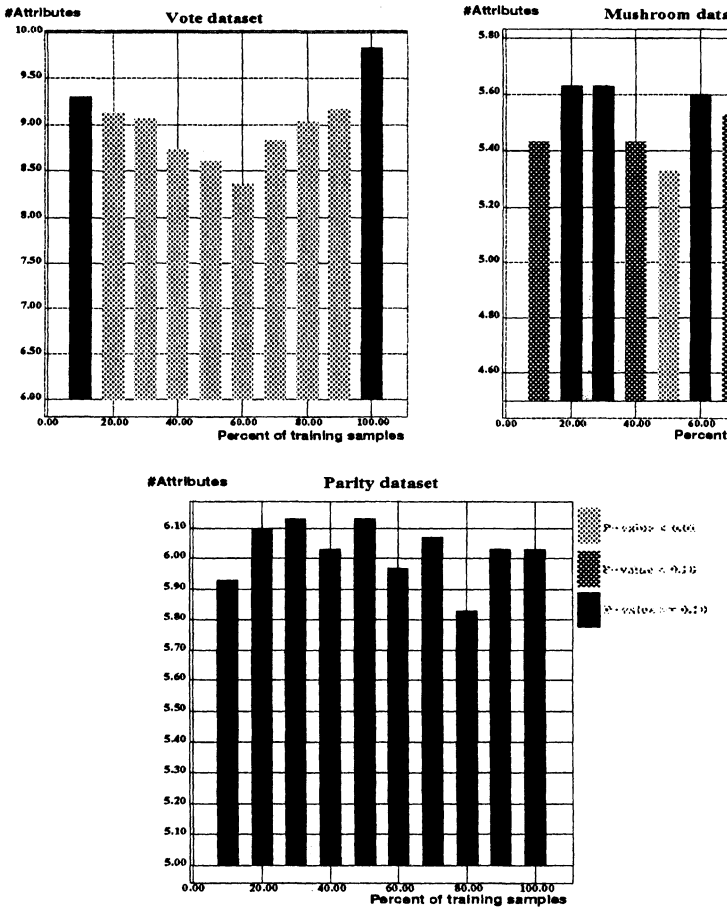


Figure 5.8. Average number of selected features. The result for 100% data is used as the reference. The difference between any two samples (e.g., 10% vs. 100%, or 20% vs. 100%) is most significant (light grey), significant (dark grey), or insignificant (black).

where n is the total number of instances in the test set, r is the cardinality of the rule set, t_i is the number of instances classified by rule i in the rule set, and C_i is the number of conditions in the antecedent of rule i. The rationale for this measure is that, the smaller the cardinality of the rule set is, the more compact is the rule set; the more instances covered by each rule in the rule set, the fewer rules are needed, so the more compact the rule set; and the smaller the number of condition features in the antecedent of a rule in the rule set is, the simpler each rule is, and the more compact is the rule set. While Equation 5.24 provides a simple metric to evaluate the compactness of a rule set, it cannot be applied to the situation where there are overlapping instances covered by different rules, since in such a case, Σt_i is usually greater than n.

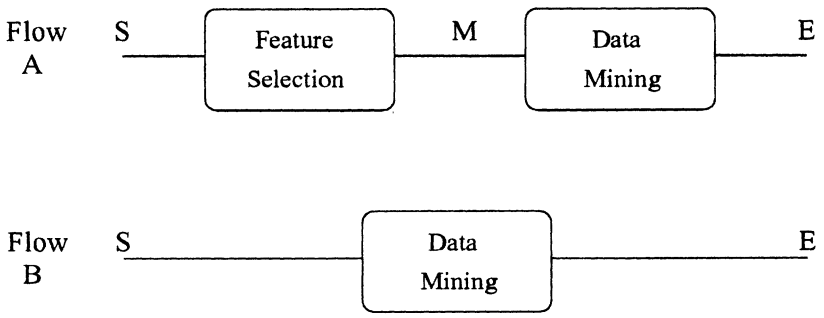


Figure 5.9. Comparing end results: with and without feature selection.

With the appreciation of a comprehensive measure for inductive rules, we come back to the issue of measuring feature selection. Feature selection is mainly treated as a preprocessing phase. It is seldom performed for selecting features *per se*, and usually involves data mining or inductive learning in the next phase. On that account, a comprehensive measure of feature selection should at least include the processes of feature selection, data mining, and the results of applying feature selection to data mining. Basically, we need to know what we gain and/or lose after we add the process of feature selection. In Figure 5.9, two flows are shown: Flow A consists of two processes (feature selection and data mining), Flow B contains one process (data mining). The overall comparison involves the following measures:

- **Computing time** - Feature selection is an added process that consumes computing time to churn out meaningful results (reduced number of features). Simpler data due to feature selection can hopefully lead to the time reduction for data mining. In most cases, we cannot afford to spend too

much time on feature selection, although the more time we spend, the better the outcome. But too little time on feature selection could be detrimental to data mining. Let t_B be the time spent on Flow B, t_A the time on data mining in Flow A, and t_{FS} the time on feature selection. It is expected that $t_B \leq t_{FS} + t_A$, but it is also hoped that $t_A \leq t_B$ since the dimensionality of the data is reduced. This has been observed in Figure 5.5 that with feature selection, C4.5 quickly converged. The increased time for Flow A is expected to bring us improvements on other accounts.

- **Predictive accuracy** - This is the dominant measure for inductive learning since it measures how well the data is summarized and generalized into abstract forms of representation such as rules. If the predictive accuracy of the rules is way off the mark (for example, worse than flipping a coin or choosing the most frequently occurred class), it means that the time spent on inductive learning is wasted. We generally expect that by only using relevant features, a learning algorithm can not only learn faster, but also better (i.e., higher accuracy). The intuition is that irrelevant features may mislead the learning algorithm, and redundant features may complicate the task of learning. It is clear if accuracy improves with feature selection. What is really difficult is whether we should call it off if much time is spent on feature selection but without improvement in accuracy. We need to consider the third aspect of feature selection, checking if feature selection can bring us simpler learning results.

- **Representation of end results** - The representation can take many forms. They could be rules, decision trees, decision lists, networks, and many others. Sticking to one form, we expect the representation to become simpler after feature selection. The simplicity of a representation often implies its better comprehensibility. Following the line of reasoning of Occam's razor - the most likely hypothesis is the simplest one that fits the data in inductive learning (Russell and Norvig, 1995), simpler representations should also imply better accuracy. Some people interpret Occam's razor as meaning "the world is inherently simple." Another argument is that there are far fewer simple hypotheses than complex ones, so that there is only a small chance that any simple hypothesis that is wildly incorrect will be consistent with all instances. (Mitch, 1997) offers some interesting arguments about Occam's razor. As was observed by (Murphy and Pazzani, 1994), however, in a case of decision trees, smallest trees do not necessarily give the best predictive accuracy. This observation indicates the need for a balanced view about accuracy and simplicity. If the simplicity of representation improves, the decrease of accuracy may be tolerable, or vice versa.

It is ideal if we can achieve them all, i.e., reduced time, improved accuracy, and simplified results. Often is the case that we gain in some and lose in others. Therefore, we need to know if the gain is worthwhile with the price we need to pay for the loss. In other words, we have to balance the gains and losses according to the application at hand. One way is to prioritize different measures. We categorize the measures in terms of the stages:

1. *After feature selection*, we are concerned about the reduction of *features* with respect to the *time* spent on feature selection.

   (a) *Reduction of features*
   It requires two numbers to measure the reduction: N - the number of original features and M - the number of selected features by a feature selection algorithm. The larger the difference, the more reduction of features.

   (b) *Time*
   t_{FS} is the time spent on feature selection. The gain in reduction costs computing time. The loss in time should be proportional to the gain in reduction of features. If the return of spending much computing time turns out immaterial, we need to reconsider the feature selection method used because of its poor efficiency. Another way of examining if t_{FS} is too much is to check it against the time spent on data mining (to be discussed next).

2. *After data mining*, we have another set of concerns. They are mainly about the performance of data mining or induction before and after feature selection (predictive accuracy and simplicity of results), and about the time saving or increasing due to feature selection for the whole process of data mining. Each item below requires two measures, i.e., before and after feature selection, see Flow A and Flow B in Figure 5.9. End result (E) is measured by predictive accuracy and E's simplicity.

   (a) *Accuracy*
   The difference between Acc_A and Acc_B should be expected to be positive (or $Acc_A - Acc_B \geq 0$).

   (b) *Simplicity*
   We expect that after feature selection, the simplicity of data mining results improves. The simplicity of induced results depends on the form of the results. If-Then rules can be measured by the number of rules and the average number of conditions in antecedents of rules. A network can be measured by the number of layers, the number of hidden units, as well as the number of connections. A decision tree can be measured by

the number of nodes. It is not recommended to compare the simplicity across different forms, for example, comparing the simplicity of rules with that of networks, although different forms of induced results may exhibit various degrees of comprehensibility.

(c) *Time*
t_A and t_B are the times spent on the two flows (A and B), respectively. It is reasonable to expect that time t_A is greater than time t_B. However, t_A should be comparable with t_B. If the difference of the two is too large, it indicates that feature selection consumes much more time than data mining or induction itself.

Depending on a task at hand which may or may not involve data mining, we choose appropriate measures for evaluation.

5.6 APPLYING FEATURE SELECTION METHODS

It is well known that no single method can be best suited for all applications. Some methods work better than others for some applications. With so many feature selection methods available, it is not an easy task to decide on which method should be used for an application. This chapter is devoted to this issue. The basic idea is to categorize applications according to some measurable criteria, and establish mappings between categories and groups of methods. Table 4.1 in Chapter 4 offers a suggestion about the groups of feature selection methods. We focus on categories of applications here. We can categorize applications in many ways (or based on different criteria). Two general criteria are about (1) *knowledge* - how much we know about an application; and (2) *speed* - how fast we need a solution. The best performance (e.g., best accuracy, simplest rules) is always a goal while we consider the knowledge and speed criteria. Another important factor is of data characteristics in making a choice of proper feature selection methods. Now we proceed to discuss the knowledge and speed criteria plus the data characteristics, followed by the guidelines of applying feature selection methods.

5.6.1 Prior knowledge

The study of types of knowledge regarding features can help us in applying knowledge to feature selection. We discuss different types of knowledge related to feature selection and how knowledge can help in removing irrelevant and/or redundant features, and noise in the data.

- **Relevance knowledge**. It indicates that a group of features is relevant to a particular goal class. Strong relevant knowledge alone can eliminate

irrelevant and/or redundant features. Weak relevant knowledge can help guide the search for optimal features.

- **Noise knowledge**. This type of knowledge provides information about types of noise and level of noise in the data. It is extremely useful in handling noise. An example is of the Monk3 data in which removing irrelevant feature can increase the number of inconsistency, instead of decreasing it. By knowing that there is 5% of noise, all irrelevant features can be removed.
- **Correlation knowledge**. Knowing that some features are highly correlated to some others can help removing redundant features. Another use of correlation knowledge is if we know that a random feature is correlated to the class, it may look like relevant, but actually irrelevant (as in the Corral data, one random feature is 75% correlated to the class label).

All types of knowledge are useful in feature selection. The basic principle of using knowledge in feature selection is *applying knowledge first*. Put it another way, knowledge-driven feature selection should be tried before data-driven feature selection[6], particularly when the knowledge is strong. Another use of prior knowledge is the double checking of the results by data-driven knowledge, as we did in selecting for experiments those data sets whose relevant features are known.

5.6.2 Processing speed

Broadly speaking, complete search algorithms are more time consuming than heuristic or nondeterministic search algorithms. Viewing from a perspective of how solutions are churned out by a feature selection method, we can divide all feature selection algorithms into two categories: classic and anytime. We briefly discuss the two below.

Classic algorithms. This type of algorithms produce results only *once* during the process of feature selection. The user of such algorithms usually has to wait until the end of the process in order to obtain any results. Most exhaustive search algorithms are of this type. One shortcoming of classic algorithms is that the user could be kept waiting in vain for a very long time before the result is pumped out. The other type of algorithms produce improved results continuously. They are called anytime algorithms.

Anytime algorithms. The term *anytime algorithms* was coined in the context of time-dependent planning (Boddy and Dean, 1994) in their early work. Anytime computation extends the traditional notion of a computational procedure by allowing it to return many possible approximate answers to any given

input. What is special about anytime algorithms is the use of well-defined quality measures to monitor the progress in problem solving and allocate computational resources effectively. Desired properties of anytime algorithms listed in (Zilberstein, 1996) are

1. Measurable quality - the quality of an approximate result can be determined precisely,

2. Recognizable quality - the quality of an approximate result can easily be determined at run time,

3. Monotonicity - the quality of the result is a nondecreasing function of time and input quality,

4. Consistency - the quality of the result is correlated with computation time and input quality,

5. Diminishing returns - the improvement in solution quality is larger at the early stages of the computation, and it diminishes over time,

6. Interuptibility - the algorithm can be stopped at any time and provide some answer, and

7. Preemptability - the algorithm can be suspended and resumed with minimal overhead.

Using the abovementioned properties, we can tell anytime algorithms from those that are not. Examples of existing anytime algorithms, as identified by (Boddy and Dean, 1994), are: numerical approximation, heuristic search, dynamic programming, probabilistic algorithms, probabilistic inference, and discrete or symbolic processing.

The *speed* criterion can help in choosing between classic algorithms or anytime algorithms. Anytime algorithms can produce quick and dirty intermediate solutions and improve them over time, though they may require a similar amount of time as that required by classic algorithm in order to find optimal solutions. The distinction between the two types is an important factor in constructing a contingency theory of applying various feature selection methods.

5.6.3 Data characteristics

The application of feature selection methods is strongly influenced by data characteristics: type, quality, and size.

Data type. Data consists of feature-values. Features can be continuous, or discrete. If a feature selection method is more suited for continuous data, it may be inappropriate for discrete data; and vice versa. When data is mixed with both continuous and discrete data, special consideration should be taken. The class feature can have binary or multiple values. Sometimes, extra efforts are needed when a feature selection method, which is designed for binary class values, is applied to data with multiple class values.

Data quality. The quality of data is closely linked to noise, missing values, and other faults. Noise here includes wrongly input values for features and class, random features, and any unexpected perturbation of data caused by human mistakes, storage error, or data recording malfunction. An experienced domain expert can roughly estimate the data quality. Not all feature selection methods can handle data with noise, or missing values, some methods can handle noisy data with the aid of domain experts. In cases of missing values, we may need to preprocess them before conducting feature selection. Missing values have different meanings in different application domains. A missing value could mean that a value is missing, or the value is not taken deliberately (as in a medical report, some tests were not done because there is no such need).

Data size. As we mentioned earlier, the data size can be estimated by two variables: (1) the number of features (N) and (2) the number of instances (P). Another useful variable is the number of relevant features (M). Under normal circumstances, N is the deciding variable. If it is large (> 20), it suggests that the feature selection problem is usually hard. The value of M can also help us select a proper method for feature selection. If N is large but M is small (we know the value of M but do not know which M features) or if the difference between N and M is small, a complete search algorithm can still be considered since it does not consume much time as it appears to.

5.6.4 How to choose a feature selection method?

The above discussion on prior knowledge, processing speed and data characteristics has prepared us to develop a procedure that can guide us in various situations to choose a method for feature selection. The prior knowledge about a task can be rich or lean. The amount of knowledge is our first consideration in deciding on a choice of a feature selection method. Another important factor is the amount of time we are allowed to perform feature selection. Generally, the more the time spent, the better the quality of selected features. It is only natural that we make full use of the amount of time allowed in feature selection. Figure 5.10 shows a flow chart for decision making with two considerations:

knowledge and time. The flow chart offers a general guideline as a first step in finding a proper method. When knowledge is not sufficient or strong for us to make a decision, we have to rely on using data to the extent subject to the amount of time available. If time is plenty, we tend to choose search-intensive methods (complete search, for example); if time is scarce, we usually employ heuristic methods to avoid intensive search, anytime algorithms are also a good choice since they are quick in producing some solutions; otherwise, we consider nondeterministic search methods. In the flow chart, we assume our preference of knowledge over time. In reality, we may be forced to give the two factors (knowledge and time) equal treatment. Also we should bear in mind that the guideline is general and serves as an example to show how these two factors influence decision making. The choice of a method can be rather complicated in practice and the boundaries between different methods may not be as clear-cut as they are shown in Figure 5.10. However, having information about the previous chapters and being equipped with guidelines suggested here, the reader should be able to customize a most suitable feature selection method for a task at hand.

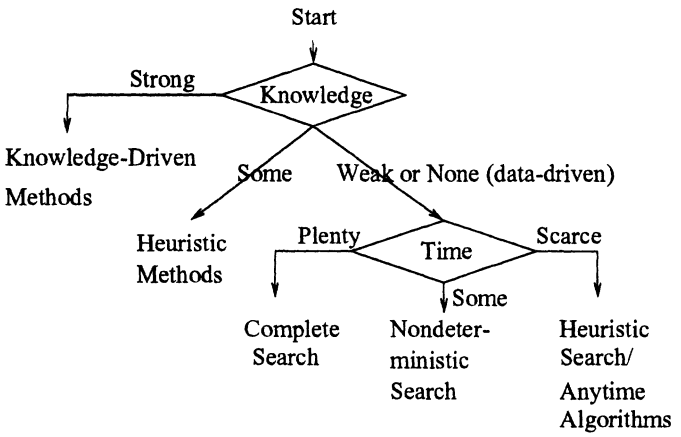


Figure 5.10. Choosing a proper feature selection method based on knowledge and time available.

Continued from Figure 5.10, we group the discussed methods in terms of their search strategies and check them out against data characteristics such as noise, continuous or discrete features, multiple class values, redundant features, and randomly correlated features. The results are obtained based on the empirical study done earlier and summarized in Table 5.12.

Table 5.12. Feature selection methods and their capabilities of handling data of different characteristics. Comp - Complete, Heur - Heuristic, Nond - Nondeterministic; Cont/Disc - Continuous or Discrete, Rednt - Redundant, RndCorr - Randomly Correlated Noise; Dpnd means it depends.

Search	Method	Noise	Cont/Disc	MultiClass	Rdnt	RndCorr
Comp	Focus[a]	✗	Disc	✗	✓	✓
	ABB	✓	Disc	✓	✓	✓
	B&B	Dpnd	Disc	✓	✓	✓
Heur	Relief[a]	✓	Both	✗	✗	✗
	SFG	✗	Disc	✓	✓	✗
	WSFG	✓	Dpnd	✓	✓	✗
	WSBG	✓	Dpnd	✓	✓	✗
Nond	LVF	✓	Disc	✓	✓	✓
	LVW	✓	Dpnd	✓	✓	✗
Hybr	QBB	✓	Disc	✓	✓	✓
Incr	LVI	✓	Disc	✓	✓	✓

[a] Focus and Relief are reimplemented here and can handle noise and multi-classes. Reimplemented Relief follows Relief-F (Kononenko, 1994).

Another important factor in choosing a method is the knowledge of M (the number of relevant features) in regard to N (the number of all features). For example, we can design a procedure with three methods (Focus, ABB, and QBB) as follows:

If M is known as SMALL
  Apply **Focus**
Else If $N \leq 9$
  Apply **ABB**
Else
  Apply **QBB**

If M is small, regardless of N, we can afford to apply Focus. Of course, SMALL is a floating concept and its meaning is changeable under different circumstances depending on time available for the task. Without information about M, we often end up making a choice between **ABB** and **QBB**[7]. (Pudil and Novovicova, 1998) studied a method that uses the knowledge about problems to guide feature selection.

We often need to take into account the purpose of applying feature selection in method selection. To recap, the major purposes are (1) improving predictive accuracy, (2) simplifying the learned results, and (3) achieving dimensionality reduction. When we are only concerned about dimensionality reduction, we pay more attention to how general the selected features are (in other words, they are not idiosyncratic to a particular classification algorithm), so a filter feature selector is more suitable since it is independent of any classifier; but if we aim to improve predictive accuracy while simplifying the learned results, a wrapper feature selector would be more appropriate.

Application tasks often have their peculiar requirements. The simplest ones for feature selection tasks are of the output type of a feature selection method. It is a ranked list or a minimum subset of features. In some sense, a ranked list is more flexible than a minimum subset because if we want to have a subset of m features, we can simply select the first m features in a ranked list. In addition, we can also add more features according to their relative relevance in the ranked list if such a need arises. Notwithstanding its flexibility, a ranked list is usually obtained by a stepwise method and may not truly represent the features' relevance. In any case, the ranking of relevance is only linear and cannot reflect any higher order relevance such as hierarchical relevance. Table 5.13 categorizes the mentioned feature selection methods in the light of the output type. It is clear that both filter and wrapper models can produce minimum subsets and ranked lists. Put it another way, choosing a model does not determine a feature selection method's output type. As seen in Table 5.13, the incremental method LVI can be of either filter or wrapper depending on which model is used inside LVI. For the hybrid method QBB, two filter selectors (LVF and ABB) give rise to another filter selector. B&B is considered as a wrapper model here since we go with the tradition and use an error rate as its evaluation measure. Certainly, if the user wishes to use other measures, then B&B becomes a filter model.

As we see now, so many factors should be considered in determining a feature selection method. To facilitate the understanding of the relations between the factors and with search strategies and evaluation measures, an influence map is drawn in Figure 5.11. "M/N info" denotes the knowledge of M and N which can help in determining a search strategy or a feature selection method. We also see that search strategies and evaluation measures further limit our choices of feature selection methods. Data characteristics affect evaluation measures because some measures may only be suitable for particular kinds of data.

Summarizing all the five determining factors into one picture (Figure 5.12), we demonstrate how the discussed feature selection methods are categorized in terms of these factors. The purpose of feature selection determines the choice of a filter or wrapper model. The relations between the methods and data char-

Table 5.13. Output types of some feature selection methods and their models (filter or wrapper).

Output Type	Method	Model
Minimum Subset	Focus	Filter
	ABB	Filter
	LVF	Filter
	LVW	Wrapper
	LVI	Both
	QBB	Filter
	B&B	Wrapper
Ranked List	Relief	Filter
	SFG	Filter
	WSFG	Wrapper
	WSBG	Wrapper

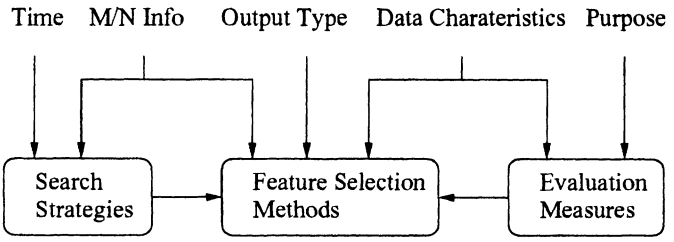


Figure 5.11. An influence map shows how many factors have influence over evaluation measures, search strategies, and selection methods.

acteristics can be found in Table 5.12. In order to simplify further this picture, evaluation measures are not included. It is obvious that accuracy is used as a measure for methods that are of a wrapper model. For the filter selectors, SFG and Relief use classic evaluation measures and the rest use consistency measures (Figure 2.2 in Chapter 2). Unfilled spaces in Figure 5.12 do not mean that there exist no feature selection methods. This figure is just another example to categorize the discussed feature selection methods with many factors. In (Dash and Liu, 1997), an extensive review of existing feature selection methods is presented. An attempt is made to identify missing combinations of evaluation measures and feature generation schemes. Representative feature selection algorithms for each combination are illustrated through examples.

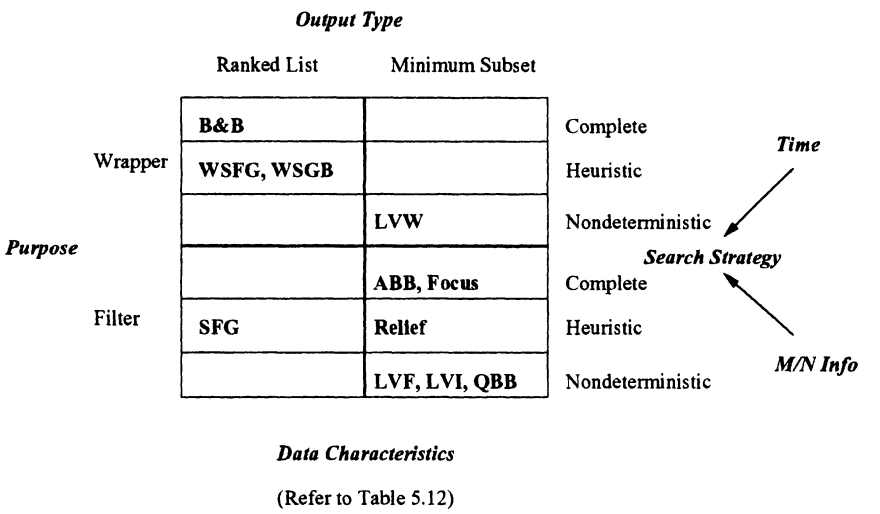


Figure 5.12. Choosing a method: bold italic words are the major factors with their values in normal fonts; search strategy is also in bold italic; feature selection methods are in bold type.

5.7 CONCLUSIONS

We start with motivations for evaluation in this chapter. The eventual goal of evaluation is to enable us to apply feature selection methods well. We first discuss performance assessment by spelling out three key quantitative measures: predictive accuracy, simplicity of the learned results and the number of selected features. We briefly introduce relevant statistical concepts such as covariance, correlation, hypothesis testing, level of significance, error estimation, and describe evaluation methods for classification. Then we move on to discuss how to evaluate selected features, direct or indirect evaluation, and when to use which. To reinforce the understanding of these evaluation methods, we provide examples of evaluation. In detail, we illustrate the issues of choosing data and classifiers for experimentation taking a goal-oriented stand. Through these detailed examples, we hope that the reader will appreciate what experimentation can do in feature evaluation. After all these, we point out the necessity of balancing among various performance criteria and remind ourselves that in reality, we may be forced to take into account more issues and strive a balanced overall measure. With so many factors to consider, the reader may ask whether it is necessary to conduct experimentation. (Tichy, 1998) collects a number of

arguments against experimentation, refutes them one by one and explains the importance of experiments for computer science in general. It also provides a good list of general readings on experimentation. Towards the end of this chapter (Section 5.6), we are ready to map out the sophisticated relations among the general tasks, data characteristics, search strategies, evaluation measures, and feature selection methods. Again, examples are used to explain the practical issue in data mining and inductive learning - how to efficiently and effectively apply feature selection methods to solving real world problems.

This section also concludes our discussion on feature selection methods for classification. Feature selection can find its application in other areas of data mining. We shall explore further in the next chapter.

References

Boddy, M. and Dean, T. (1994). Deliberation scheduling for problem solving in time-constrained environments. *Artificial Intelligence*, 67(2):245–285.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24:123–140.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283.

Cohen, P. (1995). *Empirical Methods for Artificial Intelligence*. The MIT Press.

Dash, M. and Liu, H. (1997). Feature selection methods for classifications. *Intelligent Data Analysis: An International Journal*, 1(3).

Dietterich, T., Hild, H., and Bakiri, G. (1990). A comparative study of ID3 and backpropagation for English text-to-speech mapping. In *Machine Learning: Proceedings of the Seventh International Conference*. University of Texas, Austin, Texas.

Domingos, P. (1997). Why does bagging work? a Bayesian account and its implications. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 155 - 158. AAAI Press.

Fisher, D. and McKusick, K. (1989). An empirical comparison of ID3 and backpropagation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 788–793.

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

Fu, L. (1994). *Neural Networks in Computer Intelligence*. McGraw-Hill.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Publishing Company, Inc.

Hillier, F. and Lieberman, G. (1990). *Introduction to Operations Research*. McGraw-Hill Publishing Company, 5th edition.

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

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher.

Kazmier, L. and Pohl, N. (1987). *Basic Statistics for Business and Economics*. McGraw-Hill International Editions, 2nd edition.

Kohavi, R. and Wolpert, D. (1996). Bias plus variance decomposition for zero-on loss functions. In Saitta, L., editor, *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 275–273. Morgan Kaufmann Publishers, Inc.

Kong, E. and Dietterich, T. (1995). Error-correcting output coding corrects bias and variance. In Prieditis, A. and Russell, S., editors, *Machine Learning: Proceedings of the Twelfth International Conference*, pages 313–321. Morgan Kaufmann Publishers, Inc.

Kononenko, I. (1994). Estimating attributes : Analysis and extension of RELIEF. In *Proceedings of the European Conference on Machine Learning*, pages 171–182.

Liu, H. and Motoda, H., editors (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Liu, H. and Setiono, R. (1998). Scalable feature selection for large sized databases. In *Proceedings of the Fourth World Congress on Expert Systems (WCES'98)*. Morgan Kaufmann Publishers.

Mendenhall, W. and Sincich, T. (1995). *Statistics for Engineering and The Sciences*. Prentice Hall International, 4th edition.

Merz, C. and Murphy, P. (1996). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science.

Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in Artificial Intelligence.

Mitch, T. (1997). *Machine Learning*. McGraw-Hill.

Murdoch, J. and Barnes, J. (1993). *Statistical Tables for Science, Engineering, Management and Business Studies*. The Macmillan Press, 3rd edition.

Murphy, P. and Pazzani, M. (1994). Exploring the decision forest: An empirical investigation of Occam's razor in decision tree induction. *Journal of Art. Intel. Res.*, 1:257–319.

Pudil, P. and Novovicova, J. (1998). *Novel Methods for Subset Selection with Respect to Problem Knowledge*, pages 101 – 116. In (Liu and Motoda, 1998).

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

Quinlan, J. (1994). Comparing connectionist and symbolic learning methods. In Hanson, S., Drastall, G., and Rivest, R., editors, *Computational Learning*

Theory and Natural Learning Systems, volume 1, pages 445–456. A Bradford Book, The MIT Press.

Russell, S. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Prentice Hall.

Setiono, R. and Liu, H. (1995). Understanding neural networks via rule extraction. In *Proceedings of International Joint Conference on AI.*

Shavlik, J., Mooney, R., and Towell, G. (1991). Symbolic and neural learning algorithms: An experimental comparison. *Machine Learning*, 6(2):111–143.

Thrun, S. and et al (1991). The monk's problems: A performance comarison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

Tichy, W. (1998). Should computer scientists experiment more? *IEEE Computer*, 3(5).

Towell, G. and Shavlik, J. (1993). Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101.

Weiss, S. M. and Kulikowski, C. A. (1991). *Computer Systems That Learn.* Morgan Kaufmann Publishers, San Mateo, California.

Yen, S.-J. and Chen, A. (1995). An efficient algorithm for deriving compact rules from databases. In *Proceedings of the Fourth International Conference on Database Systems for Advanced Applications.*

Zell, A. and et al (1995). Stuttgart neural network simulator (SNNS), user manual, version 4.1. Technical Report 6/95, Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, FTP: ftp.informatik.uni-stuttgart.de/pub/SNNS.

Zilberstein, S. (1996). Using anytime algorithms in intelligent systems. *AI Magazine*, pages 73–83.

Notes

1. Lazy learning does not require a structural representation since it delays learning until classification is requested. Nevertheless, feature selection is much needed due to the curse of dimensionality.

2. The reader may question the comprehensibility of a neural network with regard to that of the data. Researchers (Towell and Shavlik, 93, Li, 94, Setiono and Liu, 95) have been successful in turning learned networks of certain types into rules.

3. It is much easier to reject a hypothesis than to accept it.

4. The reader may notice that in effect, in order to estimate accuracy for deleted features, we need a classifier. That means we have already a wrapper feature selector in the one-step procedure.

5. Many induction algorithms, as those used in a wrapper model, apply bias to narrowing down the search space. In this sense, they also rely on heuristics.

6. All feature selection methods mentioned in this book are data driven.

7. In some particular situations such as visualization, we can pre-specify a number for M. See the discussion about feature extraction in Chapter 6.

6 FEATURE TRANSFORMATION AND DIMENSIONALITY REDUCTION

In the previous chapters, we focused on feature subset selection. We now discuss related and/or less developed topics with respect to feature transformation and dimensionality reduction. The first two sections are about feature transformation, which introduce techniques in Statistics, Machine Learning, and Knowledge Discovery. The third section discusses feature discretization which is closely related to dimensionality reduction. If subset selection allows reduction in one dimension, feature discretization could enable reduction along two dimensions. In the fourth section, we go beyond the classification model and explore feature selection without class information. Data without class information are unsupervised data (unlabeled). As we move from supervised (labeled) data to unsupervised data, we also step into a territory that is not as well explored as feature selection for classification. However, we foresee the rising need for unsupervised feature selection and expect more work to be carried out in the near future.

6.1 FEATURE EXTRACTION

A common characteristic of feature extraction methods is that they all produce new features **y** based on the original features **x**. Usually, the dimensionality of **y** is reduced and is smaller than that of **x**. The side effect is that we still need to have all the original features. This group of methods is used not for reducing dimensionality of **x**, but for various purposes of an enabling role - after feature extraction, representation of data is changed so that many techniques such as visualization, decision tree building can be conveniently used.

Feature extraction started, as early as in 60's and 70's, as a problem of finding the intrinsic dimensionality of a data set - the minimum number of independent features required to generate the instances (Wyse et al., 1980). Put it in another way, the intrinsic dimensionality represents a lower bound on the number of features needed to represent the data accurately. It has been used to guide the extraction of features essential to the design of a classifier (Pettis et al., 1979). It helps address the question: Does a reasonable two or three dimensional representation of the data exist that may be analyzed visually? In addition to the above two usages, we are also concerned about issues to change the representation of data so as to simplify problem solving and to recover the real features. We introduce three methods of feature extraction.

A visualization perspective:. Data of high dimensions cannot be analyzed visually. It is often necessary to reduce its dimensionality in order to visualize the data. The most popular method of determining topological dimensionality is the Karhunen-Loeve (K-L) method (also called Principal Component Analysis[1]) which is based on the eigenvalues of a covariance matrix (**R**) computed from the data. The M eigenvectors corresponding to the M largest eigenvalues of **R** define a linear transformation from the N-dimensional space to an M-dimensional space in which the features are uncorrelated. This property of uncorrelated features is derived from a theorem stating that if the eigenvalues of a matrix are distinct, then the associated eigenvectors are linearly independent. See the proof in (Pettofrezzo, 1966). The ratio of the sum of the M largest eigenvalues of **R** to the trace of **R** is the fraction of the variance retained in the M-dimensional space. If the eigenvalues are labeled so that $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$, then the ratio can be written as

$$r = \frac{\sum_{i=1}^{U} \lambda_i}{\sum_{i=1}^{N} \lambda_i}. \tag{6.1}$$

When the ratio r is sufficiently large (0.9 or more), evaluation measure U is taken as an estimate of the intrinsic dimensionality.

For the purpose of visualization, one may take the M features corresponding to the M largest eigenvalues of $\mathbf{R}$. The K-L method is computationally inexpensive, but it requires characterizing data with covariance matrix $\mathbf{R}$ and some dangers may ensue. One of them is that correlations among multiple features cannot be characterized. Another danger is that class information is left out.

By setting a proper threshold for r (> 0.95) as in Equation 6.1, i.e., to choose the minimum number of largest M extracted features such that $r > 0.95$, the K-L method extracted two features for the Iris data, reported in (Wyse et al., 1980). We repeat this experiment by running principal component analysis from SAS as follows:

1. Normalize each feature's values into a range $[0, 1]$;

2. Obtain the correlation matrix for the four original features,

	ATTR1	ATTR2	ATTR3	ATTR4
ATTR1	1.0000	-.1094	0.8718	0.8180
ATTR2	-.1094	1.0000	-.4205	-.3565
ATTR3	0.8718	-.4205	1.0000	0.9628
ATTR4	0.8180	-.3565	0.9628	1.0000

3. Find eigenvalues of the correlation matrix,

	Eigenvalue	Difference	Proportion	Cumulative
LAMBDA1	2.91082	1.98960	0.727705	0.72770
LAMBDA2	0.92122	0.77387	0.230305	0.95801
LAMBDA3	0.14735	0.12675	0.036838	0.99485
LAMBDA4	0.02061		0.005152	1.00000

4. Select the eigenvectors according to r. This is equivalent to choosing the minimum number of `LAMBDAi` such that the `Cumulative` is just over the threshold (i.e., 0.95 for Iris data). Therefore, the largest two eigenvalues should be chosen.

5. Output the chosen eigenvectors associated with the chosen eigenvalues. They are the first two in the following.

	Eigenvectors			
	VECTOR1	VECTOR2	VECTOR3	VECTOR4
ATTR1	0.522372	0.372318	-.721017	-.261996
ATTR2	-.263355	0.925556	0.242033	0.124135
ATTR3	0.581254	0.021095	0.140892	0.801154

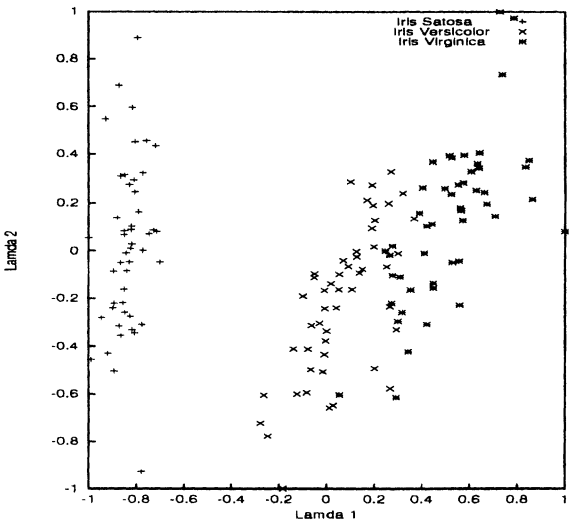


Figure 6.1. The Iris data is represented by two features extracted by Principal Component Analysis. The original data is 4-dimensional.

```
ATTR4    0.565611    0.065416    0.633801    -.523546
```

where eigenvector **VECTORi** is corresponding to eigenvalue **LAMBDAi**. Thus, we obtain a 4×2 transformation matrix $\mathbf{M}$.

6. Transform the normalized data D_{old} into data D_{new} of extracted features as follows:

$$D_{new} = D_{old}\mathbf{M}.$$

The resulting data is of 2-dimensional having the original class label attached to each instance.

The transformed data can be plotted in a 2-dimensional space (Figure 6.1) with their classes labeled with different symbols.

Recovering real features:. One often cited problem is recovering implicit 2-dimensional structure from 4-dimensional inputs. It is a 2-dimensional surface and we construct a training data set by repeatedly picking a point on the surface and then recording the distances between the chosen point and four fixed landmarks in the space[2]. The Kohonen net (self-organizing feature

maps) (Kohonen, 1984) has been employed to recover the original two features since the space we measure the distances is a 2-dimensional surface.

A Kohonen net consists of one layer of input units and one layer of output units, and units on the two layers are fully connected. Each connection is attached with a weight which is randomly initialized. Each output unit is corresponding to a weight vector. Competitive learning is its underlying principle, and no class information is needed. The competition is realized as follows:

1. Find the weight vector which is closest to the presented input vector. Call this weight vector the winner.
2. Modify the winner so as to move it closer to the input vector.
3. Repeat the first two steps through the training data set.

There are various ways of measuring the distance between weight vectors and input vectors. The simplest method involves computing the inner product of the two vectors assuming that both are normalized. Modifying the winner is to modify its weights so as to make them more similar to the values in the input vector. In a Kohonen net, the winner and all units within its neighborhood are updated using the Kohonen rule:

$$\mathbf{w}_i(q) = \mathbf{w}_i(q-1) + \alpha(\mathbf{p}(q) - \mathbf{w}_i(q-1), \qquad (6.2)$$

$$i \in N_{i^*}(d)$$

where $\mathbf{p}$ is the input vector presented at the moment, q is a time stamp such that $\mathbf{w}(q)$ is the current (just updated) weight vector and $\mathbf{w}(q-1)$ is the previous one, and the neighborhood $N_{i^*}(d)$ contains the indices for all of the units that lie within a radius d of the winner i^*:

$$N_i(d) = \{j, d_{ij} \leq d\}.$$

Here d is measured in terms of Euclidean distance, can be 1, 2 or more. In training a Kohonen net, d can be set initially large and gradually reduced to 0.

Back to our example of recovering 2-dimensional features, we have four input units with respect to four distances for each point picked on the surface, and $k \times k$ output units arranged in a 2-dimensional pattern, where k is determined by the user and designer of the network (if he decides to have 64 output units, for instance, then k is 8). Over time, each output unit gradually acquires a weight vector which causes it to be maximally responsive to inputs that were generated from a point on the original 2-dimensional surface which corresponds to its own position in the output array. At the end, the output units self-organize so as to

recover the 2-dimensional structure of the 4-dimensional input data. Interested readers can find more details and examples in (Hagan et al., 1996). The pattern arrangement of the output units can be 1-dimensional, 2-dimensional, or higher dimensional. It is the key to recovering real features. Therefore, if we know the real features roughly, a Kohonen net can help us get the details of them. Otherwise, it is an art for us to determine a proper pattern without any prior knowledge about what needs to be found.

Using class information:. The two methods described above do not use class information even if it is available. This poses a significant disadvantage in addition to other shortcomings they possess. (Setiono and Liu, 1998) propose to employ a standard two-layer feedforward network for feature extraction. The basic idea is to train the network until a pre-specified predictive accuracy (or a threshold) is achieved, prune it while maintaining the accuracy (or simplifying the network without sacrificing the accuracy), and then take hidden units in the pruned network as the new features. This neural network feature extraction algorithm consists of three parts:

1. **Network construction:** a network *Net* is constructed by adding one hidden unit at a time if necessary. In order to better estimate its predictive accuracy and to avoid overfitting the data, a validation data set (a part of the training set) is used. That is, the training data is further divided into two sets. In total, there should be three data sets. If we adopt a 10-fold cross validation test, for example, we divide the whole data into 90% and 10% for training and testing respectively. With the new arrangement, we have 80% for training, 10% for validation, and 10% for testing.

2. **Network pruning:** *Net* is pruned based on the magnitudes of the connections between units. After some connections are removed, *Net* is retrained. Pruning is terminated when the removal of a connection would cause the decrease of the accuracy below the threshold. In this process, irrelevant and/or redundant connections are removed. *Net* is further simplified. Some (often many) input units may not have any connections to hidden units; and a hidden unit without any inward connections from input units can also be removed.

3. **Data translation:** After the above two steps, obtained is a simplified network *Net*. For each instance of the data, we take hidden unit activation values of *Net* as the new values of the instance; the number of hidden units determines the number of new features. More often than not, the number of hidden units is much smaller than the number of input units. Thus, we can extract a smaller number of features out of the original features.

The essence of the first two steps is to find a necessary network, which is sufficiently simple, for the problem at hand. This feature extraction algorithm can take into account class information, handle both discrete and continuous data. Besides having to specify a "good" threshold for predictive accuracy, one also needs to carefully choose some parameters for the error function used in network training and pruning. As was reported in (Setiono and Liu, 1998), two hidden units remain after network training and pruning. Hence, two features are extracted for the Iris data. The instances described by the two features are plotted in Figure 6.2. Comparing the figure with Figure 6.1, we notice that class information makes a difference. Instances of different classes are clustered closely together within the class (Figure 6.2) and demarcations are clearer than those in Figure 6.1. As it is also known that (later in the section of discretization of this chapter) that third and fourth features (Petal Length and Petal Width) of the Iris data are good enough to describe the data, out of curiosity, we plot the data with the two features in Figure 6.3. In order to facilitate the comparison, we normalize the data into a range of $[0, 1]$ and normalize the transformed data into a range of $[-1, 1]$. The first normalization is basically required by neural networks to make sure that each input (feature) has its due influence in network training. Visually, Figures 6.1 and 6.3 have no much difference.

Summary. Three methods of feature extraction are briefly illustrated here. The first two methods (the K-L method and the Kohonen net) can be applied to unsupervised data, but the last method can only be applied to supervised data. Except that, any conclusion on the superiority of one method over another is premature. Extensive experiments and thorough study should be done to draw a fair conclusion on and to provide a guideline about which method is preferred under certain circumstances. However, these method bear some resemblances: (1) all extract *new* features which are different from original features in nature; (2) each method requires some efforts to decide on ideal parameters; and (3) the number of new features is smaller than the original ones.

6.2 FEATURE CONSTRUCTION

Because of the consideration of efficiency, many selective inductive learning systems employ the divide-and-conquer strategy. That is, by repeatedly selecting the most promising individual feature, the selective algorithms divide the data into smaller but more uniformed partitions in terms of class values. Examples of such learning systems are CART (Breiman et al., 1984), ID3 and C4.5 (Quinlan, 1986, Quinlan, 1993), and CN2 (Clark and Niblett, 1989), among many. They can realize their maximum potential if the original features are directly

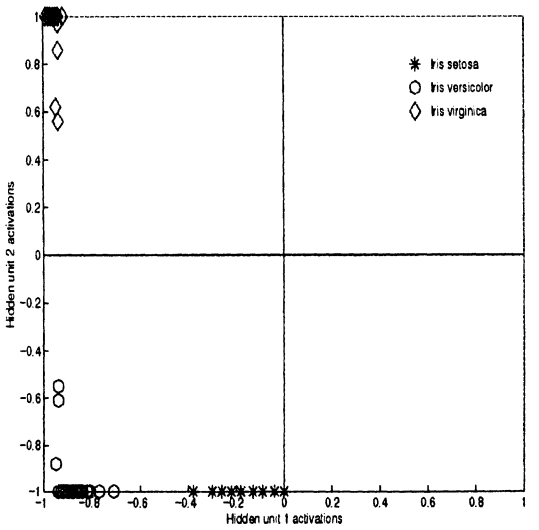


Figure 6.2. The Iris data is represented by two features extracted from a neural network. The original data is of 4-dimensional.

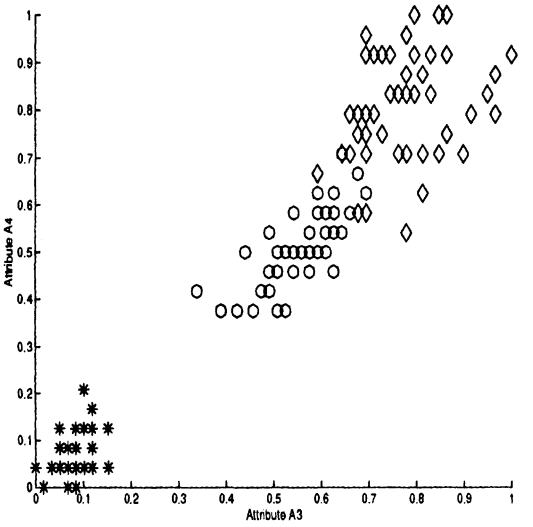


Figure 6.3. The Iris data is represented by two original features: Petal Length and Petal Width.

relevant to characterizing the concepts to be learned. However, the divide-and-conquer strategy also brings with it some problems such as replication, repetition, and fragmentation. Let's use decision tree induction as an example of selective induction algorithms to elucidate what these problems are. The replication problem can be observed if subtrees are replicated in a decision tree; the repetition (or repeated testing) problem is present if features are repeatedly tested (more than once) along a path in a decision tree; and the fragmentation problem exists if data is gradually partitioned into small fragments (Pagallo and Haussler, 1990, Brodley and Utgoff, 1995). Replication and repetition always imply fragmentation, but fragmentation may occur without any replication or repetition if many features need to be tested (Friedman et al., 1996). There is a detailed account of the fragmentation problem in (Vilalta et al., 1997). Constructive inductive learning systems have been suggested to overcome the limitation of representation in original features and attack the problems of replication, repetition, and fragmentation. New compound features are constructed out of the original features. The primary goals of constructing new features include improving the overall predictive accuracy of a classifier, decreasing the overall complexity of learned concepts, or both.

Feature construction is generally defined as the application of a set of constructive operators to a set of existing features, resulting in the construction of one or more new features intended for use in describing the target concept (Matheus and Rendell, 1989). All new features are defined in terms of existing features, as such no inherently new information is added through feature construction. Feature construction can be applied repeatedly to generate more compound features. Constructed features are used by a selective induction system as candidates to divide and conquer the problem of learning target concepts.

The most common constructive operator is *product* for nominal features, *and* for binary features. Using it alone has shown great success in combating the replication problem in decision tree building. From a decision tree that suffers from the replication problem, one can observe the same decision test ($x_3 = 1$?) leading to the sub-tree being replicated in the tree. As a matter of fact, all selective induction systems are prone to the replication problem, which leads to a severe problem called the fragmentation problem, the data is quickly split into fragments so small that statistical reliability of each fragment is lost. In order to exhibit the replication problem, we use a simple concept of four binary features: $x_1x_2 + \bar{x}_3x_4$, its value is either 1 or 0. The reader can try to build his own decision tree and he will observe the replication problem no matter how he chooses one feature at a time to split the data. One of the trees built is shown in figure 6.4a.

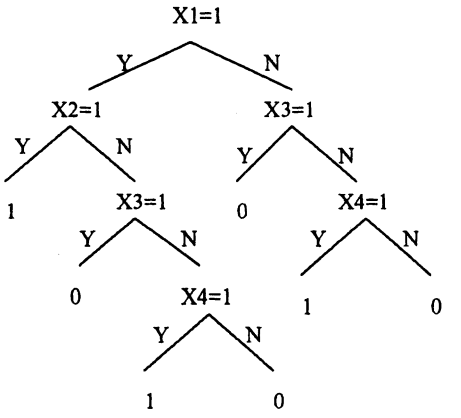


Figure 6.4a. Illustration of the replication problem in a decision tree building for target concept $x_1x_2 + \bar{x_3}x_4$.

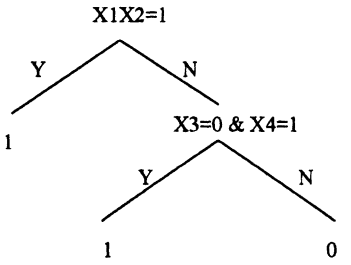


Figure 6.4b. The removal of replication in a decision tree building using constructed features x_1x_2 and $\bar{x_3}x_4$.

After applying feature construction, assuming we have two new features x_1x_2 and $\bar{x_3}x_4$, we can build a much simpler tree (seen in Figure 6.4b). Analyzing the two figures, we can also see the mitigation of the fragmentation problem using the constructed features. The tree in Figure 6.4a has a total of seven branches, and the deepest branch requires four decision tests. Assuming that data is evenly partitioned at each test, at the last test, there are only one eighth of the data for testing. This situation is improved as seen in Figure 6.4b in which there are only three branches and a total of two decision tests. The data at the second test is half of the original.

Through years of research, many constructive operators are designed and implemented. The common ones are equivalence (i.e., a new feature's value is 1 if two features $x = y$, otherwise 0.), inequalities, addition, subtraction, division, maximum(S) (i.e., a new feature's value is the maximum value of all features in feature set S), minimum(S), average(S). One extremely useful constructive operator is Count(S, C) shown in (Bloedorn and Michalski, 1998). It denotes number of features in set S satisfying condition C. The new feature found for the second Monk's problem (Monk2) is, after feature construction, Count($\{A_1, ...A_6\}$,*FirstValue*) = 2 which actually states the concept to be learned: an instance belongs to the concept, if exactly two of six features take their first value. After this new feature is defined, Monk2 is simplified to a linearly separable problem from a problem with high interdependence between features. In this case, what we need for learning is the new feature constructed by operator Count, although the original features are still needed in construction. For a selective inductive learning algorithm, however, no selection is necessary in this case. Another similar example is reported in (Michie et al., 1994) about the Iris data. Among four original features (Sepal length, Sepal width, Petal length, and Petal width), only one compound feature is necessary, that is, Petal area which is the multiplication of Petal length and Petal width.

Obviously, there could be infinitely many constructive operators and for some operators, a large number of ways of applying them. For the example of operator Count, in a more general setting, one should try other values besides *FirstValue*. For a more complicated case (in the spirit of Monk2), e.g., an instance belongs to the concept, if exactly two of six features take their values as follows: one takes its last value and the other takes its second value. Naturally, the above defined operator Count won't work and a new operator is needed. Researchers realize that the problem with constructive induction is that it requires strong biases in order to produce valid generalizations. Strong and appropriate generalization biases are most readily available in the form of domain knowledge. The strong bias for generalization might have come from knowledge that in checkers, for example, a feature useful at one board location can often be translated to other board locations. The use of relevant domain knowledge can significantly affect the quality of constructed features. (Wnek and Michalski, 1994) give a classification of constructive induction systems: data-driven (DCI), knowledge-driven (KCI), hypothesis-driven (HCI), and multistrategy constructive induction (MCI) based on what is used to construct compound features. A variety of fragmentary knowledge is also identified in (Donoho and Rendell, 1996) for the purpose of feature construction.

Summary. Feature construction has long been considered a powerful tool for increasing both accuracy and understanding of structure, particularly in

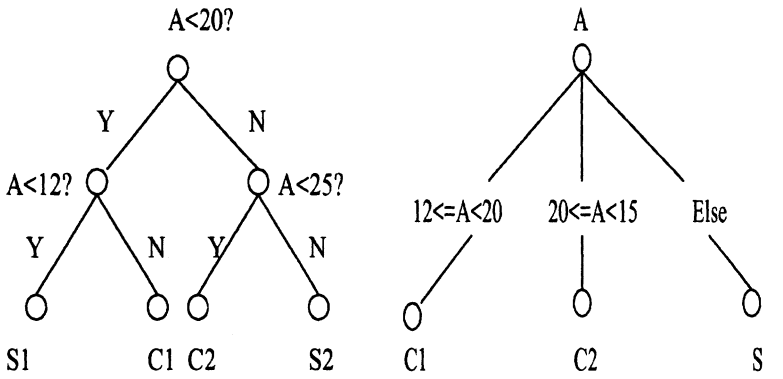


Figure 6.5. The effect of discretization in decision tree building.

high dimensional problems (Breiman et al., 1984). Constructed features help alleviate the replication and fragmentation problems and build compact and accurate decision trees. Feature construction is also an effective tool to handle concept drifting. This is because we can perform feature construction, from time to time, on original features and form compound feature to capture drifted concepts. Representative examples of systems that perform feature construction include FRINGE (Pagallo, 1989), BACON (Langley et al., 1987), Stagger (Shlimmer, 1987), DUCE (Muggleton, 1987), PLSO (Rendell, 1985), CTRE (Matheus and Rendell, 1989), CN2-MCI (Kramer, 1994), and AQ17-DCI (Bloedorn and Michalski, 1998).

6.3 FEATURE DISCRETIZATION

This is another effective enabling technique which receives a great deal of attention in data mining and machine learning communities. The task is to discretize the values of continuous features into a smaller number of intervals, where each interval is mapped to a discrete symbol. The benefits of doing so include: (1) simplify the data description; (2) improve understandability of data as well as results from any data mining or machine learning systems; (3) making data accessible to more data mining or machine learning algorithms; and (4) alleviating the repetition problem in decision tree induction. In the problem of repetition , a feature is repeatedly tested along a path in a decision tree (Fayyad and Irani, 1993, Brodley and Utgoff, 1995). One example is that a continuous feature A is repeatedly tested before discretization, and the number of tests is reduced significantly after discretization (see Figure 6.5).

An old fashion way of discretization is to do it manually. For instance, one can discretize a person's age (0 - 150 years old) into *child, adolescent, adult, middle age, elderly* and decide on the cut-points (11 years old is one that demarcates the boundary between child and adolescent, e.g.) based on common sense or consensus. Without any knowledge about a feature, this manual way of discretization can be rather arbitrary. Hence desired is an automated way of discretization based on the data. Two simple methods are equal-width-intervals (EWI) and equal-frequency-intervals (EFI). EWI divides the sorted values of a feature between the minimum and maximum into n intervals of equal size, where n is specified by the user. EFI finds $n-1$ boundaries for n intervals so that each interval contains approximately the same number of instances of the data. The two methods do not rely on the class information so they can be used in the context of unsupervised learning. They are also easy to implement and can produce reasonable results. However, they are prone to errors since they do not consider the class information even if it is available so that it is unlikely that the interval boundaries occur in the places that best facilitate accurate classification, and since the one parameter n can only be guessed for all features, though it may be the case that each feature has its own ideal parameter for the number of intervals.

Many classification algorithms such as CART (Breiman et al., 1984) and C4.5 (Quinlan, 1993) do consider the class information when they branch on a continuous feature. One cut-point is sought to divide the sorted values of the feature into two branches - binary intervals. In C4.5 (similarly in ID3 (Quinlan, 1986)), since a continuous feature can have n distinct values in a training data set, $n-1$ cut-points are searched to find the best one that maximizes the information gain measure. Discretizing features while inducing a decision tree is called dynamic discretization. Discretizing features before tree induction is called static discretization. (Catlett, 1991) shows via experiments that it is preferable to use static discretization. Researchers later extend binary intervals to multiple intervals. (Catlett, 1991) describes D-2 which applies the binary interval procedure recursively as long as the information gain of each split exceed some threshold and a limit on the maximum number of intervals has not been exceeded. (Fayyad and Irani, 1993) applies the minimum description length principle to determine the number of intervals.

ChiMerge is another automated discretization algorithm (Kerber, 1992) that operationalizes the quality of multiple intervals by using the χ^2 statistic to determine if the relative class frequencies of adjacent intervals are distinctly different or if they are sufficiently similar to justify merging them into a single interval. The χ^2 statistic can be used to test the independence of two variables (contingency-table tests). Independence implies that knowledge of the category in which an observation is classified with respect to one variable has no effect

on the probability of being in one of the several data categories with respect to the other variable. Used in discretization for merging two intervals of a continuous feature, it tests the hypothesis that the feature and the class are independent (the null hypothesis H_0[3]). We reject H_0 if the calculated χ^2 is greater than θ - critical χ^2 value which is determined by degree of freedom of the data and the user's confidence, and can be found in the table of the χ^2 distribution. If the conclusion of the χ^2 test is that the class is independent of the feature of the intervals, then the intervals of the concerned feature should be merged, otherwise it indicates that the difference in relative class frequencies is statistically significant and therefore the intervals should remain as they are.

The ChiMerge algorithm is shown in Table 6.1 which consists of three basic steps for discretization of each continuous feature: (1) sort the data with the values of the feature in ascending order; (2) repeat until no χ^2 of any two intervals is less than θ (which is determined by *conf* (confidence level)) [4]. In the simplest implementation, after each merger of two intervals, χ^2's of all remaining intervals are recalculated, and the smallest value is found. If it is smaller than θ, the merging continues; if the smallest χ^2 is greater than θ, no merger can be made; and (3) check whether the number of remaining intervals is greater than *max*, if yes, *conf* is increased, new θ is found, and step (2) is repeated. Step (3) is to make sure that not too many intervals remain after discretization. Choosing higher values for *conf* causes the merging process to continue longer, resulting in discretization with fewer and larger intervals.

For the reader's convenience, we provide the formula for computing χ^2 here:

$$\chi^2 = \sum_{i=1}^{2} \sum_{j=1}^{k} \frac{(A_{ij} - E_{ij})^2}{E_{ij}} \tag{6.3}$$

where:

k = number of classes,

A_{ij} = number of instances in the ith interval, jth class,

R_i = number of instances in the ith interval = $\sum_{j=1}^{k} A_{ij}$,

C_j = number of instances in the jth class = $\sum_{i=1}^{2} A_{ij}$,

N = total number of instances = $\sum_{i=1}^{2} R_i$,

E_{ij} = expected frequency of $A_{ij} = R_i * C_j / N$. If either R_i or C_j is 0, E_{ij} is set to 0.1. This modification is to avoid too small a value in the denominator. The degree of freedom of the χ^2 statistic is one less the number of classes.

Why do we need to enforce the number of remaining intervals not to be too large as in step (3)? If too fine intervals are created, which implies many intervals exist, a discretized feature can still have many discrete values; if this feature is chosen[5] to branch in tree induction, it can cause the fragmentation problem (Pagallo and Haussler, 1990), i.e., the data is quickly split into small

Table 6.1. ChiMerge: a χ^2 statistic based discretization algorithm.

ChiMerge Algorithm

Input: $conf = 0.90$ - threshold (confidence level) of the χ^2 test
$max = 7$ - the maximum number of intervals
D - data

/* Both $conf$ and max are set according to (Kerber, 1992) */

initialize: $\theta = \chi^2$ value at $conf$

for each feature x_i, $i \in \{1, \ldots, N\}$

1: `sort`(x_i, D)

2: **while** *true*

``χ^2-`cal`(x_i, D)

χ^2 = `find-smallest-`χ^2 /*among all adjacent intervals*/

if $\chi^2 > \theta$

`break`

else `merge-two-intervals-with-`χ^2

end while

3: if #Intvl for $x_i > max$

`increase` θ

go to step 2:

end for

fragments and results in sample shortage for the building of next subtrees. It is important to discretize (i.e., to merge values into intervals) as much as possible (to have fewer intervals). Some researchers suggest a threshold for the maximum number of intervals *allowed* for each feature (as we see, it is 7 in ChiMerge), if the number of intervals exceeds this maximum number, discretization should continue. This threshold is also required for algorithms such as equal-width-intervals, equal-frequency-intervals, and D-2 as one of the stopping criteria.

ChiMerge has a principled way of discretization since it bases its merger on the χ^2 statistic and takes into account of the class information. However, it is required to specify $conf$ and max. As we know, in general, it is ideal that each continuous feature has its own $conf$ especially when some of the features are irrelevant and redundant. The current form of ChiMerge cannot handle various $conf$'s. An additional criterion is needed to automatically determine each feature's $conf$, that is, increasing $conf$ to its maximum value which leads to the minimum number of intervals for the feature. If there is such a criterion, we need not search for optimal $conf$ and max as in ChiMerge. The new criterion is to make sure the fidelity of the discretized data is the same as the original data. We're aware that the inconsistency criterion can help in this regard. With the new criterion, we can start $conf$ with a low value (e.g., 0.5) and there is no need for max. This finding gives rise to Chi2 which was first proposed in (Liu and Setiono, 1995) and later shown to be able to remove redundant and/or irrelevant continuous features in (Liu and Setiono, 1997). Thus, obtained is a discretization algorithm that can also select features. A simplified version of Chi2 is shown in Table 6.2.

The outer loop **do until** is wrapped around a ChiMerge like algorithm with each feature takes turn to be discretized. In step 3, the inconsistency of the discretized data is checked to ensure that the aggressive discretization dose not lose the fidelity of the data - classes can still be distinguished as before. We can view the difference between ChiMerge and Chi2 when they are used to discretize the Iris data (Fisher, 1936). Chi2 discretized two features (Sepal-length and Sepal-width) into one interval each so that the two features are removed, the other two features have three intervals each (Liu and Setiono, 1997), while the results by ChiMerge (Kerber, 1992) showed that 5, 4, 4, and 4 intervals for sepal-length, sepal-width, petal-length, and petal-width, respectively. Chi2 also allows under- or over-discretization for data with both discrete and continuous features. The discretization is either over or under because we cannot assume that all extant discrete features are irrelevant or relevant. If we include the extant discrete features in inconsistency checking and at least one of them is irrelevant, this irrelevant feature would contribute negatively in inconsistency checking, indicating a falsely fewer number of inconsistency

Table 6.2. Chi2: a χ^2 statistic based, aggressive discretization algorithm.

Chi2 Algorithm

Input: $conf = 0.5$ - threshold (confidence level)
δ - the allowed inconsistency, 0 by default
D - data
initialize: all $\theta_i = \chi^2$ value at $conf$
do until no-feature-can-be-merged
for each mergeable feature x_i
1: `sort`(x_i, D)
2: **while** *true*
χ^2-`cal`(x_i, D)
χ^2 = `find-smallest-`χ^2
/*among all adjacent intervals*/
if $\chi^2 > \theta_i$
break from the while loop
else `merge-two-intervals-with-`χ^2
end while
3: if (`inconCheck`$(D) \leq \delta$) /*D is discretized.*/
increase θ_i /*Prepare for the next round*/
else
mark x_i as not mergeable /*This feature is out.*/
end for
end do

than if the irrelevant feature is not included, hence over-discretization; and if we exclude the extant discrete features in inconsistency checking and at least one of them is relevant, the result of discretization could be more accurate if that excluded relevant feature were included in inconsistency checking, hence, it is under-discretization. In general, if the extant discrete features are considered in inconsistency checking during discretization, it is over-discretization; otherwise, it is under-discretization if we do not take in account the discrete features in inconsistency checking[6]. ChiMerge need not make such distinction. It usually under-discretizes the data due to its lack of the additional criterion helping determine the optimal threshold θ.

Some further problems with discretization. Discretization can be done in two fashions: univariate or multivariate. A univariate method considers discretization one feature at a time independently of other features. All the methods mentioned so far belong to univariate discretization. There are some further issues to be addressed: (1) the ordering problem - which feature should be discretized first; and (2) the interdependence problem - some continuous features collectively determine the class labels. ChiMerge is a good example of univariate discretization. It specifies a significance value for all features' χ^2 tests and a threshold for the maximum number of intervals that there exists after discretization. So, there is no ordering problem. If one wants to aggressively discretize the features as Chi2 does, the ordering problem becomes obvious. Some features could be eliminated before others. For example, the first feature chosen for aggressive discretization is most likely to be discretized into only one value, therefore eliminated first. This is because it is more likely to satisfy the inconsistency criterion with other features not being discretized. Thus designed in Chi2 is a round robin mechanism to mitigate the ordering problem.

Although the inconsistency checking also helps alleviate the interdependence problem, it is better battled with multivariate discretization. The run time complexity is, however, usually too prohibitive to implement practical multivariate discretization systems. That is why we do not hear much about them.

Summary. The discretization methods are commonly used to quantize the values of features. Properly discretized data can simplify the task of learning and improve the comprehensibility of the learned results. Many discretization methods exist. Some require class information, and some don't. This section shows that feature discretization can lead to feature selection. Representative discretization systems are ID3 (Quinlan, 1986) and C4.5 (Quinlan, 1993) in which binarization is used, D-2 (Catlett, 1991) that binarizes a continuous feature recursively, (Fayyad and Irani, 1993)'s system that employs the Minimum Description Length principal to determine when recursive binarization should

stop, (Cerquides and de Mantaras, 1997)'s system that uses the Mantaras distance in discretization, 1R discretizer inspired by the 1R classifier (Holte, 1993), and Zeta discretizer (Ho and Scott, 1997) based on measuring the strength of association between a continuous feature with the class. A brief survey can be found in (Dougherty et al., 1995).

6.4 BEYOND THE CLASSIFICATION MODEL

Recall that data are usually collected for reasons not directly linked to data mining. It is common that data are unsupervised. Special data mining tools have been devised for unsupervised data from statistics, pattern recognitions, machine learning, and database communities, such as numerical clustering, conceptual clustering, association rules. The reasoning about the need for feature selection for classification applies to clustering and association rules . The difference now is that there is no class information. Lack of this information often means that tasks of feature selection are more arduous.

6.4.1 Unsupervised feature selection: clustering

Human beings tend to group things into clusters without necessarily having a particular goal or purpose in mind. Once a cluster is discovered, it can provide a starting point for deeper exploration. In this sense, clustering can naturally be applied to feature selection. Assuming relevant features are those that can effectively define formed clusters, we can simply select these features and discard those not used by any cluster representation. This is the basic idea of using clustering to select features. However, there exist many kinds of clustering algorithms. These clustering methods may have varied effects on selecting features. The clustering algorithms can be divided into two camps: hierarchical and non-hierarchical. Nearest neighbor algorithms are non-hierarchical, for example, which is given first as some introduction to clustering. Then we discuss two representative methods: numerical taxonomy and conceptual clustering.

Clustering: Nearest Neighbor algorithms. The idea of this type of clustering is closely related to measuring similarity among instances. It can be found in the literature as early as 70's (Duda and Hart, 1973). The basic concept is that similar instances should belong to the same cluster. The commonly used measures for similarity are distance measures: if instances near each other are similar, they should belong to the same cluster. Euclidean distance, city-block, and Hamming distance are some examples of distance measures. If we have two instances $\mathbf{x}$ and $\mathbf{y}$, the distance between the two can be calculated as follows:

Euclidean distance: $\sqrt{\sum_{i=1}^{N}(\mathbf{x}_i - \mathbf{y}_i)^2}$

City-block: $\sum_{i=1}^{N} |\mathbf{x}_i - \mathbf{y}_i|$

Hamming distance: $\sum_{i=1}^{N} a_i$, where $a_i = 1$ if $\mathbf{x}_i \neq \mathbf{y}_i$, and $a_i = 0$ otherwise. Note that Hamming distance is only for binary data.

The Nearest Neighbor algorithm is as follows: assuming there are n clusters, for each instance to be clustered, calculate its distance to the center of every cluster, find the closest cluster to which if the distance is less than the threshold, assign this instance as a member of the cluster, and the center of the cluster is recalculated; if the distance exceeds the threshold, a new cluster is formed with the instance as the center of the new cluster. In the beginning, no cluster exists, so n is 0. We can choose the first instance as the center of the first cluster. The algorithm is summarized in Table 6.3.

Table 6.3. Nearest Neighbor algorithm (NearN).

NearN Algorithm

Input: d_{max} - threshold for maximum distance of two instances
U_d - distance measure
D - data

initialize: d_{max}
choose first instance I_1 in D as the cluster center C_1
i = 1 /*the number of instances*/
n = 1 /*the number of clusters*/
repeat
for each I_i, calculate its U_d with each C_j
/*C_j is an existing cluster center*/
if $C_{min} = \min_{j=1}^{n}(U_d(C_j, I_i)) < d_{max}$,
add I_i into C_{min}
else use I_i to initiate a new cluster C_i, n++
i++
until all instances are clustered
Output: clusters C_i, $i = 1, 2, ..., n$

There are several important issues to be considered:

1. Conflicts can occur if one instance is equally close to more than one cluster center. This can be solved by using the group-average method. A group distance is defined as the average of the distances for all pairs in the Cartesian product of the two clusters. When clustering an instance to a cluster, it means that we need to calculate the distances between an instance and all

members in a cluster, instead of one center of the cluster. This treatment will certainly reduce the chance of conflicts.

2. Choosing instances as the initial cluster centers can be important in determining the number of and types of clusters formed. A good choice is an instance located in the real center of a cluster to be formed. Nevertheless, this information is usually not known before clusters are formed. If the information is known, one can modify the algorithm by allocating centers of the clusters with the known values and fixing them while clustering. If these clusters are not known beforehand, one can run the algorithm several times by choosing an instance randomly as the first cluster center. One should not be surprised if each run gives a different set of clusters.

3. Determining a proper value for d_{max} is also tricky. Too small or too large a value will lead to too many or too few clusters. If we know beforehand the centers of clusters, we may not need this threshold at all. It is because we can just find the closest cluster for an instance. As we know, it is usually not the case, and on the contrary, we don't have any idea about clusters and their centers. One solution is to rely on visualization of the data.

Nearest neighbor algorithms have also been used for classification problems. For example, a weighted nearest neighbor algorithm was described in (Cost and Salzberg, 1993). Lately, this type of classification algorithms is described as lazy learning (Aha, 1997). Interested readers are referred to these papers as a starting point. Nearest neighbor algorithms have also been thoroughly investigated in the pattern recognition community. See (Dasarathy, 1991) for discussions on various aspects of nearest neighbor algorithms.

Clustering: agglomerative algorithms. All agglomerative algorithms are under the heading of numerical taxonomy. Despite using different distance (or similarity) measures with varied efficiency, they all form clusters in a bottom-up fashion by combining two closest (most similar) instances to form a cluster until there is only one cluster left in the data (a newly formed cluster is also an instance for the next round of clustering). This basic algorithm is shown in Table 6.4. Initially, if there are n instances in a data set, there are n clusters. When a merger occurs, the number of instances is reduced by 1.

One good point of this algorithm is that no parameter is required. However, it is not clear how to treat a cluster as an instance in the later clustering. One way is to simply take average values of the two instances to have the new instance. This only works for numerical data. For symbolic data, it does not make sense to average two symbols. This has to be solved by concept formation which we discuss next. The reader may notice the many unnecessary updates

Table 6.4. An agglomerative algorithm.

Agglo Algorithm
Input: U_d - distance measure
D - data
repeat
for $i = 1$ **to** n
for $j = i + 1$ **to** n
$d_{ij} = U_d(x_i, x_j)$
$d_{i^*j^*} =$ **min**(d_{ij})
$x_k =$ **merge**$(x_i^*$ and $x_j^*)$
removex_i^* and x_j^* from D
add x_k into D
$n - -$
until $n = 1$ /*There is one cluster*/.
Output: a binary tree with original instances as leaves

of distances in the algorithm when a new instance is formed. These updates are unnecessary because only two instances were merged (and we know their indexes), hence only one row (and its corresponding column) should be removed and the other row (and its corresponding column) should be updated with average values. Using a matrix to keep pairwise distances can be a straightforward implementation of the algorithm to avoid unnecessary updates. One more point to note is that different distance measures may give inconsistent results of clustering. In practice, a distance measure is chosen according to the need arising from an application.

In Table 6.4, we describe a bottom-up design of the algorithm, building clusters from instances. Can we reverse the process? That is, we work in a top-down manner: consider the whole data set as a cluster, try to split it into two subsets of data, and do it recursively until each cluster contains only one instance. If we can find an effective splitting measure, in theory, it can be done. However, there are 2^n ways of splitting n elements into two groups for the first split. This time complexity alone is much greater than $O(n^3)$, the time complexity of **Agglo Algorithm**. The top-down version is, therefore, too expensive to be of real use.

Clustering: concept formation. Conceptual clustering was originated from the need for describing human categorization. As was noticed, numerical taxonomy or Nearest Neighbor and those simply relying on distance measures cannot serve to satisfy the need (Michalski and Stepp, 1983). COBWEB (Fisher, 1987)

is one such system that does not use a distance measure, but uses a heuristic measure called *category utility* (CU) suggested in (Gluck and Corter, 1985) for evaluating the quality of a categorization and a classification:

- CU attempts to maximize both the probability that two objects in the same category have values in common and the probability that objects in different categories will have different property values.
- CU is defined as:

$$\sum_k \sum_i \sum_j p(f_i = v_{ij}) p(f_i = v_{ij}|c_k) p(c_k|f_i = v_{ij})$$

This sum is taken across all categories, c_k, all features, f_i, and all feature values, v_{ij}.
$p(f_i = v_{ij}|c_k)$ (*predictability*) is the probability that an object has value v_{ij} for feature f_i given that the object belongs to category c_k. The higher this probability, the more likely two objects in a category share the same feature values.
$p(c_k|f_i = v_{ij})$ (*predictiveness*) is the probability with which an object belongs to category c_k given that it has value v_{ij} for feature f_i. The greater this probability, the less likely objects not in this category will have those feature values.
$p(f_i = v_{ij})$ serves as a weight, assuring that frequently occurring feature values will exert a stronger influence on the evaluation.

By combining these values, high CU measures indicate a high likelihood that objects in the same category will share properties, while decreasing the likelihood that objects in the different categories having properties in common.

The clustering is done in a top-down fashion: (1) start with the root (in the beginning, it contains only one instance), for each subsequent instance, do (2) find the best choice for it among (a) creating a new child cluster (under the current cluster) to accommodate the instance, (b) hosting the instance in an existing cluster, (c) merging the best two child clusters to accommodate the instance, and (d) splitting the best cluster (upgrading its child nodes) to find a cluster that can accommodate the instance. This is summarized in Table 6.5. Operations **merge** and **split** are physically performed only when they are really needed. In calculating C_1 to C_5, the concept hierarchy is not tampered with in any way.

When we analyzed pros and cons for the directions (top-down vs. bottom-up) of forming numerical taxonomy, we argued that bottom-up is more appropriate. However, in **ConForm**, we adopt an opposite direction for concept formation. This can be done because the algorithm is incremental, and search

Table 6.5. An algorithm for concept formation (ConForm).

ConForm Algorithm
Input: CU - category utility
D - data
initialize: root = 1st instance in D
ConForm(R, I) /*R - root, I - subsequent instance*/
if R is a leaf
make R a parent node of R_{old} and I
else
$C_1 = CU$(newChild for I)
$C_2 =$ **best**$(CU$(extantChild$_i$ hosting $I))$ /*child$_j$*/
$C_3 =$ **2ndBest**$(CU$(extantChild$_i$ hosting $I))$ /*child$_k$*/
$C_4 = CU$(merge2bestChildren to host I)
/*Two best children are C_j, C_k*/
$C_5 = CU$(splitBestChild (C_j) to host I) /*child$_s$ */
if C_1 is best
let R host I /*Place I by itself in the new cluster*/
if C_2 is best
ConForm(child$_j$, I) /*Place I in cluster child$_j$*/
if C_4 is best
child$_m$ = **merge**(C_j, C_k)
ConForm(child$_m$, I)
if C_5 is best
split child$_s$
ConForm(R, I)
Output: a concept hierarchy

is mainly localized. If two instances are similar to each other, they will be eventually clustered together. On the other hand, the bottom-up choice is not that intuitive from an incremental viewpoint. However, **ConForm** may be sensitive to the order in which the instances are presented. In (McKusick and Langley, 1991, Liu and Wen, 1994), they analyzed various situations and proposed some methods that can produce well-formed concept trees with local restructuring operators.

In (Michalski and Stepp, 1983), they offered a comparison between numerical taxonomy and conceptual clustering. In (Fisher and Langley, 1985), they gave a brief survey of various approaches to conceptual clustering. CLASSIT proposed in (Gennari et al., 1989) is a system that extends earlier conceptual clustering systems including COBWEB (Fisher, 1987) and UNIMEM (Lebowitz, 1987), and can learn concepts from continuous values. Combining continuous features with symbolic ones is also discussed in their paper. AutoClass is a Bayesian approach to probabilistic clustering (Cheeseman et al., 1988).

Purpose. No matter how many types of clustering algorithms exist, from the viewpoint of feature selection, the task is to remove redundant and irrelevant features by keeping the features used to describe clusters. In other words, a needed feature is a relevant feature.

A naive and direct implementation of this idea is (1) to find properties (or feature-values) possessed by all members of a cluster and only by the members of the cluster, (2) to form a feature set for each cluster by removing those features not used in defining the properties, and (3) to union all feature sets of all clusters to obtain a super set of selected features. There is, however, a danger of taking the super set as the minimum subset needed for describing the data. This is because in presence of outliers, there may be many dangling clusters with few instances as in the results formed by NearN algorithms. In the worst case, one may select all features. The key to successful application of NearN algorithms to feature selection is to remove the outliers. One can, for example, examine the clusters and consider those with few instances as unnecessary clusters caused by outliers. As for hierarchical clustering, one need to determine a proper level of basic taxonomy or concepts. That is, the level should not be too general nor too specific. Let's consider the two extreme cases: the most general - the whole data set is a cluster, and the most specific - each instance is a cluster. In either case, all features could be selected. In order to choose a subset of features, it is necessary to determine a proper level in a taxonomy or a hierarchy. Psychologists (Rosch, 1978) have demonstrated the existence of base-level categories. For example, "chair" is more basic than either its generalization, such as "furniture," or its specialization, such as "office chair". We can borrow this idea to find the proper level for feature selection.

All in all, it is a tough nut to crack. In a general setting, we would recommend the use of NearN algorithms for selecting features.

After suitable clusters are found, one can further refine sets of features for each cluster. Intuitively, some features may play pivotal roles, some less important. If we consider all clusters in a sense of global data analysis (Vilalta et al., 1997), we may be able to remove more features. C4.5rules (Quinlan, 1993) is a good example of using global data analysis to remove redundant rule conditions: for each rule R, a new rule R' is formed by removing condition C_i, both rules are globally evaluated and only one is retained according to a pessimistic estimation of their corresponding error rates. In our context where each cluster is considered as a class, we can have one less feature if the feature is removed and instances in this cluster can still be differentiated from instances in other clusters.

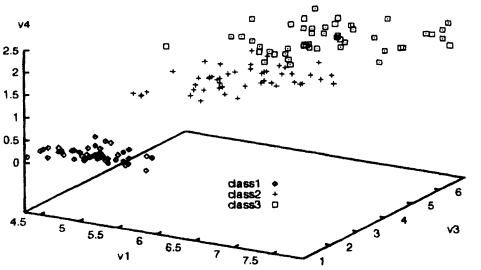
Summary. The major difference between conceptual clustering and numerical taxonomy lies in that the former performs clustering not on the basis of some distance measures of objet similarity, but on the basis of some concept membership such as category utility. Concepts are formed by conceptual clustering. This difference may not be significant for our purpose of clustering. One may use one of them to form clusters and pick out features useful depending on the need and convenience.

Numerical taxonomy or concept hierarchy is not of our interest in feature selection. Various approaches to clustering result in different clusters. A clustering technique is just a means to the end of feature selection. All three introduced clustering approaches have their disadvantages such as defining threshold values or a proper level in a hierarchy. By all means, clustering is not the only way that can handle unsupervised feature selection. In the next, we introduce another method of feature selection dealing with unsupervised data.

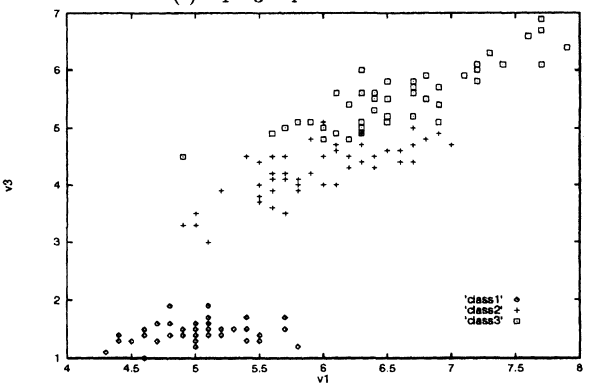
6.4.2 Unsupervised feature selection: entropy

The features selected by clustering methods are not ranked. Whichever features needed in describing a cluster are necessary and should be chosen. If one is interested in obtaining a ranked list of features, other approaches have to be sought. We describe one that relies on an entropy measure. This approach (Dash et al., 1997) is based on the observation that removing an irrelevant and/or redundant feature from the feature set may not change the underlying concept of the data, but not so otherwise. By visualizing the Iris data (75 instances) in 3-dimensional and 2-dimensional spaces, one can see the difference between removing an irrelevant feature and a relevant one. In Figures 6.6a(i) and 6.6b(i), we start with two ranked lists of features (v_1, v_3, v_4) and

(v_3, v_4, v_1), after removing the last feature in the two lists, we obtain the projections of data in two figures (Figures 6.6a(ii) and 6.6b(ii)), respectively. We can see that Figure 6.6b(ii) exhibits more distinct clusters than Figure 6.6a(ii). Hence, we conclude that feature v_4 is more relevant than feature v_1. We need to devise a measure that can tell us the difference we just experienced.



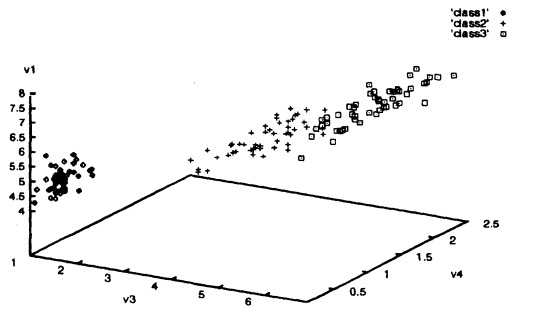
(i) v_1-v_3-v_4 dimension



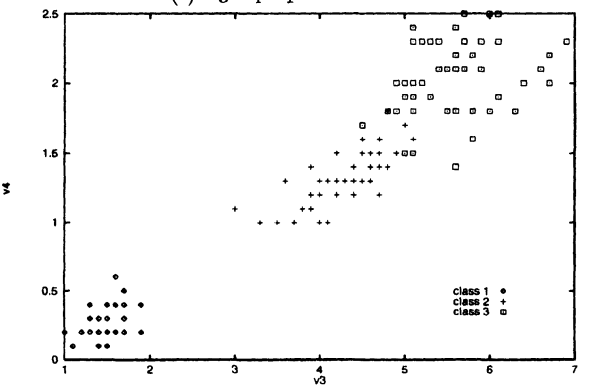
(ii) v_1-v_3 dimension

Figure 6.6a. The Iris data in 3-dimensional and 2-dimensional spaces.

A measure. There is a need for a measure that evaluates the distinctness among the underlying clusters given a subset of features. The data has orderly configurations if it has distinct clusters, and has disorderly or chaotic configurations otherwise. From entropy theory (Fast, 1962), we know that entropy (or



(i) v_3-v_4-v_1 dimension



(ii) v_3-v_4 dimension

Figure 6.6b. The Iris data in 3-dimensional and 2-dimensional spaces.

probability) is less for orderly configurations, and more for disorderly configurations for the simple reason that there are few orderly configurations compared to disorderly configurations. Entropy should be very low (close to 0.0) for very close or very distant pairs of instances, and very high (close to 1.0) for those instances separated by a distance close to the mean distance. We use a similarity measure (S) that is based on distance, and assumes a very small value (close to 0.0) for very close pairs of instances that probably fall to a single cluster, and a very large value (close to 1.0) for very distant pairs of instances that probably fall into different clusters. The idea is to remove as many features as possible but still maintain the level of distinction as if no features were removed. For

two instances, the entropy measure is $E = -S\log_2 S - (1-S)\log_2(1-S)$ which assumes the maximum value of 1.0 for $S = 0.5$, and the minimum value of 0.0 for $S = 0.0$ and $S = 1.0$ (Klir and Folger, 1988).

For a data set of N instances the entropy measure is given as:

$$\mathbf{E} = -\sum_{i=1}^{N-1}\sum_{j=i+1}^{N}(S_{ij}\times\log S_{ij} + (1-S_{ij})\times\log(1-S_{ij})) \qquad (6.4)$$

where S_{ij} is the similarity value between the instances x_i and x_j.

When all features are *numeric* or *ordinal*, the similarity value of two instances is:

$$S_{ij} = e^{-\alpha\times D_{ij}} \qquad (6.5)$$

where D_{ij} is the distance between the instances x_i and x_j, and α is a parameter. If we plot similarity against distance, then the curve will have a bigger curvature for a larger α. The experiments with various values for α suggest that it should be robust for all kinds of data sets, and not just for certain data sets. In this work, α is calculated automatically by assigning the value of similarity (i.e., 0.5) in Equation 6.5 for the maximum entropy (Equation 6.4). Mathematically, α can be obtained as:

$$\alpha = \frac{-\ln 0.5}{\bar{D}}$$

where $\bar{D}$ is the average distance among the instances in a hyper-space. Hence, α is determined by the data. This, in fact, produces good results for the tested data sets.

Euclidean distance measure is used to calculate the distance D_{ij} between any two instances x_i and x_j. In a multi-dimensional space, it is defined as:

$$D_{ij} = [\sum_{k=1}^{U}(\frac{x_{ik} - x_{jk}}{max_k - min_k})^2]^{1/2}$$

where max_k and min_k are the maximum and minimum values for the k^{th} dimension. The interval in the k^{th} dimension (i.e., k^{th} feature) is normalized by dividing it by the maximum interval $max_k - min_k$ before calculating the distance.

Similarity for *nominal* variables is measured using Hamming distance. The similarity value of any two instances is given as:

$$S_{ij} = \frac{\sum_{k=1}^{U}|x_{ik} = x_{jk}|}{M} \qquad (6.6)$$

where $|x_{ik} = x_{jk}|$ is 1 if x_{ik} equals x_{jk} and 0 otherwise, and M is the number of features in the subset under consideration.

For mixed data (i.e., with both numeric and nominal features), we can discretize numeric values first before applying this measure (some feature discretization algorithms can be found in Section 6.3 of this chapter.)

Ranking features. With the similarity measure, we can evaluate whether a feature is important following a sequential backward selection algorithm (refer to Chapter 3 for the details of such algorithms and related topics):

1. start with the full set F,
2. remove one feature f from F in turn and obtain a subset F_f,
3. find the difference between entropy F and entropy F_f,
4. let f_k be the feature such that the difference between F and F_{f_k} is minimum,
5. update $F = F - \{f_k\}$, and
6. repeat (2) - (5) until there is only one feature in F.

Iris data revisited. The features of Iris data are ranked as x_3, x_4, x_1, and x_2. In order to see whether the order of the sequence is reasonable, we run C4.5 on the data with $\{x_3\}$, $\{x_3, x_4\}$, $\{x_3, x_4, x_1\}$, and $\{x_3, x_4, x_1, x_2\}$. If the order is sensible, we should expect that after a certain point, assuming that there are irrelevant features, error rates should increase (the irrelevant features hinder effective learning) or at least not decrease. As shown in Figure 6.7, using features x_3, x_4, the decision trees built by C4.5 achieve the lowest average error rate of 10-fold cross validation.

Summary. Ranking features using the concept of entropy does not rely on class information. In this sense, it is similar to the clustering idea for unsupervised feature selection. However, the focus here is no more on clusters or concept hierarchies. Instead, we rank features by gradually removing least important features in maintaining the orderly configurations. So obtained is a ranked list of features. Direct comparison between this method and the clustering method has not been done yet for unsupervised clustering. It is clear that the computational complexities of the two methods are similar. The issues of further experimentation are mainly about (1) whether they select features similarly and (2) under what circumstances, one method should be preferred to the other.

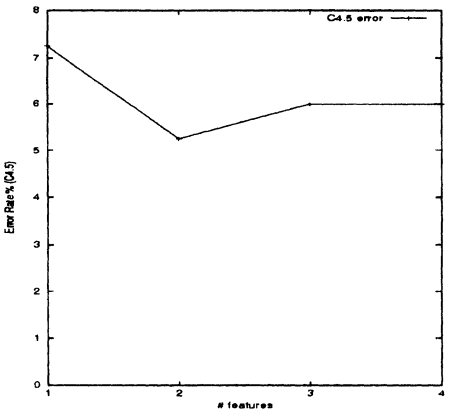


Figure 6.7. The effect on error rates: including features from the most important to the least important x_3, x_4, x_1, x_2.

6.5 CONCLUSIONS

In this chapter, we discuss topics related to feature selection, such as feature extraction, feature construction, and feature discretization. As we now know, feature extraction produces new features that are transformed from original features. The number of extracted features can be determined according to an application, but we do need original features in the transformation. Feature construction is more concerned about enhancing the representational power of the original features by generating compound features in overcoming problems such as fragmentation and replication. The compound features are composed of the original features. It is not surprising that some original features can be removed because (1) they are not used in the compound features, and (2) they are not important any more after the compound features are constructed. Feature discretization is a potent weapon for dimensionality reduction. It reduces the number of values of a feature as well as the number of features (in the case that a feature contains only one value, the feature can be removed). Each topic itself has been extensively studied as a subfield. Collected in (Liu and Motoda, 1998) are some latest updates of these topics' development and their applications to real-world problems.

Unsupervised feature selection is a relatively less developed subfield comparing to subset selection, feature extraction, construction and discretization. More research is expected because of the need for dimensionality reduction in

unsupervised data. The two methods described here show, by example, what and how feature selection can be done with unsupervised data. Another important data mining task that deals with unsupervised data is mining association rules. It is difficult to select features for a classification task. It becomes even more so to remove redundant features in the context of association rules. This is because there is no clear boundary between redundant features and associated features.

References

Aha, D. (1997). Editorial. *Artificial Intelligence Review*, 11.

Bloedorn, E. and Michalski, R. (1998). *Data-Driven Constructive Induction: A Methodology and Its Applications*, pages 51 – 68. In (Liu and Motoda, 1998).

Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and Regression Trees*. Wadsworth & Brooks/Cole Advanced Books & Software.

Brodley, C. and Utgoff, P. (1995). Multivariate decision trees. *Machine Learning*, 19:45–77.

Catlett, J. (1991). On changing continuous attributes into ordered discrete attributes. In *European Working Session on Learning*.

Cerquides, J. and de Mantaras, R. L. (1997). Proposal and empirical comparison of a parallelizable distance-based discretization method. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 139 – 142.

Cheeseman, P., Kelly, J., Self, M., Stutz, J., Taylor, W., and Freeman, D. (1988). AUTOCLASS: A Bayesian classification systems. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 54–64. Morgan Kaufmann.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3:261–283.

Cost, S. and Salzberg, S. (1993). A weighted nearest neighbor algorithm for learning with symbolic features. *Machine Learning*, 10:57–78.

Dasarathy, B. (1991). *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*. IEEE Computer Society Press.

Dash, M., Liu, H., and Yao, J. (1997). Dimensionality reduction of unsupervised data. In *Proceedings of the Ninth IEEE International Conference on Tools with AI (ICTAI'97)*, pages 532–539. IEEE Computer Society.

Donoho, S. and Rendell, L. (1996). Constructive induction using fragmentary knowledge. In Saitta, L., editor, *Proceedings of International Conference on Machine Learning (ICML-96)*, pages 113–121. Morgan Kaufmann Publishers.

Dougherty, J., Kohavi, R., and Sahami, M. (1995). Supervised and unsupervised discretization of continuous features. In *Proceedings of the Twelfth International Conference on Machine Learning*. Morgan Kaufmann, Los Altos, CA.

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

Fast, J. (1962). *Entropy : the significance of the concept of entropy and its applications in science and technology*, chapter 2: The Statistical Significance of the Entropy Concept. Eindhoven : Philips Technical Library.

Fayyad, U. and Irani, K. (1993). Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann Publishers, Inc.

Fisher, D. (1987). Knowledge acquisition via incremental conceptual clustering. *Machine Learning*, 2:139–172.

Fisher, D. and Langley, P. (1985). Approaches to conceptual clustering. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, pages 691–97.

Fisher, R. (1936). The use of multiple measurements in taxonomic problems. *Ann. Eugenics*, 7(2):179–188.

Friedman, J., Kohavi, R., and Yun, Y. (1996). Lazy decision trees. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, pages 717–724.

Gennari, J., Langley, P., and Fisher, D. (1989). Models of incremental concept formation. *Artificial Intelligence*, 40:11–61.

Gluck, M. and Corter, J. (1985). Information, uncertainty, and the utility of categories. In *Proceedings of the Seventh Annual Conference of the Cognitive Science Society*, pages 283–87. Lawrence Erlbaum, Irvine, CA.

Hagan, M., Demuth, H., and Beale, M. (1996). *Neural Network Design*. PWS Publishing Company.

Ho, K. and Scott, P. (1997). Zeta: A global method for discretization of continuous variables. In *Proceedings of The Third International Conference of Knowledge Discovery and Data Mining*, pages 191–194. Newport Beach, CA.

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

Jun, B., Kim, C., Song, H., and Kim, J. (1997). A new criterion in selection and discretization of attributes for the generation of decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(12):1371–1375.

Kerber, R. (1992). ChiMerge: Discretization of numeric attributes. In *AAAI-92, Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 123–128. AAAI Press/The MIT Press.

Klir, G. and Folger, T. (1988). *Fuzzy Sets, Uncertainty, and Information*, chapter 5: Uncertainty and Information. Prentice-Hall International Editions.

Kohonen, T. (1984). *Self-organization and Associative Memory*. Springer-Verlag, Berlin.

Kramer, S. (1994). CN2-MCI: A two-step method for constructive induction. In *Proceedings of the Workshop on Constructive Induction and Change of Representation, International Conference on Machine Learning (ML-94/COLT-94)*.

Langley, P., Simon, H., Bradshaw, G., and Zytkow, J. (1987). *Scientific Discovery: Computational Explorations of the Creative Processes*. Cambridge, MA: The MIT Press.

Lebowitz, M. (1987). Experiments with incremental concept formation. *Machine Learning*, 1:103–138.

Liu, H. and Motoda, H., editors (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Liu, H. and Setiono, R. (1995). Chi2: Feature selection and discretization of numeric attributes. In *Proceedings of the Seventh IEEE International Conference on Tools with Artificial Intelligence*, pages 388–391.

Liu, H. and Setiono, R. (1997). Feature selection via discretization. *IEEE Trans on Knoweldge and Data Engineering*, 9(4):642–645.

Liu, H. and Wen, W. X. (1994). Joint concept formation. *Journal of Knowledge Acquisition*, 6:75–87.

Matheus, C. and Rendell, L. (1989). Constructive induction on decision trees. In *Proceedings of International Joint Conference on AI*, pages 645–650.

McKusick, K. and Langley, P. (1991). Constrains on tree structure in concept formation. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, pages 810–816.

Michalski, R. and Stepp, R. (1983). Automated construction of classifications: Conceptual clustering versus numerical taxonomy. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 5(4).

Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine Learning, Neural and Statistical Classification*. Ellis Horwood Series in Artificial Intelligence.

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

Muggleton, S. (1987). Duce, an Oracle-based approach to constructive induction. In *Proceedings of International Joint Conference on AI*, pages 287–292. Morgan Kaufmann.

Pagallo, G. (1989). Learning DNF by decision trees. In *Proceedings of International Joint Conference on AI*, pages 639–644. Morgan Kaufmann.

Pagallo, G. and Haussler, D. (1990). Boolean feature discovery in empirical learning. *Machine Learning*, 5:71–99.

Pettis, K., Bailey, T., Jain, A., and Dubes, R. (1979). An intrinsic dimensionality estimator from near-neighbor information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1:25–37.

Pettofrezzo, A. (1966). *Matrices and Transformations*. Dover Publications, Inc. New York.

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

Quinlan, J. (1988). Decision trees and multi-values attributes. In J.E., H., Michie, D., and J., R., editors, *Machine Intelligence*, volume 11. Oxford University Press.

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

Rendell, L. (1985). Substantial constructive induction using layered information compression: Tractable feature formation in search. In *Proceedings of International Joint Conference on AI*, pages 650–658. Morgan Kaufmann.

Rosch, E. (1978). Principles of categorization. In Rosch, E. and Lloyd, B., editors, *Cognition and Categorization*. Erlbaum, N.J.

Setiono, R. and Liu, H. (1998). *Feature Extraction via Neural Networks*, pages 191–204. In (Liu and Motoda, 1998).

Shlimmer, J. (1987). Learning and representation change. In *Proceedings of AAAI*, pages 511–515. Morgan Kaufmann.

Vilalta, R., Blix, G., and Rendell, L. (1997). Global data analysis and the fragmentation problem in decision tree induction. In van Someren, M. and Widmer, G., editors, *Machine Learning: ECML-97*, pages 312–326. Springer-Verlag.

Wnek, J. and Michalski, R. (1994). Hypothesis-driven constructive induction in AQ17-HCI: A method and experiments. *Machine Learning*, 14.

Wyse, N., Dubes, R., and Jain, A. (1980). A critical evaluation of intrinsic dimensionality algorithms. In Gelsema, E. and Kanal, L., editors, *Pattern Recognition in Practice*, pages 415–425. Morgan Kaufmann Publishers, Inc.

Notes

1. Most statistical software packages (e.g., SAS `http://www.sas.com/SASHome.html`) include principal component analysis.

2. Note that the input data was originally prepared to show the self-organizing capabilities of Kohonen net.

3. Rejecting H_0, we accept the alternative hypothesis H_1 that the feature and the class are not independent.

4. Given $0 < conf \leq 1$ and the degree of freedom, θ is uniquely determined by the χ^2 distribution (Murdoch and Barnes, 1993). For example, if the degree of freedom is 1, and $conf = .95$, $\theta = 3.841$.

5. It's more likely that this feature will be chosen since it has many values (Quinlan, 1988, Mingers, 1989) when information gain is used.

6. Here we leave the option open and let the user decide which one to take.

7 LESS IS MORE

Now we almost arrive at the end of our journey of various methods for feature manipulation. It is time for us to reflect the underlying principle for all the ideas and motivations behind so many methods. *"Less is more"* is the philosophy manifested throughout this book. We are not only facing a grim reality of ever growing amounts of data but also constrained by limitations of tools and representations. Nevertheless, we are determined to continue our pursuit of finding regularities from disordered data and discovering knowledge hidden in massive data. With more data accumulated, we hope to discover many more valuables from the data. *"Less is more"* says that in order to get more useful findings from the data, we need to first make the data less by removing its irrelevant parts.

With less dimensionality, we can overcome some limitations imposed by the best technologies: we can handle data that could not be dealt with before dimensionality reduction; we can discover knowledge that would have been lost in massive data. Such examples are abundant. With fewer features, we can focus better on the relevant data, learn more sensible knowledge from the data; with simpler learned representations, we can understand better what has been learned from the data.

Feature selection methods help reduce data for mining or learning tasks and enable those mining algorithms, which were unable to mine, to mine. By reducing data, feature selection amplifies the capabilities of mining tools (present or future), narrows the tools gap, extends the life span of existing tools so as to gain time for us to develop new tools.

7.1 A LOOK BACK

The techniques introduced in this book are about making data less or simpler while maximally keeping the relevant information. In effect, this type of techniques stems from the human's never ending pursuit to understand nature and to explore the unknown, which is also the essential motivation of machine learning and data mining algorithms. While machine learning algorithms such as classification and clustering attempt to summarize data into a concise form that can be understood and reused by humans, feature selection and related techniques try to facilitate the learning and to make it more efficient and effective. The difference between feature selection and machine learning is that the change to the data caused by feature selection is not as drastic as that by machine learning. The data is reduced only in scale, the representation remains. As in an example of decision tree induction, we obtain a decision tree which is a hierarchical structure, it is in a completely different representation from that of the data. Therefore, it is expected that feature selection techniques are usually less time consuming and less computation intensive than the techniques of machine learning and data mining. This explains why feature selection methods can help when the methods of machine learning and data mining falter.

Features are important basic units with which data can be described and on which data mining algorithms can work. In the beginning of the journey, we relied on a classification model to explain the problem of feature selection. We have gone through a spiral process: first top-down, next bottom-up, then top-down again. By considering each perspective, we arrived at a unified model; by studying each aspect, we were able to design our very own feature selection method for an application. Nevertheless, we were not satisfied by building special feature selection methods. We moved on to scientifically evaluate different methods and came up with some guidelines in applying feature selection methods. The last point is particularly pertinent to real-world applications. It is the balanced solution that we look for, as we understand that no single feature method is superior over others. The guidelines can make easier our task of applying which methods under various circumstances. Properly used in practice, the rich repertoire of feature selection methods holds the promise of an enabling technology that could reduce dimensionality, focus on the features

related to the target concept, avoid data dredging, and trace the growth and change of data.

Feature selection is just part of a larger endeavor to simplify data and reduce its dimensionality (Liu and Motoda, 1998). In addition to feature selection, we have studied other techniques that allow us to enhance representation power and reduce dimensionality. Feature extraction is about ways of finding a small number of new features that describe the data as if the original features are used. Feature construction achieves the goal of dimensionality reduction by using more powerful features - compound features. Feature discretization realizes the objective of dimensionality reduction by transforming continuous features to discrete ones. Features exist with data. Without data, features cannot be materialized; without features, data is difficult to describe. Techniques of feature manipulation can also vary with different data types (supervised vs. unsupervised).

In the tasks of data mining, data is the source and knowledge is the destination. A data flow diagram is shown in Figure 7.1 to draw a picture about the relations between data, techniques, data mining tools, and knowledge. Data is divided into supervised and unsupervised. Feature manipulation techniques are applied to reducing data. Dimensionally reduced data is presented to data mining and machine learning algorithms. At the end, rules are induced and knowledge is discovered. Two basic paradigms (classification and clustering[1]) are also indicated for supervised and unsupervised data in the figure.

As we approach to a comfortable stop of the journey, we cannot presume the cease of our pursuit. Yes, much work has been done, yet more exciting challenges and opportunities await us ahead.

7.2 A GLANCE AHEAD

Data mining is a fruitful area of research with an abundant wealth of practical applications. Technology has made it so easy to gather data that more and more companies and organizations find themselves inundated with data and are eager to extract knowledge from it. Data mining is a complex and delicate process that requires more than getting software and starting to run some programs. One of the biggest difficulties in implementing any data mining strategy is trying to extract clean and reliable data before we do the mining. Therefore we should expect the increasing demand for data cleaning and dimensionality reduction. Feature selection, discretization and transformation have been proven effective with good results. As more specialized tools are developed and deployed, we will begin to learn what the bigger issues are in developing powerful feature selection tools that can be used by average users. A meta feature selection algorithm is one example of what we can try. In the end of Chapter

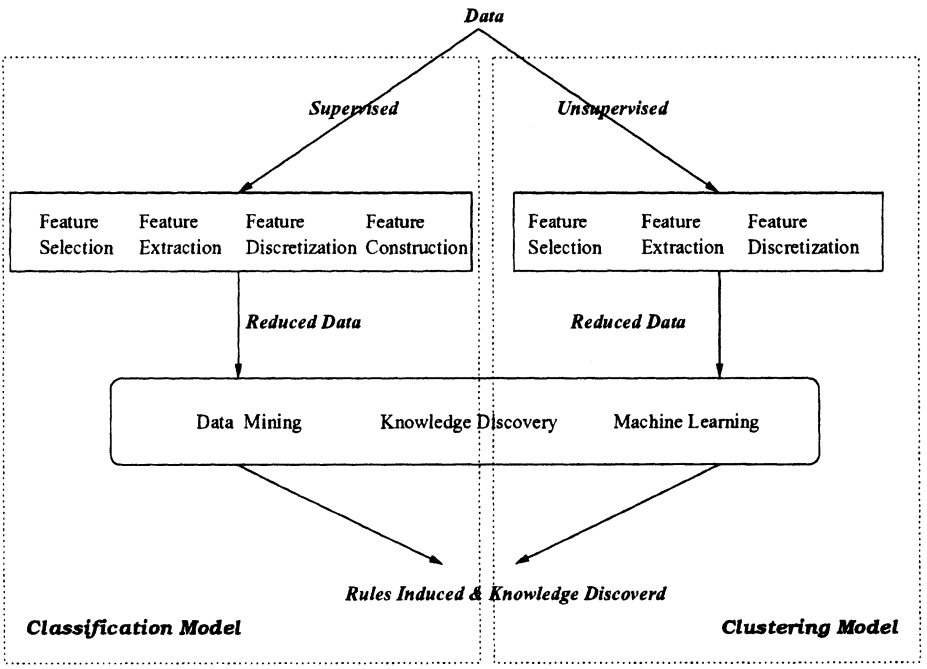


Figure 7.1. A data flow diagram: relations between data, feature manipulation techniques and knowledge discovery.

5, we have made some attempt in this direction. We hope that the endeavors outlined in this monograph will help further the efforts.

Building new tools requires more research and deeper understanding in the field. Plentiful research problems need to be addressed in the context of data mining. Listed here are some issues related to feature selection.

- *Search* is still a central issue. In general, the more we search, the better a solution. In reality, however, we have to compromise the quality of the solution with the time of search we are given. One issue is how to search maximally within the time constraint. Research in anytime algorithms has shown some promising results in this regard. The other issue is how to search intelligently. That is, if we can avoid blind search or focus search in the right direction, we do not have to reduce the intensity of search so as not to compromise the quality. In other words, search is directed. This leads us to the study of using *a priori* knowledge to guide search.

- *Focusing* and *partitioning* are two of the techniques that we can rely on to maintain the search intensity within the time constraint. Focusing narrows down the search space and avoids searching in "futile" areas. To identify promising areas for search, we need educated guesses based on our knowledge of application domains and skillful use of various methods. The identification problem is not an easy nut to crack.

  Partitioning is to decompose the search space into independent and manageable sub-spaces (partitions). With the aid of distributed computing, search can be intensified in each partition. The final solution emerges from the solutions found in partitions. On one hand, partitioning does not require educated guesses to focus on particular partitions, yet achieve what focusing can, as we know that educated guesses may mislead search sometimes. On the other hand, we face the decomposition and unification problems. It may not be trivial to decompose a problem into independent sub-problems. Likewise, we may encounter difficulties in combining sub-solutions into a final, overall solution.

- *Concept drift* is an often occurred problem in data mining. Basically, what was relevant in the past may not be so at the present. In the context of feature selection, for example, some irrelevant features may become relevant, or vice versa, as things constantly and gradually change. How to capture this concept drift is critical for feature selection as well as for data mining. The simplest way of solving the problem of concept drift is to redo feature selection regularly. Nevertheless, with the huge sized data, the redoing of feature selection is not as simple as it looks. If we do it too often, no concept drift may be noticed and resources are then wasted. If we seldom do it, damages may have been done because of the ignorance of concept drift.

- *Scaling* is concerned about the size of instances P in a data set (Chaudhuri, 1998). One generic way to scale the data over large data set is to use sampling. Whether sampling is appropriate for a class of data analysis techniques, and even when appropriate, how much to sample and how, will become important questions.

  Instance selection is another way of handling the problem of scaling. The underlying idea is similar to that of feature selection - to select relevant instances instead of relevant features. In (Blum and Langley, 1997), they provide a survey of example selection methods such as windowing (Quinlan, 1986), peepholing (Catlett, 1992), query-by-committee (Seung et al., 1992), and so on.

Dimensionality reduction has also been an important issue studied by researchers in many fields. A recent report (Barbara et al., 1997) introduces and

evaluates, from a database point of view, many techniques available for data reduction[2]: Singular Value Decomposition, Wavelets, Regression, Log-Linear Models, Histograms, Clustering Techniques, Index Trees, and Sampling. These methods are originated from various fields, and now considered and used to attack the same problem - dimensionality reduction.

What is used in the first phase can also be used in the last. We rely on selection techniques to clean data and reduce data in the pre-processing before data mining. We can also apply selection techniques to filtering the results returned from data mining in the post-processing of the KDD process. Many can be mined out from data. However, it does not mean that all mined are nuggets - interesting, relevant and useful. It all depends on the user's needs, the application at hand, and existing knowledge. Combining feature and instance selection techniques, we look forward to techniques that remove uninteresting, irrelevant, or redundant patterns according to the user's intentions of data mining.

References

Barbara, D., DuMouchel, W., Faloutsos, C., Haas, P., Hellerstein, J., Ioannidis, Y., Jagadish, H., Johnson, T., Ng, R., Poosala, V., Ross, K., and Sevcik, K. (1997). The New Jersey data reduction report. *Bulletin of the Technical Committee on Data Engineering*, 20(4).

Blum, A. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271.

Catlett, J. (1992). Peepholing: Choosing attributes efficiently for megainduction. In *Proceedings of the Ninth International Conference on Machine Learning*, pages 49–54. Morgan Kaufman.

Chaudhuri, S. (1998). Data mining and database systems: Where is the intersection? *Bulletin of the Technical Committee on Data Engineering*, 21(1).

Liu, H. and Motoda, H., editors (1998). *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers.

Murdoch, J. and Barnes, J. (1993). *Statistical Tables for Science, Engineering, Management and Business Studies*. The Macmillan Press, 3rd edition.

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

Seung, H., Opper, M., and Sompolinsky, H. (1992). Query by committee. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pages 287–294. ACM Press, New York.

Notes

1. Mining association rules is something between the two paradigms, which works on unsupervised data but applies statistical techniques in a similar way as classification does.

2. In the report, authors assume data with $N \leq 10$. Obviously, they are more concerned about the largeness P - the number of instances from the database stand.

Appendix A
Data Mining and Knowledge Discovery Sources

We provide here some pointers related to machine learning (ML), data mining and knowledge discovery (KD) for the reader's convenience. For periodically updated, added or deleted links, the reader should visit the web page: `http://www.iscs.nus.edu.sg/~liuh/Fsbook`.

A.1 WEB SITE LINKS

Legends:
B - Bibliographies; D - Datasets; I - Information on events or conferences; L - Links to KDD sites; N - Newsletters or mailing List; S - Softwares or programs; P - Papers.

Name	Site and Contact	Resource
The KD Mine	http://www.kdnuggets.com/index.html G. Piatetsky-Shapiro gps@kdnuggets.com	_DILNSP
A site for comprehensive information in KDD that contains links to E-newsletter, siftware, datasets, publications and other related sites.		
The Data Mine	http://www.cs.bham.ac.uk/~anp/ TheDataMine.html A. Parke anp@cs.bham.ac.uk	B_IL_SP
A major server of KDD and OLAP information that provides a repository of papers, bibliographies and links to conferences, softwares and web sites.		
UCI ML Site	http://www.ics.uci.edu/~mlearn/ UC Irvine jmuramat@ics.uci.edu	_DILNSP
A comprehensive Machine Learning site. Popular for its large repository of standard data sets and machine learning programs for experimental evaluations.		
MLnet ML Archive at GMD	http://www.gmd.de/ml-archive ML Group of GMD ml-archive@gmd.de	BDIL_SP
It offers a wide collection of machine learning information, data sets, programs and links to other machine learning resources.		

Name	Site and Contact	Resource
CMU A.I. Repository	http://www.cs.cmu.edu/Groups/AI/html/repository.html CMU AI.Repository@cs.cmu.edu	B_IL_SP
It collects files, programs and publications of interest to A.I. researchers, educators, students and practitioners. There are also web sites links to LISP, PROLOG & SCHEME.		
Machine Learning Online	http://mlis.www.wkap.nl/mach/ml_links.htm J. Schlimmer Schlimme@eecs.wsu.edu	BDILNSP
A comprehensive list of links to ML resources over the WWW. The links are sorted into various categories, like software, bibliographies, data sets, conferences & people, etc.		
a. ML Folks b. David Aha's ML & CBR Resources	http://www.aic.nrl.navy.mil/~aha/people.html http://www.aic.nrl.navy.mil/~aha/research/machine-learning.html D. Aha aha@aic.nrl.navy.mil	B_ILNSP
a. The Navy Center for Applied Research in A.I. (NCARAI). b. Aha's page provides information, programs, papers, bibliographies, and researchers in different areas of ML and CBR (case based reasoning).		
Society for A.I. and Statistics	http://www.vuse.vanderbilt.edu/~dfisher/ai-stats/society.html D. Fisher dfisher@vuse.vanderbilt.edu	B_ILN_P
A list of links to other A.I. and statistics related sites. It also contains publications & research topics in A.I. and statistics, including the AI-Stats E-mailing list.		
WWW Virtual Library on Statistics	http://www.stat.ufl.edu/vlib/statistics.html Mike Conlon mconlon@stat.ufl.edu	BDILNSP
An up-to-date online resource to statistical software, datasets and links.		
PC Webopaedia Data Mining Page	http://www.pcwebopedia.com/data_mining.htm Sandy Bay Software, Inc. webmaster@sandybay.com	__IL___
It provides definitions and contains news, articles and links to useful sites in data mining applications.		

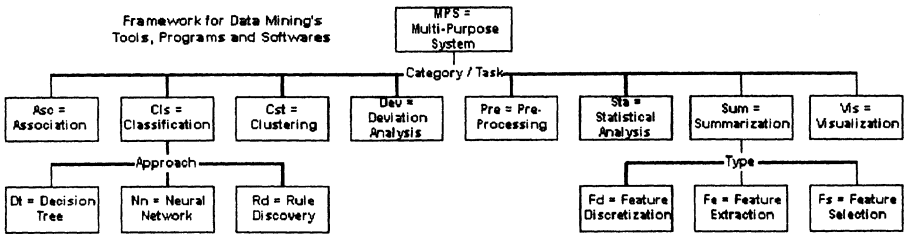
Name	Site and Contact	Resource
Evaluation of Intelligent Systems	http://eksl-www.cs.umass.edu/eis/ D. Jensen jensen@cs.umass.edu	_DILN__
An online resource that provides "one-stop shopping" for managers, system-builders, researchers, and users who wish to study the empirical behavior of information systems.		
Univ of Portsmouth ML Archive	http://www.sis.port.ac.uk/ml-algs/ S. Thompson sgt@sis.port.ac.uk	_D___S_
It includes various LISP interfaced programs of popular machine learning algorithms, like AQ, Cobweb, CN2, FOIL, ID3, C4.5, kNN, etc. and data sets in LISP format.		
IBM Almaden Research Center - Quest DM project	http://www.almaden.ibm.com/cs/quest/ P. Greissl greissl@vnet.ibm.com	_D___SP
Research in data mining using IBM Intelligent Miner. It contains Synthetic Data Generation Codes for Associations, Sequential Patterns and Classification.		
The WWW Virtual Library page on A.I.	http://www.cs.reading.ac.uk/people/dwc/ai.html J. Bowen J.P.Bowen@reading.ac.uk	_DILNSP
An online resource to A.I. programs, software, datasets, bibliographies, links in WWW.		
ML group of LIACC	http://www.ncc.up.pt/liacc/ML/ P.Brazdil & J.Gama statlog-adm@ncc.up.pt	_D_L_SP
Research on evaluation & characterization of learning systems using various ML methods. StatLog project - A comparative studies of different ML, neural and statistical classification algorithms. Contains about 20 different algorithms and datasets.		
KDD at GTE	http://info.gte.com/~kdd/kdd-at-gte.html GTE Laboratory absweb@gte.com	_____S_
It contains predictive modeling techniques using a multi-strategy approach, combining techniques such as neural networks, decision trees, clustering, & nearest neighbor classifiers. Tools include KEFIR, CHAMPS.		

A.2 ELECTRONIC NEWSLETTERS, PAGES AND JOURNALS

E-Newsletter	Site and Moderator	Frequency
Knowledge Discovery Nuggets	http://www.kdnuggets.com/subscribe.html G. Piatetsky-Shapiro gps@kdnuggets.com	Weekly
A mailing list focusing on Data Mining and KDD research . & applications.		
Machine Learning List	http://www.ics.uci.edu/~mlearn/MLList.html M. Pazzani ml@ics.uci.edu	Monthly
A mailing list focusing on the scientific study of Machine Learning.		
AI and Statistics Mailing List	http://www.vuse.vanderbilt.edu/~dfisher/ai-stats/mailing-list.html D. Fisher ai-stats@watstat.uwaterloo.ca	Monthly
A mailing list providing information on AI and Statistics		
Datamining Page	http://home.hkstar.com/ skoo/datamine.htm Stephen Koo skoo@hkstar.com	Weekly
Articles written for Hong Kong Economic Times - I.T. Times, including product and book reviews, interviews, etc.		
DBWorld	http://www.informatik.uni-trier.de/~ley/db/dbworld.html R. Ramakrishnan dbworld@cs.wisc.edu	Weekly
A mailing list providing messages of general interest to the Database community		
Journal of Intelligent Data Analysis	http://www.elsevier.com/locate/ida A. Famali editor@ida-ij.com	Quarterly
An E-Journal to examine issues related to the research & applications of AI techniques in data analysis.		
Journal of AI Research	http://www.jair.org/ M. Wellman wellman@umich.edu	Half Yearly
The journal includes research articles, technical notes, survey & expository in A.I.		

E-Newsletter	Site and Moderator	Frequency
Journal on Data Mining & Knowledge Discovery	http://www.research.microsoft.com U. Fayyad fayyad@microsoft.com	Quarterly
The journal consolidates papers in both the research & practice of KDD, surveys of implementation techniques & application papers.		
Kluwer Machine Learning Journal	http://mlis.www.wkap.nl/ T.G. Dietterich Services@wkap.nl	Quarterly
A Journal on research in robotics, computer & information science, AI, machine learning, expert systems & cognitive science.		
IEEE Data Engineering Bulletin	http://www.research.microsoft.com/research/db/debull/ D. Lomet lomet@microsoft.com	Quarterly
An E-bulletin focusing on the design, implementation, modeling, theory and application of DB systems & their technology.		

A.3 SOME PUBLICALLY AVAILABLE TOOLS



Name	Site and Contact	Category
MLC++	http://www.sgi.com/Technology/mlc/source.html R. Kohavi mlc@postofc.corp.sgi.com	MPS - {Cls, Cst, Vis, Dev} & Pre - {Fs, Fd}
A public ML Library in C++ (main engine behind MineSet). Various inducers and pre-processing wrappers are available.		
MOBAL	http://nathan.gmd.de/projects/ml/mobal/mobal.html ML Group GMD mobal@gmd.de	MPS & ILP Learning, Knowledge Acquisition
An enhanced public version of the GMD ML and knowledge acquisition system for 1st-order KBS development. Uses multi-strategy ML methods for automated knowledge acquisition.		
TOOLDIAG	http://www.inf.ufes.br/~thomas/www/home/tooldiag.html T. Rauber thomas@inf.ufes.br	MPS {Cls}
A set of public tools (in C) for statistical pattern recognition of multivariate numerical data for classification		
DBMiner	http://db.cs.sfu.ca/DBMiner H. Jiawei han@cs.sfu.ca	MPS {Cls, Assoc, Sum, Vis}
An interactive mining tool for multiple-level knowledge in DB.		
Emerald	http://aic.gmu.edu/kiconemerald.html J. Wnek jwnek@aic.gmu.edu	MPS {Cls, Cst, Sum}
A research system of 5 different ML programs: AQ, INDUCE, CLUSTER, SPARC & INDUCE.		

Name	Site and Contact	Category
Kepler	http://nathan.gmd.de/projects/ml/kepler-englisch.html S. Wrobal stefan.wrobel@gmd.de	MPS {Cst, Cls - Dt, Nn}
A multi-paradigm, multi-purpose DM research system, extensible through a "plug-in" interface. It is builds on prev work on systems such as Mobal and Explora		
Weka (2.2)	http://www.cs.waikato.ac.nz/~ml" ML Gp at Waikato wekasupport@cs.waikato.ac.nz	MPS {Cls, Sum}
A softwarew workbench which integrates many different ML tools within a common framework and a uniform GUI.		
Sipina-W	http://eric.univ-lyon2.fr/~ricco/sipina.html ftp://hp-eric.univ-lyon2.fr/pub/sipina D. Zighed zighed@diogene.univ-lyon2.fr	Cls - Dt
It includes an ANALYSIS Module consisting of CART, Elisee, ID3 and C4.5, ChAID and SIPINA; and 2 new methods - QR_MDL (MDL principle) and WDTaiqm (Bayesian).		
OC1	http://www.cs.jhu.edu/~salzberg/announce-oc1.html ftp://ftp.cs.jhu.edu/pub/oc1 S. Salzberg salzberg@cs.jhu.edu	Cls - Dt
"Oblique Classifier 1" is a Dt induction system designed for applications where the instances have continuous feature values.		
C4.5	http://www.rulequest.com/ J.R. Quinlan quinlan@rulequest.com	Cls - {Dt, Rd}
The classical decision tree induction tool. Latest patches include a module for converting a decision tree to a set of rules. A commercial upgraded version is called C.5.		
FOIL	ftp://ftp.cs.su.oz.au/pub/ J. Quinlan quinlan@rulequest.com	Cls - Rd
FOIL reads extensional specifications of a set of relational data and produces Horn clause definitions (relationship).		
NEuroNet	http://www.neuronet.ph.kcl.ac.uk/neuronet/software/software.html C. Hinton neuronet@kcl.ac.uk	Cl - Nn
An on-line repository of information and available softwares on neural networks.		

Name	Site and Contact	Category
AutoClass C AutoClass III	http://ic-www.arc.nasa.gov/ic/projects/bayes-group/group/autoclass/autoclass-c-program.html W. Taylor taylor@ptolemy.arc.nasa.gov	Cls, Cst
AutoClass is an unsupervised Bayesian classification system that seeks a maximum posterior probability classification.		
Iris (Descartes)	http://allanon.gmd.de/and/and.html G. L. Andrienko gennady@nathan.gmd.de	Vis
Iris is a research prototyping tool supporting the automated visualization and interactive manipulation of spatially (map) referenced data. It is marketed as Descartes.		
TiMBL	http://ilk.kub.nl/software.html ILK Research Group Timbl@kub.nl	Cls - Dt, Rd
An implemention of several memory-based learning techniques for discrete data. A representation of the training set is explicitly stored in memory, and new cases are classified by extrapolation from the most similar stored cases.		
SNNS	ftp://ftp.informatik.uni-stuttgart.de/pub/SNNS/ http://www.lans.ece.utexas.edu/winsnns.html A. Zell zell@informatik.uni-stuttgart.de	Cls - Nn, Cst, Vis
A simulation environment for research on and applications of neural networks.		

Appendix B
Data Sets and Software Used in This Book

With the sources listed in Appendix A, the reader can find many data sets available for experimentation. This appendix is designed for the reader who may wish to experiment himself the methods with the data sets used in this book. We describe what are the data sets and software and where the reader can obtain soft copies of the data sets and programs.

B.1 DATA SETS

The data sets are mainly from the UC Irvine Machine Learning Repository (Merz and Murphy, 1996) if not specified. They contain different types of irrelevant, redundant, or noisy features.

- **Corral** The data was designed in (John et al., 1994). There are six binary features, A_0, A_1, B_0, B_1, I, and C. Feature I is irrelevant, feature C is correlated to the class label 75% of the time. The Boolean target concept is $(A_0 \wedge A_1) \vee (B_0 \wedge B_1)$. C4.5 chose feature C as the root. This is an example of data sets in which if a feature like C is removed, a more accurate tree will result. Among well shuffled 64 (2^6) instances, 32 instances make up the training data, another 32 instances the test data.

- **Monk1, Monk2, and Monk3** The data sets were described and used in (Thrun and et al, 1991). They were used for a performance comparison of different machine learning algorithms. Each data set has six discrete features plus one Boolean class feature. The training and test data sets are separately provided.

  - Monk1. Only three out of six features are relevant.
  - Monk2. All feature are relevant to the target concept. We want to see that a feature selection method does not remove any relevant features.
  - Monk3. Only three out of six feature are relevant, what's more is that in the training data the classes of 5% of the data were wrongly labeled. This data set is used to see how a feature selection method can handle this kind of noise.

- **Parity5+5** The target concept is the parity of five bits. These five features are interdependent in determining classes. The data set contains 10 features, of which 5 are uniformly random (irrelevant). The training set contains 100 instances randomly selected from all 1024 instances. Another independent 100 instances are drawn to form the test set. Most heuristic feature selectors

will fail on this sort of problems since an individual feature does not mean anything.

- **Parity5+2** This is a modified version of Parity5+5 by replacing its 6th and 7th columns of random values with its 1st and 2nd columns, respectively. In other words, we introduce two redundant features, in addition to the three (8th - 10th) irrelevant features.

- **Led17** This data is generated artificially by a program at the UCI repository. It contains 24 features among which the first 7 are used to display a value between 0 - 9 in the seven segment display system. The remaining 17 features are generated randomly. All the values are binary except the class which takes a value between 0 and 9 (representable in seven segments). The number of instances to be generated is determined by the user. In our experiment, we use 20000 instances.

- **Iris** There are four continuous features, the class feature has three values (Setosa, Versicolor and Virginica). The length and breadth of both petal and sepal were measured on 50 flowers of each class. In total, there are 150 instances. The data is separated into training and test sets by taking odd and even instance numbers, respectively. Two features (Petal Length and Petal Width) are relevant ones (Michie et al., 1994).

`http://www.iscs.nus.edu.sg/~liuh/Fsbook/Data` is a web site where the reader can obtain the above listed data sets. Each data set consists of X.names, X.test, X.data, and X.all where X is the name of the data set. X.names defines the type of each feature and the value of class, X.data (or training data) and X.test are obtained from X.all which is often used for cross validation.

B.2 SOFTWARE

The reader can obtain the C code by accessing the web site at `http://www.iscs.nus.edu.sg/~liuh/Fsbook/Code` in which three subdirectories (*Cla, Sel, Dis*) are about the codes for classification, feature selection and feature discretization. *The codes are given for research purposes only. All programs and supporting materials are provided as is without warranty of any kind, either expressed or implied. Neither the authors nor anyone else who has involved in the project of producing the software shall be liable for any direct, incidental, or consequential damages resulting from use of the software or documentation, regardless of the theory of liability. The authors reserve the copyrights of all the programs listed here.*

Classification. Only the program for Naive Bayesian Classifier (NBC) is provided. C4.5 (Quinlan, 1993) and SNNS (Zell and et al, 1995) can be found in their original sources (seen in Appendix A.3).

Feature Selection. The following programs are available: Focus, ABB, B&B, Relief, SFG, WSFG, WSBG, LVF, LVW, LVI, and QBB. Some programs such as Focus, Relief, and B&B are the reimplementation of the original algorithms found in the literature. For the original codes, the reader may contact the original authors. In the readme files, detailed instructions are given about compilation and execution.

Discretization. Four programs are given for trial Equal-width-intervals, Equal-frequency-intervals, ChiMerge, and Chi2. Again, details about compilation can be found in their readme files. At the time of finishing writing this book, a package of feature discretization is under construction. The reader may send email to liuh@iscs.nus.edu.sg for the status of the package.

Although no service is promised, the reader is welcome to report any bugs to liuh@iscs.nus.edu.sg for purposes of improving the programs and discussing better solutions for the future research.

References

John, G., Kohavi, R., and Pfleger, K. (1994). Irrelevant feature and the subset selection problem. In *Machine Learning: Proceedings of the Eleventh International Conference*, pages 121–129. Morgan Kaufmann Publisher.

Merz, C. and Murphy, P. (1996). UCI repository of machine learning databases. `http://www.ics.uci.edu/~mlearn/MLRepository.html`. Irvine, CA: University of California, Department of Information and Computer Science.

Michie, D., Spiegelhalter, D., and Taylor, C. (1994). *Machine Learning, Neural and Statistical Classification.* Ellis Horwood Series in Artificial Intelligence.

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

Thrun, S. and et al (1991). The monk's problems: A performance comarison of different learning algorithms. Technical Report CMU-CS-91-197, Carnegie Mellon University.

Zell, A. and et al (1995). Stuttgart neural network simulator (SNNS), user manual, version 4.1. Technical Report 6/95, Institute for Parallel and Distributed High Performance Systems (IPVR), University of Stuttgart, FTP: ftp.informatik.uni-stuttgart.de/pub/SNNS.

About the Authors

Dr. Huan Liu is a senior lecturer in the department of information systems and computer science at National University of Singapore. Before he joined NUS, he was with Telecom Australia Research Labs (1989-1994). His research interests include intelligent and telecommunications systems design, knowledge acquisition, symbolic representation of neural networks, data preprocessing, and data mining. He and his group have worked on data mining related projects such as dimensionality reduction, customer retention, feature transformation. He received his BEng in electrical engineering and computer science from Shanghai Jiao Tong University and his MSc and PhD from the University of Southern California. He is a member of the ACM, AAAI, and IEEE Computer Society (senior).

Prof. Hiroshi Motoda is a professor of the division of intelligent systems science at Osaka University's Institute of Scientific and Industrial Research. His research interests include machine learning, knowledge acquisition, scientific knowledge discovery, data mining, heterogeneous reasoning, qualitative reasoning, information filtering, and knowledge based systems. He is on the the Japanese Cognitive Science Society board of trustees and on the editorial board of Artificial Intelligence in Engineering, International Journal of Human-Computer Studies, and Knowledge and Information Systems: An International Journal. He received his Bs, Ms and PhD degrees in nuclear engineering from University of Tokyo. He is a member of AAAI, IEEE Computer Society, JSAI, JSSST, IPSJ and JCSS.

Index